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

Diplomová práce

Sparkle – rozšíření nástroje pro tvorbu SPARQL dotazů

Plzeň 2017 Bc. Josef Kazák

Prohlášení

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

V Plzni dne 15. května 2017

Bc. Josef Kazák

Abstract

Sparkle – extensions for visual SPARQL query builder

This thesis deals with development of extensions for an existing SPARQL query builder that is called Sparkle. The theory section focuses on characteristics of SPARQL and associated technologies.

The results of this master's thesis are proposals of Sparkle program extensions, whereas three of them are implemented. The practical section introduces a set of designed extensions and implementation details of realized parts. The next section describes testing of these extensions and compares them with other tools for SPARQL queries building. In conclusion, the results of this thesis are discussed and evaluated.

Abstrakt

Sparkle – rozšíření nástroje pro tvorbu SPARQL dotazů

Tato diplomová práce se zaobírá vytvářením rozšíření do již existujícího nástroje pro tvorbu SPARQL dotazů, který je nazýván Sparkle. V teoretické části textu se nachází charakteristika problematiky již zmíněného SPARQL a přidružených technologií.

Výsledky praktické části této diplomové práce jsou návrhy rozšíření programu Sparkle, přičemž z těchto konceptů jsou celkem tři realizovány. Tudíž v praktické části textu je uveden soubor navržených rozšíření a popis implementovaných částí. Dále je popsáno jejich testování a porovnání vytvořených rozšíření s jinými programy pro tvorbu SPARQL dotazů. V neposlední řadě je uvedeno zhodnocení dosažených výsledků.

Obsah

Úvod

Cílem diplomové práce (dále jen DP) je vytvoření rozšíření do již existujícího nástroje Sparkle, určeného pro tvorbu „SPARQL Protocol and RDF Query Language“ (SPARQL) dotazů. Vývoj nástroje započal v rámci své DP v roce 2014 Jan Šmucr [1] a poté ještě provedli několik změn Petr Včelák a Michel Soběhart. Na práci všech těchto vývojářů tato DP navazuje.

Úvod textu DP je zaměřen na rozbor problematiky SPARQL a dalších technologií, které souvisejí s řešeným tématem. Ovšem jedná se o poměrně rozsáhlé problematiky, tudíž je již zmíněný rozbor zaměřen pouze na nezbytné základy používané v dalších částech textu.

Po uvedených základech následuje analýza nalezených a obdobně zaměřených aplikací, jako je program Sparkle. Tato analýza mimo jiné obsahuje výčty významných funkcionalit jednotlivých aplikací, kterými Sparkle na začátku zpracovávání této DP nedisponuje.

Na základě již zmíněné analýzy a domluvy s vedoucím této DP jsou v další sekci textu uvedena vybraná prioritní rozšíření určená pro realizaci. Tento výběr následují příslušné popisy provedených implementací.

Ve zbývajících kapitolách se nalézá testování výše zmíněných rozšíření a zhodnocení dosažených výsledků. V neposlední řadě se v textu nachází porovnání realizovaných funkcionalit s programy ostatních autorů.

1 Seznámení s problematikou

1.1 Resource Description Framework

Resource Description Framework (dále jen RDF) je podle [2] systém pro popis zdrojů, který vytvořila skupina World Wide Web Consortium[1] (dále jen W3C). Zmíněným popisovaným zdrojem může být jakýkoliv objekt (např. dokument, zvukový soubor či video). RDF umožňuje standardizovaný popis zdrojů, díky čemuž jsou tyto zdroje srozumitelné nejen pro lidi, ale i pro stroje.

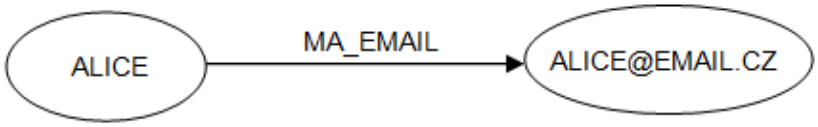
Popisy jsou ukládány do struktur, kdy jedna struktura je tzv. „RDF statement“ (v českém jazyce obvykle překládáno jako „RDF tvrzení“ či „trojice“). Tato trojice se skládá ze subjektu, predikátu a objektu. Subjekt je popisovaný zdroj, predikát je vlastností subjektu a objekt je hodnota vlastnosti zde uvedeného subjektu. Množina trojic prezentuje pojmenovaný orientovaný graf[2], kde predikáty jsou hranami a ostatní hodnoty uzly.

Pro představu, jak může taková trojice vypadat, je níže uveden příklad – viz kód 1.1. V příkladu je subjektem výraz *Alice*, predikátem slovo *ma_email* a objektem hodnota *alice@email.cz*. Inspirace pro ukázku vzata z [3].

```
Alice ma_email alice@email.cz .
```

Kód 1.1: Ukázka RDF trojice

Tento výraz lze také vyjádřit jako graf, jednou z možností znázornění je:



Obrázek 1.1: RDF trojice vyjádřená jako graf

[1] World Wide Web Consortium je organizace pro vývoj a správu webových standardů.

[2] Pojmenovaný orientovaný graf je podle [4] konečná množina uzlů, ve které jsou některé uzly propojeny hranami. Orientované hrany jsou uspořádané dvojice vrcholů (uzlů) – tj. mají přesně dané počáteční a koncové uzly.

K výše zmíněné trojici je možné přidat čtvrtou hodnotu – název grafu. Ovšem taková trojice musí být součástí daného grafu. Po přidání názvu je pak tato struktura označována jako „RDF quad“ (volně překládáno jako „čtveřice“).

Z kódu 1.1 lze pozorovat, že např. subjekt *Alice* je obecná hodnota. Pro získání jednoznačnosti jednotlivých zdrojů je v RDF využíván identifikátor „Internationalized Resource Identifier“ (dále jen IRI). IRI určuje zdroj a je (podle [5]) podobné klasickému „Uniform Resource Identifier“ (URI). Mezi IRI a URI existuje několik rozdílů. Zásadní odlišností je, že IRI využívá Unicode namísto „American Standard Code for Information Interchange“ (ASCII) znaků. Z toho vyplývá, že URI je podmnožinou IRI. Pro subjekt *Alice* z kódu 1.1 může být IRI např. výraz „`http://příklad.org/Alice`“ a URI např. „`http://priklad.org/Alice`“.

V kódu 1.2 je doplněna trojice z ukázky kódu 1.1 o právě zmíněné URI, které je zároveň i IRI. V příkladu kódu 1.2 je použit slovník „Friend of a Friend“ (dále již jen FOAF). Tento slovník slouží podle [6] pro popis osob a vztahů mezi nimi. Jmenný prostor tohoto slovníku je na `http://xmlns.com/foaf/0.1/`. Více o slovnících – viz dále v sekci 1.2.

```
http://priklad.org/Alice
  http://xmlns.com/foaf/0.1/mbox "alice@email.com" .
```

Kód 1.2: Ukázka reálné trojice s IRI

Hodnoty RDF musejí být mimo jiné zapisovány. K tomu je používán např. formát RDF/XML [7], který vychází svou skladbou ze známého „eXtensible Markup Language“ (XML) [8]. Níže je uvedena ukázka v RDF/XML (uzpůsobená na kód 1.2).

```
<?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/">
  <rdf:Description rdf:about="http://priklad.org/Alice">
    <foaf:mbox>alice@email.com</foaf:mbox>
  </rdf:Description>
</rdf:RDF>
```

Kód 1.3: Ukázka RDF/XML

V kódu 1.3 jsou vidět textové řetězce jako „foaf:mbox“. Jedná se o zkrácený zápis, kde část před dvojtečkou se nazývá „prefix“ a za dvojtečkou je vlastnost. Prefix slouží k zastupování přiřazené IRI. [3]

Kromě RDF/XML existuje mnoho dalších formátů pro zápis. Mezi nejznámější patří N-Triples, Notation3 a v neposlední řadě Turtle. Poslední jmenovaný je zobrazen v kódu 1.4 (ukázka je vytvořena na základě kódu 1.2). [9, 10, 11]

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

page:Alice foaf:mbox "alice@email.com" .
```

Kód 1.4: Ukázka Turtle syntaxe

1.2 RDF slovníky a ontologie

Při pohledu na ukázku trojice z kódu 1.1 je zřejmé, že predikát *ma_email* může být při popisování jednotlivých zdrojů chápán různě. Z toho důvodu jsou v RDF využívány tzv. „RDF vocabularies“ (v českém jazyce obvykle překládáno jako „RDF slovníky“). Tyto slovníky se skládají z množin pojmů a vlastností, které jsou vytvořeny právě k užití při vytváření dalších slovníků a ontologií. Výhodou slovníků je, že jejich části mají jasně definované významy, které jsou vždy chápány stejně. Již zmíněná „ontologie“ slouží (podle [12, 13]) k popisu určité části světa (lidského zájmu).

1.2.1 RDF slovník

Jedním ze základních slovníků pro popis zdrojů v rámci RDF je podle [7, 14, 15] „RDF slovník“. Definice tohoto slovníku se nachází v následujícím jmenném prostoru `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. Slovník obsahuje pouze elementární prvky pro popis zdrojů, mezi které se řadí např.:

- *rdf:Statement* – třída RDF trojic,
- *rdf:subject* – vlastnost určující subjekty trojic,
- *rdf:predicate* – vlastnost vyjadřující predikáty trojic,
- *rdf:object* – vlastnost, která identifikuje objekty trojic,
- *rdf:Property* – třída pro RDF vlastnosti,
- *rdf:type* – určuje, že daný zdroj je instancí nějaké třídy.

1.2.2 RDF Schema

RDF Schema (dále jen RDFS) je (podle [2, 16, 17]) jedním ze základních slovníků, který slouží zejména pro vytváření dalších slovníků atp. Jedná se o ontologický jazyk, který do RDF přidává mechanismy pro popis množin souvisejících zdrojů a vztahů mezi nimi. Tudíž je možné jej označit za nadstavbu již zmíněného RDF slovníku (viz výše v sekci 1.2.1). Kompletní definice tohoto slovníku se nachází ve jmenném prostoru `http://www.w3.org/2000/01/rdf-schema#`.

Mezi významné přidané prvky RDFS např. patří:

- *rdfs:Resource* – cokoliv v trojici je této třídy instancí,
- *rdfs:Class* – každý zdroj trojice, který je typu třída, je instancí této třídy,
- *rdfs:Literal* – třída pro textové řetězce, čísla, ... → třída pro literály.

Mezi vlastnostmi (tedy vztahy mezi subjekty a objekty) RDFS jsou:

- *rdfs:range* – stanovuje třídu objektu trojice,
- *rdfs:domain* – stanovuje třídu subjektu trojice,
- *rdfs:subClassOf* – určuje, že daná třída je podtřídou jiné nadtřídy → instance dané podtřídy jsou i instancemi již zmíněné nadtřídy,
- *rdfs:subPropertyOf* – zdroje, propojené vlastností sem spadající, jsou také propojeny nadřazenou vlastností,
- *rdfs:label* – slouží k popisu zdroje srozumitelnou formou pro člověka.

1.2.3 Web Ontology Language

Web Ontology Language (dále jen OWL) je jazyk pro vytváření ontologií, který vznikl na základě sjednocení jazyků „Ontology Inference Layer“ (OIL) a „DARPA Agent Mark-up Language“ (DAML-ONT). OWL je vytvořený konsorciem W3C a umožňuje více popisovat vztahy mezi jednotlivými zdroji než RDF slovník či RDFS. OWL je označitelný za nadstavbu právě zmíněných technologií. V době vypracovávání této DP existují celkem dvě verze. [12, 18, 19, 20, 21]

OWL první verze lze (podle [18, 19]) rozdělit celkem do tří variant, které se liší podle vyjadřovací síly:

- *OWL Lite* – jednodušší verze OWL, která podporuje uživatele, kteří potřebují klasifikační hierarchii a jednoduchá omezení,
- *OWL DL* – verze určená pro uživatele, kterým vyhovuje kompromis mezi vyjadřovací silou a výpočetní výkonností,
- *OWL FULL* – nabízí maximální expresivitu a syntaktickou svobodu RDF.

Dále pak OWL 2 je možné (podle [22]) rozčlenit na následující podjazyky:

- *OWL 2 EL* – vhodné pro ontologie, které obsahují velký počet vlastností a tříd. Výhodou je odvozování v polynomiálním čase,
- *OWL 2 QL* – navržený pro snadný přístup a dotazování se na data v RDBMS[3],
- *OWL 2 RL* – je zejména vhodný v případech, kdy jsou potřeba pro odvození systémy založené na pravidlech.

1.3 SPARQL

SPARQL je dotazovací jazyk a protokol pro RDF[4]. SPARQL byl vyvinut skupinou „Data Access Working Group“ (DAWG), která spadá pod W3C. Existují dvě verze SPARQL 1.0 a SPARQL 1.1, jejichž nejdůležitější rysy pro tuto práci jsou uvedeny dále v této části textu. [23, 24]

K popisování jsou užity rovněž příklady SPARQL dotazů, které jsou vždy aplikovány na RDF v kódu 1.5 (inspirováno z [3]). RDF definuje subjekty (znázorňující lidi) a jejich vlastnosti (e-mailové schránky – „foaf:mbox“, křestní jména – „foaf:givenName“ či příjmení – „foaf:surname“). Některé subjekty se znají, to znázorňuje vlastnost „foaf:knows“.

[3]RDBMS je relační řídící databázový systém.
[4]RDF je krátce popsáno v sekci 1.1.

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

p:Alice foaf:knows      p:Jim ;
        foaf:mbox       "alice@email.com" ;
        foaf:givenName "Alice" .
p:Jim   foaf:knows      p:Kate ;
        foaf:surname    "Doe" .
p:Kate  foaf:mbox       "kate@email.com" ;
        foaf:knows      p:Alice .
```

Kód 1.5: Ukázka RDF v Turtle syntaxi

1.3.1 SPARQL 1.0

Skupina W3C stanovila verzi SPARQL 1.0 jako konečné doporučení v roce 2008. Tato verze umožňuje čtyři způsoby dotazování – SELECT, ASK, CONSTRUCT a DESCRIBE. Samotné dotazování funguje na principu vyhledávání vzoru (daného dotazem) v úložišti. [25]

Dotaz SELECT

Dotaz SELECT (podle [3, 25]) slouží pro výběr záznamů v úložišti, které odpovídají požadavkům zapsaným v dotazu. Výstupem tohoto typu dotazu je tabulka s nalezenými záznamy.

Níže je uveden příklad SELECT dotazu využívající data, která jsou znázorněna v kódu 1.5. Hlavním účelem příkladu je vyhledání všech e-mailových adres v RDF (jsou vyhledávány hodnoty vlastností „foaf:mbox“). Možná podoba výsledku tohoto dotazu je znázorněna v tab. 1.1.

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

SELECT ?email
WHERE {
  ?subject  foaf:mbox ?email
}
```

Kód 1.6: Ukázka SELECT dotazu

Tabulka 1.1: Výsledek dotazu z kódu 1.6

email
”alice@email.com”^^xsd:string
”kate@email.com”^^xsd:string

Příklad zobrazuje základní strukturu dotazu SELECT. Struktura se skládá z prefixů, části SELECT a WHERE. Prefixy jsou uvedeny v sekci 1.2, část SELECT určuje hodnoty zobrazené ve výsledné tabulce a WHERE klauzule definuje požadovaný podgraf.

Dotaz ASK

Jedná se o podobný dotaz jako výše zmíněný SELECT. Zásadním rozdílem je výsledek, kterým zde není tabulka se záznamy z úložiště, ale logická hodnota pravda (TRUE) či nepravda (FALSE). Dotaz tedy slouží pouze ke zjištění, zda se daný podgraf v úložišti nachází či nikoliv. Pokud je vrácena hodnota TRUE, pak se hledané záznamy v úložišti nachází a pokud FALSE, tak daný podgraf přítomen není. [3, 25]

Výsledkem příkladu dotazu ASK (viz kód 1.7), kterým jsou dotazována data z kódu 1.5, je hodnota TRUE.

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

ASK
WHERE {
  ?subject  foaf:mbox "alice@email.com"
}
```

Kód 1.7: Ukázka ASK dotazu

V příkladu ASK dotazu lze pozorovat, že základní struktura je obdobou struktuře dotazu typu SELECT (viz 1.3.1). Rozdílem je, že do specifické části ASK (část před WHERE) se již žádné další hodnoty nedoplňují (oproti proměnným u dotazu SELECT).

Dotaz CONSTRUCT

Tento typ dotazu vrací výsledky obdobně jako výše zmíněný SELECT, ovšem získané záznamy jsou vráceny v podobě RDF grafu (odpovídajícího dotazu), přičemž výsledky mohou být v různých notacích (např. RDF/XML, Turtle, ...). [3, 25]

Níže uvedený příklad dotazu CONSTRUCT (inspirovaný z [3]) je aplikován na RDF z kódu 1.5. Smyslem tohoto dotazu je vytvoření „vcard:fn“ vlastnosti z „foaf:mbox“. Zmíněný „vcard“ slouží (podle [26]) k popisu lidí a organizací. Výsledek dotazu je pak znázorněn v syntaxi Turtle – viz kód 1.9.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX p: <http://priklad.org/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT {
  p:Alice vcard:fn ?name
}
WHERE {
  p:Alice  foaf:givenName  ?name
}
```

Kód 1.8: Ukázka CONSTRUCT dotazu

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

p:Alice  vcard:fn  "Alice" .
```

Kód 1.9: Ukázka výsledků dotazu CONSTRUCT

Ve struktuře dotazu v kódu 1.8 se nachází specifická část CONSTRUCT, která slouží k definování výstupního grafu. Ostatní uvedené části mají stejný význam jako u dotazu SELECT.

Dotaz DESCRIBE

DESCRIBE získává (podle [3, 25]) popis dat z úložiště, která odpovídají jeho parametrům. Výsledný popis je v RDF.

Příklad (kód 1.10) vytváří popis hledaného zdroje (v tomto případě zastoupeného proměnnou „?node“). Účelem tohoto dotazu je vyhledat zdroj a k němu získat co nejvíce informací. Výsledek takového dotazu je znázorněn v kódu 1.11 (aplikováno na RDF z kódu 1.5).

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

DESCRIBE ?node
WHERE {
  ?node  foaf:mbox  "alice@email.com"
}
```

Kód 1.10: Ukázka DESCRIBE dotazu

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

p:Alice   foaf:givenName   "Alice" ;
          foaf:knows       p:Jim ;
          foaf:mbox        "alice@email.com" .
```

Kód 1.11: Ukázka výsledků dotazu DESCRIBE z kódu 1.10

Specifickou částí uvedených klauzulí dotazu v kódu 1.10 je klauzule DESCRIBE. Ta slouží k určování výstupu dotazu.

Ostatní možnosti

Mezi významná specifika SPARQL (podle [3, 25]) patří možnost vytváření skupin trojic s určitými modifikacemi přístupu k těmto skupinám. Mezi těmito modifikacemi lze nalézt např. klíčové slovo OPTIONAL, které zapříčiní, že do výsledků daného dotazu jsou zahrnuty i výsledky, které neobsahují hodnoty specifikované v takové skupině.

Další možností SPARQL je řazení výsledků podle určitého parametru (např. proměnné). K řazení se používá klauzule ORDER BY, ke které lze přidat hodnota udávající charakter řazení. Hodnotou způsobu řazení je buď ASC nebo DESC. První ze jmenovaných pojmů (ASC) zapříčiní řazení vzestupně a druhý sestupně. Pokud není parametr klauzule (charakterizující způsob řazení) uveden, tak je obvykle řazeno vzestupně.

Kromě seskupování a řazení je možné využívat tzv. „stránkování“. Jedná se o možnost omezení výsledků na celkový počet (to se stanovuje klauzulí LIMIT) a také lze určit první stránky (záznamy), které budou přeskočeny (to se určuje díky OFFSET).

V příkladu (kód) je znázorněno použití stránkování, které určuje, že první záznam bude přeskočen a ze zbylých se zobrazí maximálně jeden výsledek. Dále je v příkladě možné zpozorovat skupinu OPTIONAL, umožňující přidání i těch výsledků, které nemají vlastnost „foaf:givenName“. V příkladu se také nachází klauzule ORDER BY, která seřadí výsledky vzestupně.

Tab. 1.2 znázorňuje výsledek příkladu z kódu 1.12 při aplikaci na RDF z kódu 1.5.

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

SELECT ?email ?name
WHERE {
  ?subject  foaf:mbox ?email
  OPTIONAL {
    ?subject foaf:givenName ?name
  }
}
ORDER BY ASC(?email)
LIMIT 1 OFFSET 1
```

Kód 1.12: Ukázka tzv. „stránkování“

Tabulka 1.2: Výsledek dotazu z kódu 1.12 při aplikaci na RDF v kódu 1.5

email	name
”kate@email.com”^^xsd:string	

1.3.2 SPARQL 1.1

V roce 2013 stanovila skupina W3C verzi SPARQL 1.1 jako konečné doporučení. Tato verze navazuje na předchozí SPARQL 1.0, ale přináší mnoho různých vylepšení. V této části textu jsou ty nejdůležitější novinky (z hlediska této práce) blíže vysvětleny (popř. doplněny příklady). [3, 27]

Příkaz INSERT DATA

Tato operace slouží k přidávání trojic do úložišť (tedy ke vkládání nových hodnot). Základní struktura příkazu je velmi jednoduchá, skládá se totiž pouze z prefixové části a klauzule INSERT DATA, ve které jsou specifikovány nové hodnoty. Tyto vkládané informace jsou strukturovány do trojic, které mohou být dále seskupovány (popř. mohou být určovány i grafy, do který mají být trojice vkládány). [27]

Příklad v kódu 1.13 znázorňuje příkaz INSERT DATA, který vkládá do RDF z kódu 1.5 novou hodnotu subjektu „p:Alice“. Přidanou hodnotou je vlastnost znázorňující příjmení (vlastnost „foaf:surname“).

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

INSERT DATA {
  p:Alice foaf:surname "Watson"
}
```

Kód 1.13: Ukázka INSERT DATA příkazu

Pokud v příkazu není zmíněn přímo graf, do kterého mají být informace přidány, pak jsou automaticky vkládány do výchozího grafu úložiště. Když jsou data vkládána do neexistujícího grafu, pak by měl být odpovídající graf automaticky vytvořen (to je do jisté míry závislé na konkrétním úložišti). [27]

Příkaz DELETE DATA

DELETE DATA operace slouží pro odebírání trojic z úložišť. Struktura dotazu je obdobná výše uvedenému příkazu INSERT DATA. Pokud jsou odstraňovány informace (podle [27]), které neexistují, pak je tato skutečnost ignorována.

Níže uvedený příklad DELETE DATA příkazu odstraňuje přidané informace kódem 1.13 (při aplikování na RDF z kódu 1.5).

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

DELETE DATA {
  p:Alice  foaf:surname "Watson"
}
```

Kód 1.14: Ukázka DELETE DATA příkazu

Příkaz DELETE/INSERT

Jedná se o operaci, kterou mohou být trojice v úložištích odstraňovány či přidávány. Výhodou tohoto příkazu je využívání identifikace RDF pomocí WHERE klauzule (jedná se o stejnou klauzuli, jako např. u dotazu SELECT – viz 1.3.1). Příkaz se skládá z následujících klauzulí:

- *WITH* – identifikuje graf, na který je příkaz aplikován,
- *DELETE* – stejné jako příkaz DELETE DATA – viz 1.3.2,

- *INSERT* – odpovídá INSERT DATA – viz 1.3.2,
- *USING* – při použití určuje, které RDF bude zpracováváno při vyhodnocování WHERE klauzule (při tomto vyhodnocování má vyšší prioritu než WITH),
- *WHERE* – definuje požadovaný podgraf.

Příklad v kódu 1.15 pozměňuje (při aplikaci na ukázkové RDF – viz kód 1.5) u subjektu *p:Alice* hodnotu vlastnosti „foaf:mbox“ z „alice@email.com“ na „alice@email.cz“.

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

DELETE {
  ?subject  foaf:mbox  "alice@email.com"
}
INSERT {
  ?subject  foaf:mbox  "alice@email.cz"
}
WHERE {
  ?subject  foaf:mbox  "alice@email.com"
}
```

Kód 1.15: Ukázka DELETE/INSERT příkazu

Díky tomu, že nejsou všechny klauzule v příkazu povinné, je možné používat tuto operaci mnoha způsoby, např. pouze k odstraňování nebo přidávání položek v úložištích. Jednou z mnoha variant je např. užití pouze klauzule DELETE a WHERE, pak se tento typ příkazu dostává do variace DELETE WHERE, kdy se příkaz principiálně podobá již zmíněnému DELETE DATA (viz sekce 1.3.2), ale klíčové slovo DATA je nahrazeno za WHERE s příslušnou klauzulí. Podrobný popis variací tohoto příkazu – viz [27].

Vnořené dotazy

Jednou z významných novinek SPARQL 1.1 je možnost vnořování dotazů. To umožňuje využívat při vyhodnocování daného dotazu (příkazu) výsledky jiného vnořeného dotazu. Vnořených dotazů v rámci jednoho příkazu může být i více, přičemž se mohou i řetězit. To znamená, že vnořený dotaz může být závislý na výsledcích jiného vnořeného dotazu. Při vyhodnocování takto komplexního příkazu (obsahujícího vnořené dotazy) se postupuje od nejvíce zanořených k těm méně zanořeným. [3]

Níže uvedený příklad (viz kód 1.16) pomocí vnořeného dotazu nalezne subjekt, který je následně předán nadřazenému DELETE/INSERT příkazu v podobě proměnné „?subject“. Předaná hodnota je již známá z kódu 1.5 – `http://priklad.org/Alice`. Nadřazený příkaz poté přidá subjektu novou vlastnost „foaf:surname“ (představující příjmení) s hodnotou „Watson“.

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

INSERT {
  ?subject foaf:surname "Watson"
}
WHERE {
  {
    SELECT *
    WHERE {
      ?subject foaf:mbox "alice@email.com"
    }
} }
```

Kód 1.16: Ukázka příkazu s vnořeným dotazem

Agregační funkce

Dalším vylepšením jsou agregační funkce. Tyto funkce umožňují zpracování výsledků dotazu před zobrazením uživateli. Agregacemi lze výsledky různě seskupovat a provádět nejrůznější statistické či aritmetické operace. [3]

Mezi agregační funkce např. patří:

- *COUNT* – vrací počet příslušných záznamů, které jsou u této funkce parametrem,
- *SUM* – jedná se o součet záznamů z parametru funkce,
- *MIN* – získá minimální hodnotu z výsledků odpovídajících parametru funkce,
- *MAX* – opak funkce MIN, tedy získá maximální hodnotu svých výsledků,
- *AVG* – jedná se o klasický aritmetický průměr, který je aplikován na hodnoty v parametru funkce,
- *GROUP_CONCAT* – vrací výčet nalezených záznamů oddělených určitým separátorem. Při aplikování této funkce se tedy musí definovat již zmíněný separátor, jehož definice může vypadat např. jako v kódu 1.17, kde jsou hodnoty proměnné „?subject“ odděleny středníkem,

- *SAMPLE* – získává náhodnou hodnotu z množiny, která je jejím parametrem.

```
(GROUP_CONCAT(?subject; separator=";") as ?subjects)
```

Kód 1.17: Ukázka GROUP_CONCAT agregační funkce

Kód 1.18 vyobrazuje užití funkce COUNT, která vrací (při aplikaci dotazu na RDF z kódu 1.5) počet vlastností subjektu `http://priklad.org/Alice`. Výsledek dotazu je znázorněn v tab. 1.3.

```
PREFIX p: <http://priklad.org/>

SELECT  (COUNT(?property) AS ?properties)
WHERE {
  p:Alice ?property  ?c
}
```

Kód 1.18: Ukázka dotazu s agregační funkcí

Tabulka 1.3: Výsledek dotazu – viz kód 1.18

properties
"3"^^xsd:integer

Agregační funkce jsou často používány v kombinaci s klauzulí GROUP BY. Tato klauzule slouží pro seskupování výsledků podle zadaných parametrů. Skupiny hodnot mohou být dále omezeny podmínkou. Samotné omezení se vytváří klauzulí HAVING s patřičným parametrem.

Příklad v kódu 1.19 znázorňuje způsob zápisu klauzulí GROUP BY a HAVING. Dotaz (aplikovaný na RDF z kódu 1.5) v tomto případě vrací počty jednotlivých vlastností (rozdělených do skupin podle významu) subjektu `http://priklad.org/Alice`, jejichž počet je větší nebo roven jedné. GROUP BY v tomto příkladu seskupuje vlastnosti (tzn. pokud by v RDF bylo u daného subjektu více vlastností „foaf:mbox“, pak by byly zařazeny do stejné skupiny). Klauzule HAVING filtruje z výsledků skupiny, jejichž počet vlastností nemá požadované množství. Výsledek dotazu je uveden v tab. 1.4.

```
PREFIX p: <http://priklad.org/>

SELECT  (COUNT(?property) AS ?properties)
WHERE {
  p:Alice ?property  ?object
}
GROUP BY ?property
HAVING ( COUNT(?property) >= 1 )
```

Kód 1.19: Ukázka dotazu s GROUP BY a HAVING

Tabulka 1.4: Výsledek dotazu – viz kód 1.19

properties
"1"^^xsd:integer
"1"^^xsd:integer
"1"^^xsd:integer

Cesty z vlastností

Novinkou je rovněž možnost (podle [3]) vytváření tzv. „property paths“ (do češtiny lze přeložit jako „cesty z vlastností“). Jelikož lze RDF hodnoty prezentovat jako graf, ve kterém jsou vlastnosti hranami, pak jedna nebo více hran na sebe navazujících vytváří (podle teorie grafů – [4]) cestu. Taková cesta se tedy skládá z navzájem navazujících vlastností a spojuje dvě různé hodnoty (uzly) v RDF.

Užití cest ve SPARQL je znázorněno v kódu 1.20. Tento dotaz vyhledává (při aplikaci na RDF v kódu 1.5) ze subjektu „p:Alice“ přes hodnoty v RDF e-mailovou adresu subjektu „p:Kate“. Výsledkem dotazu je „"kate@email.com"^^xsd:string“.

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

SELECT ?email
WHERE {
  p:Alice foaf:knows/foaf:knows/foaf:mbox ?email .
}
```

Kód 1.20: Ukázka cesty z vlastností

Negace

Mezi nové prvky SPARQL 1.1 patří tzv. „negace“. Tyto negace jsou používány ve WHERE klauzuli dotazu, kde definují, zda příslušná část se má (nebo naopak nemá) ve vyhledaném podgrafu nacházet. K tomuto se používá klíčové FILTER EXISTS a FILTER NOT EXISTS. [3]

Výsledek příkladu v kódu 1.21 (aplikovaný na RDF nacházející se v kódu 1.5) je znázorněn v tab. 1.5. Principem příkladu je vynechávání trojic, které obsahují vlastnost „foaf:mbox“. Poté jsou navíc filtrovány hodnoty, které nemají vlastnost „foaf:surname“ (filtrování se nachází ve VALUES – popis viz následující sekce).

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

SELECT  *
WHERE {
  ?subject  ?predicate  ?object
  FILTER NOT EXISTS {
    ?subject  foaf:mbox  ?email
  }
}
VALUES ?predicate { foaf:surname }
```

Kód 1.21: Ukázka užití FILTER NOT EXISTS

Tabulka 1.5: Výsledek dotazu – viz kód 1.21

subject	predicate	object
<http://priklad.org/Jim>	foaf:surname	”Doe”^^xsd:string

Mezi negace se dále řadí MINUS, která odfiltrovává hodnoty své části na základě shody v RDF. Použití je obdobné jako u FILTER NOT EXISTS, tudíž zde není nutné uvádět další ukázku.

Přímé vkládání známých hodnot

Ve SPARQL 1.1 mohou být data vkládána přímo do příkazu. K tomuto vkládání dat slouží klauzule VALUES, která určuje, jakých hodnot může daná část příkazu (proměnná) nabývat.

V níže uvedeném příkladu (kód 1.22) se nachází ukázka užití klauzule VALUES. Při aplikování dotazu na RDF v kódu 1.5 jsou vybrány pouze ty subjekty (s příslušnými vlastnostmi), které mají vlastnost „foaf:mbox“. Možná podoba výsledku dotazu – viz tab. 1.6.

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

SELECT  *
WHERE {
  ?subject  ?predicate  ?object
}
VALUES ?predicate { foaf:mbox }
```

Kód 1.22: Ukázka užití VALUES

Tabulka 1.6: Výsledek dotazu – viz kód 1.22

subject	predicate	object
<http://priklad.org/Alice>	foaf:mbox	"alice@email.com"^^xsd:string
<http://priklad.org/Kate>	foaf:mbox	"kate@email.com"^^xsd:string

Přiřazování proměnným

V neposlední řadě mezi nově přidanými funkcionalitami je možnost přiřazování „výrazů“ proměnným. Výrazem může být např. součet požadovaných hodnot, který je díky tomuto možné přiřadit konkrétní proměnné.

Přiřazování se iniciuje klíčovým slovem BIND, po tomto slově je již zmíněný „výraz“, který je následovaný klíčovým slovem AS a zakončení tvoří konkrétní proměnná. Tuto funkcionalitu je možné využívat ve WHERE klauzuli, kromě toho rovněž v SELECT části dotazu či v GROUP BY. Všechny tyto možnosti zápisu jsou znázorněny níže v kódu 1.23. Příklad v tomto kódu slouží pouze pro ilustraci možností zápisu BIND, přičemž jednotlivé varianty v příkladu vždy disponují klíčovým slovem AS.

```
PREFIX  p: <http://priklad.org/>

SELECT  (COUNT(?subject) AS ?properties)
WHERE {
  p:Alice  ?property  ?c
  BIND(?property AS ?new_property)
}
GROUP BY (p:Alice AS ?subject)
```

Kód 1.23: Ukázka možností užití BIND

Ostatní novinky

Dalšími přidanými prvky do SPARQL 1.1 (podle [27]) jsou možnosti správy RDF v úložištích. K přidaným operacím např. patří:

- *CREATE* – vytváří nový graf v úložišti,
- *DROP* – je určeno pro odstraňování grafů z úložiště,
- *COPY* – kopíruje data ze zdrojového grafu do cílového, přičemž pokud cílový graf již nějaká data obsahuje, pak jsou odstraněna a nahrazena daty ze zdrojového grafu,
- *MOVE* – slouží pro zkopírování dat ze zdrojového grafu do cílového, přičemž pokud cílový graf již nějaká data obsahuje, pak jsou odstraněna a nahrazena daty ze zdrojového grafu. Po zkopírování dat je zdrojový graf odstraněn.

Kromě správy RDF v úložištích je (podle [27]) nyní možné ve SPARQL 1.1 načítat pomocí příkazu LOAD data z RDF dokumentů (opatřených IRI) a vkládat informace do grafů v úložištích. Data ve specifickém grafu úložiště je možné mazat novým příkazem CLEAR.

Nejdůležitější novinky ve SPARQL 1.1 (z pohledu této práce) jsou zde již zmíněny. Další informace se nacházejí např. ve specifikaci [3] či v [27].

2 Analýza existujících řešení

Cílem této kapitoly je popis vlastností aplikace Sparkle. Popis je vytvořen z důvodu, že program Sparkle je hlavním předmětem zájmu této DP. Některé vlastnosti ve výpisu jsou inspirovány z [1] a jiné získány na základě vlastního testování programu.

Dalším cílem je uvedení do již existujících softwarových řešení pro tvorbu SPARQL dotazů. Součástí uvedení je i souhrn hlavních (nalezených) funkcionalit jednotlivých programů, kterými program Sparkle při započetí zpracovávání této DP nedisponuje. Souhrn je zaměřen na funkcionality, které jsou na základě popisu v [1] pro Sparkle vhodné.

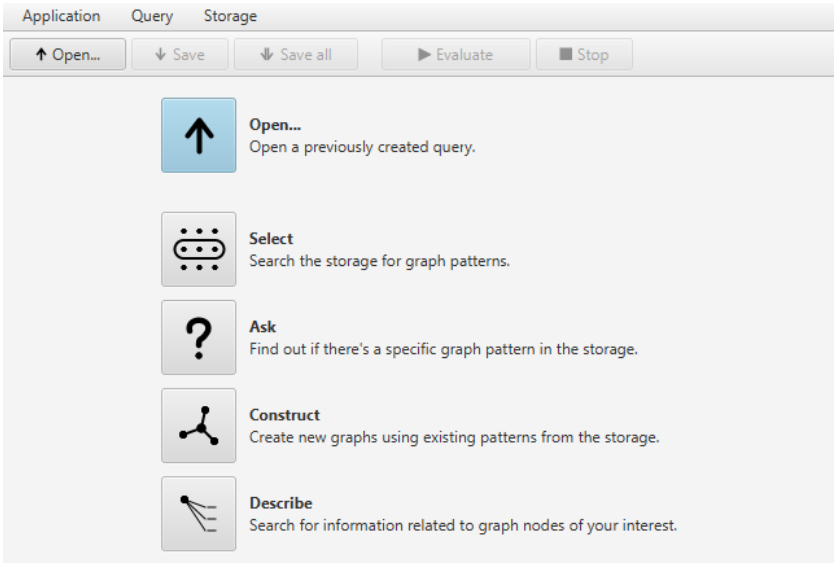
2.1 Sparkle

Program Sparkle je vyvíjen v rámci projektu „Medical Research and Education“ (dále jen MRE) na Fakultě aplikovaných věd při Západočeské univerzitě v Plzni. Aplikace slouží především k vizuální tvorbě SPARQL dotazů, mezi její význačné vlastnosti jistě patří možnost práce na lokálním i vzdáleném úložišti, intuitivní tvorba SPARQL dotazů a možnost si rozpracovaný dotaz v jakémkoliv stavu uložit do souboru a opětovně jej načíst. Domovská stránka projektu se nachází na [28].

Náhled programu Sparkle při započetí této DP se nachází na obr. 2.1, aplikace je ve verzi 1.2.0 (v podobě ze 13. 10. 2016). Vlastnostmi programu jsou:

- podpora především SPARQL 1.0 (specifika jsou zmíněna v sekci 1.3.1),
- možnost práce na lokálním i vzdáleném úložišti,
- ukládání a načítání dotazů z vlastního formátu „Sparkle query file“ (SQF),
- vlastní správa zdrojů, prefixů, funkcí (např. „ABS“ – absolutní hodnota) a datových typů (např. proměnná),
- textová i vizuální (prostřednictvím formulářů) možnost tvorby SPARQL dotazů,
- možnost vyhodnocení a zobrazení výsledků dotazů v rámci programu,
- zvýrazňování chybně zadaných částí dotazů (při využití vizuálního režimu),

- asistovaná editace (tj. kontextové nabídky, které obsahují použitelné hodnoty v rámci vytvářené části dotazu),
- obarvování syntaxe dotazu v rámci „manuální editace“ (tj. stav, kdy uživatel nechá vyhodnotit svůj dotaz – klikne na tlačítko „Evaluate“ – a po zobrazení výsledků dotazu vybere záložku „Query“),
- nad rámec SPARQL 1.0 obsahuje možnosti využití tzv. „negací“ (krátce popsány v sekci 1.3.2) a již zmíněné funkce bez agregací.



Obrázek 2.1: Úvodní obrazovka programu Sparkle při započetí této DP

2.2 Flint SPARQL Editor

Tento editor podporuje (podle [29] a na základě vlastního testování) SPARQL 1.0 a SPARQL 1.1. Nástroj disponuje asistovanou editací, interaktivní nápovědou (tj. okno obsahující položky SPARQL, které jsou podle aktuální pozice v dotazu zvýrazňovány) a v neposlední řadě je možné SPARQL dotaz v rámci editoru nechat vyhodnotit a zobrazit výsledky.

Verze programu v době zpracovávání této práce je 1.0.1. Poslední změna zdrojového kódu (v hlavní vývojové linii) je datována na 17. 10. 2012. Nástroj je dostupný na [29].

Program disponuje následujícími nalezenými funkcionalitami, kterými při započetí prací na této DP Sparkle neoplývá:

- podpora SPARQL 1.1 Query (specifikace – viz [3]),
- interaktivní nápověda (tabulka s možnými klíčovými slovy SPARQL, která zvýrazňuje použitelná slova),
- výstupy dotazů je možné zobrazovat jako tabulku, čistý text či ve formátech „JavaScript Object Notation“ (JSON) a XML,
- disponuje tlačítky pro vrácení a obnovení provedených změn,
- možnost připojení ke vzdálenému úložišti (přes tzv. „endpoint“[1]).

2.3 YASGUI

YASGUI je webový nástroj pro vytváření a vyhodnocování SPARQL dotazů. Nástroj vyniká např. podporou SPARQL 1.1 či možností získané výsledky ukládat na pevný disk ve volitelném formátu. Popis (včetně níže uvedených výhod) je vytvořen na základě vlastní analýzy a [31].

Nástroj je v době zpracovávání tohoto textu ve verzi 2.6.1 (poslední změna – v hlavní vývojové linii – je provedena dne 16. 4. 2017) a je dostupný na [32].

Tento nástroj má (kromě již jmenovaných rysů) oproti Sparkle navíc:

- zobrazování výsledků dotazů nejen v tabulkové formě, ale také ve formátu přijatém z úložiště, jako kontingenční tabulku, v „Google Chart“[2] atd.,
- fulltextové vyhledávání ve výsledcích (u tabulkového zobrazení),
- možnost ukládání výsledků dotazů v jiných formátech než u Sparkle, mezi tyto formáty patří např. JSON, „Tab-Separated Values“ (TSV) či XML,
- katalog mnoha vzdálených úložišť pro vykonávání dotazů,
- historii provedených změn (umožňuje postupně vracet provedené změny ve zpracováváném příkazu),
- kontextové nabídky u každé záložky (nabídky se zobrazí po kliknutí pravým tlačítkem myši na příslušnou záložku).

[1] Endpoint je (podle [30]) místo, kam mohou být odesílány dotazy a odkud mohou být získávány výsledky těchto dotazů.

[2] Google Chart je interaktivní webová služba pro vytváření grafů z uživatelských dat.

2.4 iSPARQL

Webový nástroj pro vytváření a spouštění SPARQL dotazů. Práce na dotazech mohou probíhat jak vizuálně (přidáváním uzlů a hran do grafu), tak v klasickém textovém editoru. Tento nástroj je především využíván jako webová vrstva v mnoha standardizovaných RDF úložištích. Uvedení je vytvořeno na základě vlastní analýzy a popisu v [33].

Nástroj iSPARQL je při vypracovávání této práce ve verzi 1.30. Poslední úprava (v hlavní vývojové linii) je datována na 21. 3. 2014 a program je dostupný na [34].

Program disponuje oproti aplikaci Sparkle následujícími vlastnostmi:

- možností grafického vytváření SPARQL dotazů,
- interaktivní nápovědou (obdoba již zmíněné v sekci 2.2),
- zvolitelnými způsoby zobrazování výsledků, mezi které např. patří:
  - jako SVG graf (SVG je podle [35] obrazový formát),
  - v režimu „Navigator“ (při tomto způsobu zobrazování jsou získané záznamy odkazy, při jejichž užití se zobrazí příslušné vlastnosti),
  - případně (v závislosti na získaných hodnotách) lze výsledky zobrazit jako obrázek, na mapě či jako data tak, jak byla získána.

2.5 Gosparqled

Gosparqled je (podle [36, 37]) knihovna zajišťující asistovanou editaci. Tato editace ovšem vyhledává kontextově závislé vlastnosti (vlastnosti spojené s právě zpracovávaným subjektem daného dotazu) – bližší popis se nachází níže.

Knihovna je určena pro použití v dalších nástrojích jako např. YASR[3]. Knihovna je dostupná na [37], kde poslední změna (v hlavní vývojové linii) je provedena 13. 12. 2014.

Hlavní výhodou Gosparqled oproti programu Sparkle je již zmíněné kontextově závislé vyhledávání, které má několik dalších funkcionalit, mezi které patří:

- automatické vyhledávání tříd, do kterých daný subjekt může patřit,

[3]YASR je (podle [38]) část z YASGUI nástrojů (popis YASGUI se nachází v sekci 2.3.)

- vyhledávání vlastností, které jsou s daným subjektem spojeny,
- doporučování cest v grafu, které znázorňuje vztah mezi dvěma subjekty,
- prohledávání grafu je možné s nastavením určité délky cest,
- vyhledávání subjektů, které nejlépe odpovídají zadaným vlastnostem a jejich hodnotám.

2.6 Gruff

Následující popis je vytvořen na základě [39] a vlastní analýzy. Jedná se o nástroj určený k prohlížení dat, správě dotazů a upravování úložišť, která jsou postavená na AllegroGraph databázích. Gruff umožňuje, kromě zobrazování informací v klasických tabulkách, vizualizaci RDF do podoby grafů. SPARQL dotazy mohou být vytvářeny v textovém i grafickém editoru. Program podporuje SPARQL 1.1 a jazyk PROLOG.

Gruff je v době vypracovávání tohoto textu ve verzi 6.4.3 (datum vydání nezjištěno) a je dostupný na [39].

Gruff disponuje oproti Sparkle následujícími vlastnostmi:

- obsahuje tzv. „tooltip“ (okno s tipy pro efektivnější využívání nástroje),
- umožňuje vizualizaci výsledků dotazů do grafů či přidání těchto záznamů do již existujících grafů,
- disponuje vizuální tvorbou SPARQL dotazů,
- vizualizace má mnoho různých rysů, mezi kterými je např. zobrazování všech uzlů pouze „labely“ („label“ je vlastnost obsahující název dané části grafu, který je pro lidi snáze čitelný),
- disponuje tlačítky pro vrácení a obnovení provedených změn.

2.7 Twinkle: SPARQL Tools

Tento program slouží (podle [40] a vlastního testování) pro tvorbu a spouštění SPARQL dotazů. Výhodou aplikace je import RDF ze souboru či možnost připojení ke vzdálenému úložišti. Výsledky dotazů mohou být zobrazeny jako text či tabulka.

Aplikace je v době vytváření tohoto textu ve verzi 2.0 (datum vydání nezjištěno), podporuje SPARQL 1.0 a ke stažení je na [40].

Program disponuje oproti Sparkle následujícími vlastnostmi:

- načítáním SPARQL dotazů z textových souborů,
- tlačítky pro vrácení a obnovení provedených změn,
- možností připojení k (přednastaveným) vzdáleným úložištím.

2.8 dotNetRDF

Popis je vytvořen na základě [41, 42] a vlastního testování. Jedná se o knihovnu, jejíž cílem je zajištění efektivní a snadné práce s RDF na platformě .NET. Knihovna poskytuje podporu pro mnoho typů úložišť (např. AllegroGraph, Fuseki, Sesame či Virtuoso). V rámci knihovny je podporován SPARQL 1.1. Při psaní tohoto textu je dotNetRDF ve verzi 1.0.12 (vydaná dne 4. 8. 2016).

Ze stránek [42] je možné stáhnout knihovnu s dalšími nástroji, mezi nimiž nechybí aplikace „dotNetRDF Store Manager“, která poskytuje grafické rozhraní pro správu úložišť a možnost dotazování. Kromě „dotNetRDF Store Manager“ se mezi nástroji nachází program „SPARQL GUI“, ve kterém je možné vytvářet a vyhodnocovat SPARQL dotazy. Dále uvedené funkcionality se týkají právě zmíněných nástrojů.

Mezi funkcionalitami navíc v programu „SPARQL GUI“ se řadí:

- podpora SPARQL 1.1,
- načítání SPARQL dotazů z textových souborů,
- ukládání a zobrazování logů (záznamů o provedených úkonech),
- možnost ukládání výsledků dotazů v uživatelsky volitelných formátech.

U aplikace „dotNetRDF Store Manager“ jsou funkcionalitami navíc (oproti programu Sparkle):

- podpora mnoha typů úložišť, kromě již zmíněných v sekci 2.8, to jsou: Stardog, 4store či In-Memory,

- možnost vytváření příkazů i ve SPARQL 1.1,
- uchovávání nedávno použitých úložišť či RDF souborů,
- export výsledků dotazů v mnoha formátech – např. NQuads, N-Triples či Turtle.

3 Návrhy rozšíření

V předchozí kapitole jsou shrnuty funkcionality programů, kterými aplikace Sparkle na začátku zpracovávání této DP nedisponuje. Jelikož je zhotovení několika rozšíření hlavním účelem této DP, tak se dále v této části textu nachází podrobnější nástin toho, která rozšíření jsou pro Sparkle vhodná, případně v jaké podobě a zda jsou proveditelná. Uvedená rozšíření jsou zvolena na základě shrnutí funkcionalit z předešlé části textu, popisu programu Sparkle v [1] a konzultace s vedoucím této DP.

3.1 Načítání SPARQL dotazů ze souborů

Načítání SPARQL dotazů z textových souborů se jeví jako užitečná funkcionalita, která by znatelně zefektivnila práci s programem Sparkle. Z uvedených aplikací v kapitole 2 takovým disponuje načítáním např. „Twinkle: SPARQL Tools“ (viz popis výhod tohoto programu v sekci 2.7). Implementací této možnosti by byl zvýšen celkový potenciál aplikace Sparkle, jelikož při současném stavu musí uživatel celý SPARQL dotaz vytvářet vždy manuálně.

Pro implementaci této funkcionality je zřejmé, že je nutná kontrola syntaxe dotazů (kontrola správného zápisu). K tomuto je vhodné využít nástroj, kterému by byla předána SPARQL gramatika, na jejíž základě by pak probíhala kontrola načítaných dotazů. Mezi tyto nástroje se řadí např.:

- *ANTLR* – celý název je následující „ANother Tool for Language Recognition“. Tento nástroj dokáže z gramatiky vygenerovat tzv. „parser“ (program, který kontroluje syntaktickou správnost svého vstupu) v programovacím jazyce Java (v tomto jazyce je vytvořen i Sparkle). Tento „parser“ pak zvládá vytváření tzv. „syntaktického stromu“ (jednotlivé části SPARQL dotazu by byly rozděleny do stromové struktury, kde vnitřní uzly by byly operátory a listy operandy – viz [43]) a navíc disponuje funkcionalitami pro jejich procházení. ANTLR je v době psaní této části textu ve verzi 4.5.3. Popis je vytvořen na základě [44, 45, 46]. Nástroj lze stáhnout na [44],
- *JavaCC* – znění celého názvu je „Java Compiler Compiler“. Jedná se o program (podle [47]) pro vytváření „parserů“ v Javě. Do JavaCC je předávána gramatika, na jejíž základě se vytváří přeložitelný zdrojový kód, který je následně použitelný v dalších aplikacích. Součástí nástroje je JJTree, který slouží k vytváření „syntaktického stromu“ jako u výše zmíněného ANTLR.

Nástroj JavaCC je vytvářen v programovacím jazyce Java, v době zpracovávání této části textu je ve verzi 7.0.2 a je dostupný na [47],

- *SableCC* – jedná se rovněž o parser generátor, který vytváří zdrojový kód v Javě. Podobně, jako JavaCC či ANTLR generuje „syntaktický strom“ a disponuje možnostmi pro jeho procházení. V době vypracovávání tohoto dokumentu je ve verzi 4 a stažitelný je z [48] (odkud jsou získány informace uvedené popisu),
- *Coco/R* – je obdobou předchozích nástrojů, existuje několik verzí, z nichž některé podporují Javu. Přesná verze v době psaní textu není zjištěna. Popis je inspirován z [49] (odkud je možné nástroj i získat).

Dále je nutné program Sparkle upravit tak, aby dokázal přijaté a „parserem“ zkontrolované dotazy zpracovávat. Implementace těchto částí vyžaduje poměrně rozsáhlé změny v programu Sparkle, a to především ve zdrojových kódech jednotlivých klauzulí dotazů. V rámci tohoto mechanismu by také musely být rozpoznávány „datové typy“ načítaných hodnot apod.

3.2 Podpora SPARQL 1.1

Vzhledem k účelu, za jakým byl program Sparkle vytvořen (tedy co nejsnazší tvorba SPARQL dotazů a prohlížení obsahu úložišť), je vhodné ze SPARQL 1.1 přidat zejména prvky, které přímo nemanipulují s celými grafy v úložištích.

Mezi žádoucí prvky patří možnost využívání agregačních funkcí, filtrování výsledků vkládáním relevantních hodnot či v neposlední řadě vnořené dotazy. Přidáním těchto a dalších vybraných možností SPARQL 1.1 (viz dále v této části textu) se výrazně zvýší užitečnost aplikace. [3, 27]

Z uvedených programů v kapitole 2 disponuje podporou SPARQL 1.1 např. YASGUI (viz sekce 2.3).

3.2.1 Agregační funkce

Mezi prioritní novinky SPARQL 1.1 (vhodné pro přidání do programu Sparkle) patří agregační funkce, jejichž krátký popis se nachází v sekci 1.3.2. Přidání této funkcionality vyžaduje zásadní úpravu zejména části dotazu SELECT, ve které jsou vybírány proměnné, jejichž hodnoty tvoří výsledek daného dotazu. Kromě

úpravy již zmíněné části by bylo nutné pozměnit i ukládání a načítání dotazu z SQF souborů a mnoho dalších změn.

Kromě úpravy klauzule dotazu SELECT se s agregacemi pojí klauzule GROUP BY a HAVING, jejichž popis se nachází v části textu 1.3.2. Tyto klauzule by bylo nutné zcela vytvořit a přidat je i do ostatních (vhodných) typů dotazů a samozřejmě (kvůli správnému zpracovávání) upravit i ostatní části programu Sparkle (týkající se těchto klauzulí).

3.2.2 Vnořené dotazy

Významnou částí SPARQL 1.1 jsou vnořené dotazy, jejichž krátký popis se nachází v sekci 1.3.2. Implementace této funkcionality by byla proveditelná několika způsoby, ale každý z nich vyžaduje poměrně složité úpravy programu Sparkle.

Možným způsobem implementace těchto vnořených dotazů je úprava kompozice prvků (rozvržení grafických elementů dotazů ve Sparkle) již existujících dotazů, a to takovým způsobem, který by umožňoval přehledné vytváření vnořených dotazů přímo mezi stávajícími grafickými prvky. V praxi by se jednalo např. u dotazu SELECT o změnu ve WHERE klauzuli tak, aby přímo v ní bylo možné skládat celé vnořené dotazy. U tohoto řešení je zřejmé, že udržení přehlednosti mezi prvky by bylo velmi problematické. Obzvláště pak v případě, pokud by obsahoval vnořený dotaz další vnořené dotazy.

Zřetelně vhodnějším způsobem realizace je vytvoření zástupné komponenty, která by byla vkládána do WHERE částí jednotlivých dotazů. Komponenta by poté udržovala vztah mezi dotazem a vnořeným dotazem, který by byl ve své vlastní záložce (ve svém okně). Toto řešení by sice odstranilo problém s přehledností, který je uveden u předchozího návrhu, ale vyvstaly by zde další problematické body. Mezi problémy by patřilo např. vytvoření zcela nového okna (typu záložky), které by umožňovalo vytváření vnořeného dotazu. Kromě tohoto problému by bylo nutné vyřešit např. provázání jednotlivých dotazů, předávání proměnných mezi nimi atd.

3.2.3 Příkazy DELETE DATA, INSERT DATA a DELETE/INSERT

Do programu Sparkle by bylo vhodné implementovat dotazy DELETE DATA a INSERT DATA. Popis těchto dvou typů dotazů se nachází v sekci 1.3.2. V případě přidávání takových dotazů by bylo nutné vytvořit prostředí pro jejich zpracovávání, a to obdobným způsobem, jako jsou implementovány již existují typy dotazů

v aplikaci Sparkle (tedy např. SELECT). Kromě prostředí (zahrnující mimo jiné ukládání a načítání dotazů ze souborů) by bylo nutné upravit (či zcela vytvořit) mechanismus vyhodnocování těchto příkazů. V neposlední řadě by bylo nutné zakomponovat tyto příkazy mimo jiné do úvodní obrazovky programu Sparkle (tzn. vytvořit nová tlačítka) pro možnost vytvoření nového příkazu.

Pro operaci DELETE/INSERT by platily obdobné náležitosti jako např. u příkazu DELETE DATA. Operace je popsána v sekci 1.3.2. Ovšem jedná se o komplexnější příkaz, který obsahuje např. klauzule WITH a USING. Takové klauzule se při započetí práce na této DP v aplikaci Sparkle nevyskytují, tudíž by bylo nutné je v rámci implementace příkazu DELETE/INSERT rovněž vytvořit.

3.2.4 Ostatní novinky SPARQL 1.1

Mezi další vhodné novinky SPARQL 1.1 k realizaci patří klauzule VALUES, jejíž popis se nachází v sekci 1.3.2. Přidání této funkcionality by vyžadovalo vytvoření nové nového prostředí v aplikaci Sparkle, které by umožňovalo vytváření klauzulí VALUES. Prostředí by muselo mít mimo jiné modifikovatelné uspořádání grafických prvků. To z důvodu, že se jedná o komponentu, která by byla součástí základních struktur některých typů SPARQL dotazů (obdobně jako např. klauzule FROM) a také WHERE klauzule.

Užitečným prvkem SPARQL 1.1 je i řetězení vlastností, jehož popis se nachází v sekci 1.3.2. Implementace tohoto rozšíření by znamenala modifikaci již existující asistované editace takovým způsobem, aby probíhalo „napovídání“ i při vytváření delší cesty z vlastností. Kromě toho by byla nutná změna v mechanismu zvýrazňování, aby cesta nebyla brána jako chyba.

3.3 Kontextové nabídky záložek

Kontextové nabídky jsou zpravidla zobrazovány při kliknutí na daný prvek pravým tlačítkem myši. V případě aplikace Sparkle by byla příslušným prvkem záložka obsahující SPARQL dotaz. V takové nabídce by byly např. funkcionality pro zobrazení či uzavření dané záložky či pro uložení změn. Podobnými kontextovými nabídkami disponuje např. YASGUI (viz sekce 2.3).

Tyto nabídky by nemělo být příliš složité do Sparkle implementovat. To z důvodu, že nástroj JavaFX (podle [50]), ve kterém je vytvořené grafické rozhraní programu, takové nabídky nativně podporuje.

Vzhledem k povaze Sparkle se nabízí do těchto kontextových nabídek přidat možnost na obarvování příslušných záložek uživatelsky volitelnou barvou. Tato funkcionalita by výrazně zvýšila přehlednost při práci v programu Sparkle na více dotazech současně.

3.4 Vyhledávání vlastností subjektů

Na základě inspirace z Gosparqled (viz sekce 2.5) by bylo vhodné do aplikace Sparkle zakomponovat napovídání vlastností (predikátů s relevantními objekty), které by byly spojeny s danými subjekty v úložištích.

Toto rozšíření by obnášelo vytvoření nových oken pro získání hloubky prohledávání (délky cest) a pro výpis získaných vlastností. Samotné prohledávání úložiště by muselo být vázané na aktuální podobu SPARQL dotazů (kvůli správné identifikaci subjektů) a zrušitelné (kdyby načítání trvalo příliš dlouho apod.). K získaným cestám z vlastností by bylo vhodné přidávat i hodnoty, které se nachází na konci daných cest. A po zvolení jedné cesty z vlastností ve výpisu, by mohla být automaticky cesta i s příslušnou hodnotou doplněna do příslušné trojice ve Sparkle.

3.5 Doplňování názvů vlastností

Vhodným rozšířením, které by více zpřístupnilo vytváření dotazů uživatelům, kteří nejsou příliš znalí SPARQL, by bylo zobrazování tzv. „label“ jednotlivých vlastností namísto aktuálních výpisů názvů ze slovníků. „Label“ hodnoty jsou popisy vlastností určené pro lidi. Tudíž by se v nápovědě programu Sparkle (při nastavení datového typu „Prefixed name“) pro slovník FOAF nezobrazovalo „foaf:mbox“, ale „personal mailbox“ (volně přeloženo jako „osobní e-mail“ – hodnota převzata z [6]).

Tato úprava by vyžadovala modifikaci ukládání RDF slovníků v rámci programu Sparkle tak, aby nebyly ukládány pouze názvy vlastností, ale i jejich hodnoty „rdfs:label“. Dále by muselo být upraveno načítání těchto vlastností v existující asistované editaci (nápověda poskytující informace nejen o vlastnostech). V neposlední řadě by musel být implementován mechanismus, který by zajišťoval (po zvolení daného názvu vlastnosti) doplnění hodnot do editovaného dotazu.

3.6 Fulltextové vyhledávání

Touto notoricky známou funkcionalitou disponuje hned několik programů z kapitoly 2, mezi které patří např. YASGUI (viz část textu 2.3). Toto vyhledávání by výrazně zjednodušilo procházení výsledků SPARQL dotazů typu SELECT.

Samotná implementace by spočívala v modifikaci zobrazovaní výsledků do tabulky takovým způsobem, aby bylo možné jednotlivé záznamy dle zadaného filtrovacího řetězce zobrazovat či skrývat. Kromě tohoto mechanismu by bylo nutné do okna výsledků dotazu SELECT přidat pole pro vkládání textových řetězců (určených k filtrování).

3.7 Vizuální skládání SPARQL dotazů

Vizuální tvorba SPARQL dotazů, jako je např. v iSPARQL (viz sekce 2.4), by byla pro program Sparkle přínosem. Realizace tohoto rozšíření by nejspíše znamenala rozsáhlý zásah do aplikace, jelikož by bylo potřeba vytvořit celý modul pro vizualizaci, navzájem provázat části dotazů mezi již existující editací dotazů a touto vizualizací. V neposlední řadě by bylo nutné vytvořit další popisné tabulky pro zvolené prvky ve vizualizaci atd.

3.8 Možnost vrácení a obnovení změn

Jedná se o běžně známou funkcionalitu napříč editory (má ji např. již několikrát uvedený YASGUI), která slouží pro vrácení provedených změn či pro obnovení posledního stavu. Tato funkcionalita (podle [1]) by měla být v programu Sparkle proveditelná využitím rozhraní „Changeable“, s jehož pomocí by se měl program Sparkle vždy dozvědět o provedené změně a v případě potřeby adekvátně reagovat.

3.9 Ostatní funkcionality

Mezi další přínosná rozšíření programu Sparkle se řadí např. okno s tipy pro efektivnější používání programu, tedy tzv. „tooltip“. Tímto oknem disponuje např. Gruff (viz sekce 2.6). Okno by mělo být automaticky zobrazováno po spuštění aplikace s již zmíněnými tipy. Implementace této funkcionality by měla spočívat ve vytvoření nového okna s možností zrušení automatického zobrazování po spuštění

programu Sparkle. Z toho vyplývá, že by se tato uživatelská volba o zobrazování okna musela ukládat. Dále by musela být vytvořena sada rad, jejíž obsah by byl zobrazován.

Notoricky známou funkcionalitou mezi editory je seznam nedávno používaných souborů. Program „dotNetRDF Store Manager“ dokáže ukládat nejen nedávno použité soubory, ale také v poslední době navázaná připojení k úložištím. Dle povahy Sparkle by bylo vhodné přidat nabídku na nedávno použité soubory (obsahující SPARQL dotazy). Při implementaci by se jednalo především o přidání nové položky do některé z existujících nabídek aplikace Sparkle a ukládání seznamu těchto souborů.

Vhodným rozšířením je ukládání tzv. „logů“ (záznamů o činnostech). Tyto logy by mohly být generovány např. při spuštění vyhodnocování dotazu či při jeho vytváření. V případě vzniku nějaké chyby by pak mohly napomoci s diagnostikou. Vytvoření tohoto rozšíření by vyžadovalo mechanismus ukládání logů a přizpůsobení jednotlivých částí Sparkle tak, aby byly schopny logy generovat.

Dalším možným rozšířením Sparkle je zobrazování výsledků SELECT dotazů nejen v tabulce, ale např. jako text či ve formátu XML (jako je tomu např. u YASGUI – viz část textu 2.3). Toto rozšíření by vyžadovalo úpravy záložky pro výsledky dotazů typu SELECT. Mezi úpravami by se muselo vyskytovat přidání nových způsobů zobrazování získaných záznamů k již existující tabulkové prezentaci.

4 Implementace rozšíření

Po dohodě s vedoucím této DP jsou pro další implementaci zvolena následující rozšíření:

- Načítání SPARQL dotazů z textových souborů, návrh tohoto rozšíření se nachází v sekci 3.1,
- Podpora vybraných částí SPARQL 1.1, zvolené novinky odpovídají návrhu v části 3.2,
- Vyhledávání vlastností subjektů – návrh této funkcionality je v sekci 3.4.

Cílem této kapitoly je vysvětlení samotné realizace výše zmíněných rozšíření. Jednotlivé popisy jsou rozděleny do vlastních sekcí, ve kterých jsou poznamenány pouze nejdůležitější části řešení. Další podrobnosti jsou uvedeny v komentářích zdrojového kódu programu Sparkle, který se nachází na přiloženém datovém nosiči.

4.1 Načítání SPARQL dotazů ze souborů

4.1.1 Analýza vstupního SPARQL dotazu

Popis analytického nástroje a způsobu užití

Jak již bylo zmíněno v sekci 3.1, pro implementaci tohoto rozšíření je vhodné využít nástroj, který zajistí kontrolu načítaného SPARQL dotazu. V již zmíněné sekci jsou mimo jiné uvedeny příklady těchto nástrojů, z nichž byl využit první jmenovaný – nástroj ANTLR. Tento nástroj (přesněji ANTLR 4) je vytvářen pod tzv. „BSD licencí“, jejíž užití je podle [51] pro program Sparkle vhodné. ANTLR je zvolen především kvůli již zhotovené gramatice SPARQL 1.1, neustále probíhajícímu vývoji, poměrně volné licenci užití – „BSD licenci“, dobré dokumentaci a mnoha existujícím příkladům o užívání jeho částí. [44, 45, 46]

Gramatika SPARQL 1.1 pro nástroj ANTLR 4 je získána z projektu „SPARQL Pretty Printer 4“. Projekt je dostupný na [52] a k užívání (včetně SPARQL gramatiky) pod licencí „Apache License 2.0“. Tato licence je podle [53] pro aplikaci Sparkle rovněž vhodná. Samotná gramatika byla získána z uvedeného projektu, který se nacházel ve verzi 4-1.0 (vydaná 23. 2. 2013). Tato gramatika se skládá

celkem ze dvou souborů, kdy první je určen pro lexikální analýzu (v projektu programu Sparkle se jedná o soubor „sparkle/src/main/antlr4/SparqlLexer.g4“) a druhý slouží pro syntaktickou analýzu, která obvykle následuje po lexikální (pokud nejsou nalezeny chyby již při první kontrole). Soubor pro syntaktickou analýzu v projektu je „sparkle/src/main/antlr4/SparqlParser.g4“.

Lexikální analýza rozděluje vstupy na tzv. „lexémy“ (základní jednotky slovní zásoby daného jazyka), ty jsou posléze převedeny na „tokeny“ („token“ je základní podoba daného slova) a předány k syntaktické analýze. Při syntaktické analýze je kontrolována struktura vstupu a vytvářen syntaktický strom (krátký popis stromu se nachází v sekci 3.1). Tento syntaktický strom je dále v programu Sparkle procházen, přičemž jsou podle hodnot a aktuálního kontextu dynamicky vytvářeny klauzule daného dotazu se získanými hodnotami. [43, 54]

Procházení syntaktického stromu je možné v aplikaci Sparkle realizovat mnoha způsoby. Jednou z možností je využití částí nástroje ANTLR, kterými jsou „visitor“ a „listener“. Neposlední možností je implementace vlastního procházení syntaktického stromu např. algoritmem prohledávání grafu do hloubky (DFS) či do šířky (BFS). [45, 55]

„Listener“ slouží k procházení syntaktického stromu metodou DFS, ovšem při prvotní implementaci načítání SPARQL dotazů se nepodařilo tuto část nástroje ANTLR efektivně přizpůsobit stávající architektuře programu Sparkle. Přesněji řečeno, bylo nutné mít v různých částech Sparkle obsažený tento „listener“ s různými implementacemi metod a k tomu udržovat aktuálního kontext v rámci načítaného dotazu z důvodu, že některé části Sparkle jsou opakovaně užívány pro různé účely s odlišnými pravidly, což je nutné ošetřit. Tato skutečnost způsobovala při využití „listener“ mnoho problémů. Užití již zmíněné části „visitor“ má z pohledu implementace ve Sparkle stejné potíže jako „listener“.

V aplikaci Sparkle se téměř ve všech částech načítání SPARQL dotazů z textových souborů využívá algoritmus DFS. DFS při prohledávání grafů (syntaktických stromů) funguje na principu navštívení dosud nezpracovaného následovníka aktuálního uzlu. Při navštívení je uzel zpracován a takto algoritmus pokračuje dokud existuje nějaký nenavštívený následovník. Když neexistuje, tak se algoritmus vrací po navštívených uzlech a pokud na nějaké pozici ve stromu nalezne dosud nenavštívený uzel, tak jej začne zpracovávat již zmíněným způsobem. Vlastní implementace DFS algoritmu umožňuje přizpůsobení mechanismu načítání existující architektuře programu Sparkle, možnost udržování informací o aktuálním kontextu, definování pouze nutných částí načítání v rámci dané klauzule atd.

Algoritmus BFS je ve Sparkle použit pouze u určitých klauzulí, ve kterých je (v závislosti na struktuře syntaktického stromu dané klauzule) výhodnější než DFS. Mezi klauzule (využívající princip algoritmu BFS) patří např. BIND (viz soubor

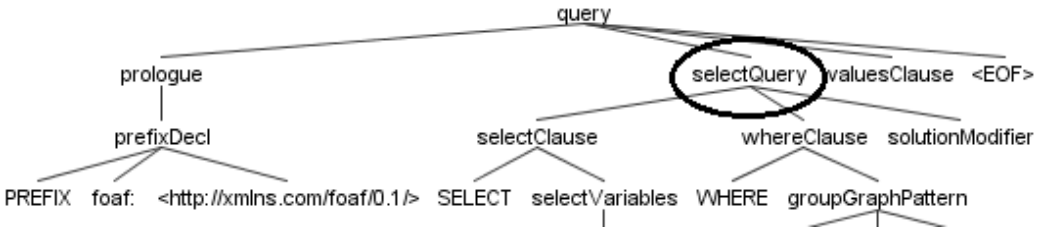
v projektu „cz.zcu.mre.sparkle.gui.query.BindPane.java“ – metoda „load“). BFS funguje na principu navštívení všech sousedů aktuálního uzlu a poté pokračuje navštěvováním všech sousedů již navštívených sousedních uzlů.

Průběh analýzy ve Sparkle

Po kliknutí na tlačítko „Open...“ v aplikaci Sparkle (popis spuštění tohoto mechanismu se nachází v příloze A) a po zvolení textového souboru obsahujícího SPARQL dotaz (pro ilustraci bude nadále používán příklad SELECT dotazu, který je uveden v kódu 1.6) se spustí mechanismus vytváření prostředí. Zmíněný mechanismus před přidáním nové záložky (popř. záložek) s načítaným dotazem nejprve vytváří syntaktický strom pomocí nástroje ANTLR (viz sekce 4.1.1). Příklad syntaktického stromu dotazu z kódu 1.6 je znázorněn v příloze B.

4.1.2 Vytváření prostředí načítaného dotazu

Z vytvořeného syntaktického stromu (viz 4.1.1) se nejprve zjistí typ dotazu, na jehož základě je zvolena relevantní metoda načítání. Typem dotazu může být např. SELECT či ASK. Podrobnější informace o těchto dotazech se nachází v sekci 1.3.1. Zjišťování druhu dotazu probíhá na základě procházení syntaktického stromu, kdy jsou vyhledávány známé vnitřní uzly, které značí daný typ dotazu. Pro SELECT je to uzel „selectQuery“ (viz obr. 4.1, který je vytvořen na základě dotazu z kódu 1.6). Tyto klíčové uzly mají vždy podobnou strukturu, např. pro ASK je to „askQuery“ či pro dotaz DESCRIBE „describeQuery“. Pokud je načítán nepodporovaný SPARQL dotaz, tak je zobrazena příslušné chybové hlášení. Algoritmus rozpoznávání typu dotazu se nachází ve třídě „QueryPane“ z balíku tříd „cz.zcu.mre.sparkle.gui.query“.



Obrázek 4.1: Znázornění uzlu indikujícího typ dotazu SELECT (pro vytvoření obrázku byl využit program „SPARQL Pretty Printer 4“ z [52])

Na základě zjištěného druhu dotazu je vytvářena instance relevantní třídy, která představuje obsah dotazu – tedy pro typ dotazu SELECT je touto třídou

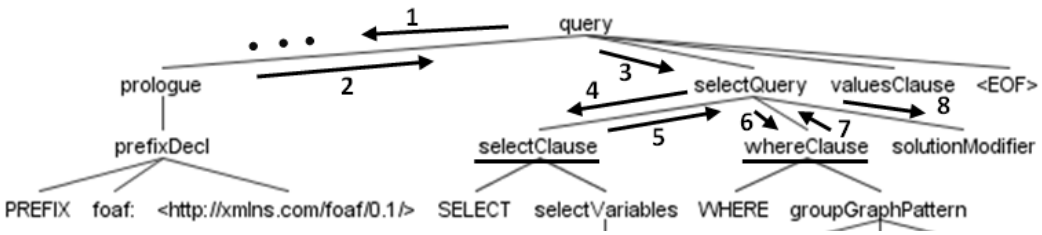
„SelectQueryPane“ z výše uvedeného balíku tříd. Instance je poté vložena do obalující třídy představující záložku v programu Sparkle (třída „SelectQueryTab“, která se nachází ve stejném balíku, jako je zdrojový kód obsahu dotazu).

4.1.3 Průběh vytváření obsahu záložky dotazu

Při inicializaci třídy „SelectQueryPane“ (obsahující jednotlivé klauzule dotazu SELECT) jsou vždy nejprve vyhledávány prefixy načítaného dotazu. Nalezené prefixy jsou následně přidány do již vytvořené prefixové báze dotazu. Tato báze je pouze dočasná, tudíž aplikace Sparkle si získané hodnoty po uzavření daného dotazu pamatovat nebude. Kromě toho je již zmíněná báze „nadřazena“ prefixové bázi celého programu. To je výhodné zejména v případě, kdy je použit jiný prefix pro již existující IRI (případně pro již existující prefix je použita jiná IRI) v bázi programu, jelikož může být v rámci daného dotazu užit načtený prefix a pro ostatní SPARQL dotazy se nic nemění. Na obr. 4.1 se skládá prefix z množiny následovníků klíčového uzlu „prefixDecl“.

Po načtení prefixů jsou vyhledávány klíčové uzly označující klauzule daného dotazu (pro SELECT se jedná např. o klauzuli WHERE, jejíž klíčový uzel je „whereClause“ – viz obr. 4.1). Při nalezení odpovídajícího uzlu v syntaktickém stromu je vyvolán specializující se mechanismus na správné získávání hodnot dané klauzule. Pro uvedenou klauzuli WHERE se jedná o část programu ve třídě „SparqlParserWhereClauseWalker“ v balíku tříd „cz.zcu.mre.sparkle.tools“. Po dokončení činnosti takového mechanismu se již zmíněný algoritmus v rámci třídy „SelectQueryPane“ pouští do vyhledávání dalších klauzulí ve zbytku syntaktického stromu. Do právě načtené klauzule algoritmus již nevstupuje.

Pro názornost je na obr. 4.2 (obrázek se vztahuje k dotazu SELECT z kódu 1.6) naznačen očíslovanými šipkami průběh vyhledání klíčových uzlů, kde čísla označují posloupnost hledání. Šipky s čísly jedna a dva naznačují načítání prefixu a od čísla tři je již nastíněn algoritmus vyhledávání klauzulí dotazu SELECT. Podtržení u některých uzlů na obrázku označuje klauzule, které jsou načítány v již zmíněných specializovaných mechanismech jednotlivých klauzulí. Tyto mechanismy jsou užívány především kvůli znovupoužitelnosti kódu a návaznosti na architekturu programu Sparkle.

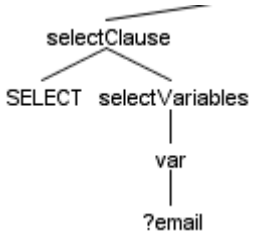


Obrázek 4.2: Ukázka vyhledávání klauzulí v dotazu z kódu 1.6 (propojené názvy klauzulí jsou vytvořeny pomocí nástroje „SPARQL Pretty Printer 4“ z [52])

4.1.4 Procházení klauzule

Klauzulí dotazů v rámci programu Sparkle existuje mnoho, přičemž každá z nich musí být zahrnuta do mechanismu načítání SPARQL dotazů z textových souborů. Procházet zde všechny by vydalo na mnoho stran textu. Navíc součástí zdrojového kódu programu Sparkle jsou i komentáře, které by měly vysvětlovat důležité podrobnosti. Tudíž je dále uveden příklad načítání pouze jedné klauzule.

Pro úplnost je uvedeno načítání klauzule stanovující výsledky dotazů typu SELECT, přičemž je i nadále užíván příklad z kódu 1.6. V syntaktickém stromu příkladu se jedná o část, jejíž klíčový uzel je „selectClause“. Klauzule je zobrazena na obr. 4.3. V rámci dalšího úkolu této DP byla zmiňovaná klauzule v programu Sparkle upravena, tudíž se následující popis bude stahovat k její aktuální podobě. Nová podoba klauzule se popsána v sekci 4.2.1.



Obrázek 4.3: Ukázka klauzule pro výběr výsledků dotazu SELECT z kódu 1.6 (obrázek je vytvořen nástrojem „SPARQL Pretty Printer 4“ z [52])

Samotné načítání klauzule je spuštěno v případě, kdy algoritmus vyhledávající klauzule dotazu (pro SELECT se nachází v již několikrát zmíněné třídě „SelectQueryPane“) nalezne uzel „selectClause“. Poté je zavoláno načítání nacházející se ve třídě „SelectClauseBasePane“, kde mechanismus nejprve zjišťuje

přítomnost speciálního znaku „*“ a modifikátoru DISTINCT (popř. REDUCED). Znak „*“ značí (podle [3]), že mají být do výsledků zahrnuty záznamy všech nalezených proměnných dotazu. Modifikátor DISTINCT garantuje unikátnost záznamů a REDUCED také odstraňuje duplikované záznamy, ale některé ponechává. Pokud je některý z modifikátorů či znak „*“ nalezen, pak je příslušný prvek dotazu v aplikaci Sparkle patřičným způsobem upraven. Pokud není nalezen znak „*“, tak je přikročeno k načítání agregačních funkcí, jejich modifikátorů a proměnných.

Pokud je nalezena proměnná (at' už s agregační funkcí či bez ní), tak je dynamicky vytvořena instance třídy „SelectVarPane“, která je zařazena do kontejneru tvořeného instancí již zmíněné třídy „SelectClauseBasePane“. Po vytvoření objektu třídy „SelectVarPane“ je spuštěn algoritmus, který načítá agregační funkce, jejich proměnné a aliasy. Alias je zástupná proměnná – např. pro složitější výraz s agregačními funkcemi apod.

Obecně jsou metody klauzulí pro načítání dotazů vytvořeny na podobném principu, jako je výše uvedený. Ovšem u některých načítaných částí je nutné hledět na aktuální kontext (ve kterém se daná načítaná hodnota nachází), na možnosti programu Sparkle a mimo jiné např. na datové typy načítaných hodnot. Ty musejí být rozpoznávány, aby byly načtené části dotazů správně v programu Sparkle otypovány a nezobrazovaly se jako chybné. Po načtení celého SPARQL dotazu jsou již užity stávající mechanismy programu Sparkle pro zajištění relevantního zobrazení a umožnění další editace.

Mechanismus načítání je v případě potřeby dále rozšiřitelný o nové typy dotazů či klauzulí. Popisované načítání bylo implementováno nejen pro již existující typy dotazů a klauzulí, ale také pro nové příkazy, které jsou implementované v rámci úkolu přidání vybraných částí podpory SPARQL 1.1, jejichž popis implementace se nachází v následující sekci.

4.2 Podpora vybraných částí SPARQL 1.1

Před zpracováváním tohoto tématického úkolu bylo rozšíření Sparkle o podporu SPARQL 1.1 předem prodiskutováno s vedoucím této DP, přičemž bylo určeno, na které novinky se zaměřit. Krátký popis vhodných částí SPARQL 1.1 se nachází v sekci 3.2 a uvedení do této problematiky je v části textu 1.3.2.

4.2.1 Agregační funkce

Jednou z hlavních přidaných funkcionalit jsou agregační funkce. Jejich stručný popis se nachází v již uvedené sekci 1.3.2. Implementací těchto agregačních funkcí je provedeno mnoho změn především v balíku tříd „cz.zcu.mre.sparkle.gui.query“. Mezi hlavní změny patří úprava stávající architektury klauzule dotazu SELECT pro výběr proměnných, které mají být zahrnuty do výsledků těchto dotazů. Původní klauzule umožňovala především manipulaci s právě zmíněnými proměnnými. Ovšem, aby bylo možné zahrnout neomezený počet agregací, vytvářet aliasy proměnných atd., bylo nutné vytvořit novou architekturu klauzule.

Tato nová architektura se skládá z kontejneru, který představuje instance třídy „SelectClauseBasePane“ (nacházející se ve výše zmíněném balíku tříd), jehož hlavním úkolem je uchovávání instancí tříd „SelectVarPane“ a „SelectClausePane“. Dále zajišťuje komunikaci jednotlivých částí klauzule s ostatními částmi aplikace Sparkle.

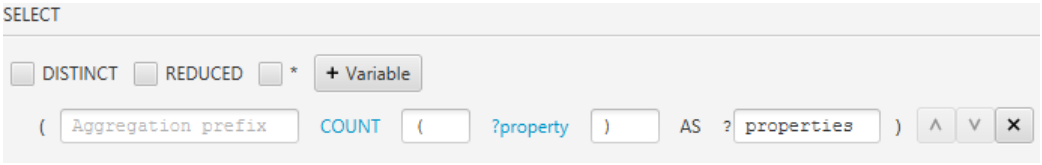
Instance třídy „SelectClausePane“ se v rámci dotazu SELECT užívá pouze jednou. To z důvodu, že obsahuje funkcionality pro přidání instancí „SelectVarPane“ a mimo jiné komponenty modifikátorů DISTINCT a REDUCED. Vytvoření instance této třídy je vždy automaticky vyvoláno při inicializaci objektu již zmíněné třídy „SelectClauseBasePane“.

Pro reprezentaci proměnných (a s nimi spojených agregačních funkcí apod.) se v klauzuli užívají instance třídy „SelectVarPane“. Tato třída umožňuje výběr proměnných, k těmto zvoleným proměnným přidávat různé agregace a další potřebné modifikátory. Zvolením proměnné dochází k přeskupení prvků daného objektu třídy „SelectVarPane“, kdy je odstraněna výběrová nabídka pro proměnné a namísto ní je vložena instance třídy „VariableLabel“, která značí zvolenou proměnnou a umožňuje vlastní odstranění z instance třídy „SelectVarPane“. Tudíž je možné znovu volit proměnnou v dané části klauzule. Pro zvolení agregačních funkcí jsou v klauzuli vytvořeny prvky fungující na stejném principu jako pro výběr proměnných. Jen jsou po zvolení agregace vloženy do instance třídy „SelectVarPane“ prvky umožňující modifikaci vybrané agregace či zmíněné proměnné. Další funkcionalitou je vytváření aliasů (zástupných proměnných). Užitím grafického prvku s nápisem „AS“ je přidáno do instance třídy „SelectVarPane“ textové pole umožňující používání aliasů.

V celé klauzuli je vytvořena správa užívaných proměnných, která zabezpečuje např. odstranění instancí tříd „VariableLabel“ (objekty třídy znázorňují zvolené proměnné) v případě neexistujícího dalšího výskytu daných proměnných mimo klauzuli v rámci zpracovávaného dotazu. Kromě správy proměnných je např. implementován mechanismus manipulace s instancemi „SelectVarPane“ v kontejneru

objektu vytvořeného na základě třídy „SelectClauseBasePane“. Tento mechanismus umožňuje např. odstraňování spravovaných instancí tříd.

Pro názornost je na obr. 4.4 vyobrazeno použití popisované klauzule v programu Sparkle na SPARQL dotaz z kódu 1.18.



Obrázek 4.4: Náhled užití klauzule (určující výsledky dotazů typu SELECT) pro příklad z kódu 1.18

4.2.2 Vnořené dotazy

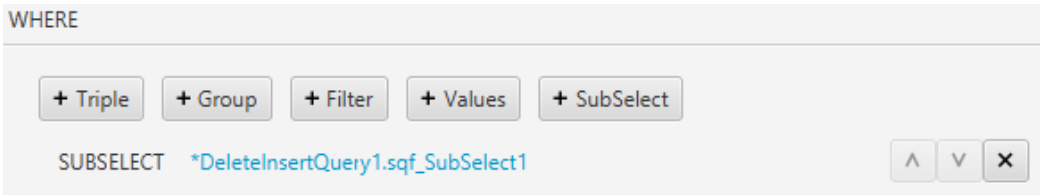
Jednou z nejvýznamnějších novinek SPARQL 1.1 jsou vnořené dotazy, jejichž popis se nachází v sekci 1.3.2. Pro implementaci tohoto rozšíření je vytvořeno několik nových tříd v balíku „cz.zcu.mre.sparkle.gui.query“ a pozměněna WHERE klauzule takovým způsobem, aby bylo možné vnořené dotazy vytvářet.

Implementace je provedena tak, že pokud chce uživatel přidat nový vnořený dotaz, pak je do WHERE klauzule dotazu vložen na relevantní místo odkaz na definici dotazu a samotná definice vnořeného dotazu je vytvořena v nové záložce programu Sparkle. Relevantním místem ve WHERE klauzuli je např. konkrétní skupina, ve které uživatel pracuje.

Reference vnořeného dotazu

Vkládaný odkaz do WHERE klauzule tvoří vazbu mezi definicí vnořeného dotazu (jeho záložkou v aplikaci Sparkle) a příslušným SPARQL dotazem, který obsahuje vnořený dotaz. Reference je instancí třídy „SubSelectPane“, která se nachází ve výše zmíněném balíku tříd. Tento odkaz iniciuje vytváření záložky s definicí vnořeného dotazu a relevantním hierarchickým názvem. Dále obstarává vracení aktuální podoby své části programu v textové formě (např. pro složení dotazu k vyhodnocování či uložení), nastavuje indikaci provedených změn v návaznosti na příslušný dotaz a v neposlední řadě vytváří referenci pro zobrazení relevantní záložky vnořeného dotazu.

Náhled reference vnořeného dotazu pro příklad z kódu 1.16 je na obr. 4.5. U tohoto odkazu jsou rovněž znázorněna indikace neuložených změn ve vnořeném dotazu, což se projevuje znakem „*“ u odkazu.



Obrázek 4.5: Ukázka reference vnořeného dotazu pro příklad z kódu 1.16

Definice vnořeného dotazu

Definice vnořeného dotazu je vždy vytvořena ve vlastní záložce. Záložka je instancí třídy „SubSelectQueryPane“, která se vyskytuje ve stejném balíku tříd jako již uvedený odkaz vnořeného dotazu. Obsahem této definice jsou následující klauzule:

- *SELECT* – udává proměnné, jejichž záznamy mají být součástí výsledku vnořeného dotazu, popis implementace je v části 4.2.1,
- *WHERE* – slouží k určení trojic, kterých se dotaz týká,
- *GROUP BY* – popis klauzule se nachází v sekci 1.3.2 a implementace v části 4.2.4,
- *HAVING* – klauzule je popsána ve stejných sekcích jako GROUP BY,
- *ORDER BY* – určuje řazení výsledků podle parametru – viz sekce 1.3.1,
- *LIMIT + OFFSET* - popis se již nachází v části 1.3.1,
- *VALUES* – popis klauzule v části 1.3.2 a implementace v sekci 4.2.6.

Ukázka obsahu záložky vnořeného dotazu se nachází v příloze C. Tato záložka je uzpůsobena pro generování své části dotazu z kódu 1.16.

Systém vnořování a pojmenovávání dotazů

Ke SPARQL operacím lze připojovat i více vnořených dotazů. Např. příkaz DELETE/INSERT z kódu 1.16 může mít ve své WHERE klauzuli libovolný počet

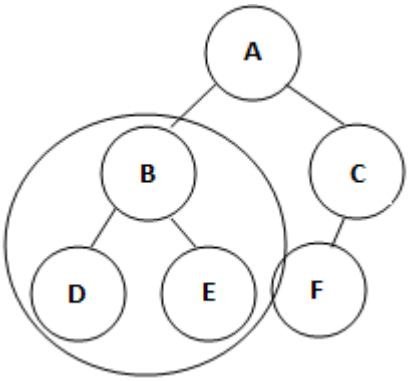
vnořených dotazů. Pojmenování těchto vnořených dotazů je vždy tvořeno z názvu SPARQL dotazu (příkazu), koncovky „_SubSelect“ a pořadového čísla v rámci dané WHERE klauzule. Názorná ukázka pojmenování vnořeného dotazu je uvedena výše na obr. 4.5 (aplikováno na příklad z kódu 1.16).

Dále je možné do vnořených dotazů vnořovat další dotazy. Při vnořování je jejich pojmenovávání již řetězeno, což znamená, že v případě přidání dalšího vnořeného dotazu do již existujícího vnořeného dotazu je převzat pro nový dotaz celý název původního vnořeného dotazu. K tomu je opět přidána již zmíněná koncovka „_SubSelect“ a pořadové číslo v rámci WHERE klauzule, ve které je přidáváno. Pokud by byl vložen další vnořený dotaz do vnořeného dotazu v příkladu na obr. 4.5, pak by byl vytvořený název „DeleteInsertQuery1.sqf_SubSelect1_SubSelect1“.

Vyhodnocování a odstraňování vnořených dotazů

Vyhodnocování SPARQL dotazů (obsahující vnořené dotazy) je možné provádět stejným způsobem (viz [1]) jako doposud. Tedy při zobrazeném hodnoceném dotazu stačí kliknout na tlačítko „Evaluate“. Provázání vnořených dotazů je vytvořeno tak, že je možné vyhodnocovat pouze určité části dotazů. V případě, že uživatel bude mít zobrazen některý z vnořených dotazů dané SPARQL operace a spustí vyhodnocování, tak je hodnocen dotaz skládající se pouze z daného vnořeného dotazu a popř. dalších jeho vnořených dotazů. Na stejném principu funguje i odstraňování vnořených dotazů.

Příkladem může být dotaz A, obsahující vnořené dotazy B a C, které mají další vnořené dotazy – viz schematický obr. 4.6. Pokud uživatel spustí evaluaci při zobrazeném dotazu B, pak je vyhodnocena část dotazu ve vyznačeném kruhu. To samé platí i pro odstraňování.



Obrázek 4.6: Schéma vyhodnocování (odstraňování) části dotazu

Ukládání

Při ukládání vnořených dotazů je využíváno existujícího systému, pouze jsou do WHERE klauzulí ukládány příslušné vnořené dotazy (do XML značek „SubSelect“). Příklad uložení části dotazu z kódu 1.16 se nachází v příloze D.

4.2.3 Přidané typy příkazů

Další částí implementace tématického celku podpory vybraných částí SPARQL 1.1 je přidání příkazů DELETE DATA, INSERT DATA a DELETE/INSERT. Nejdůležitější provedené změny (tykající se tohoto úkolu) se nacházejí v balíku tříd „cz.zcu.mre.sparkle.gui.query“ a v balíku „cz.zcu.mre.sparkle.gui.evaluation“. V prvním jmenovaném balíku jsou vytvořeny definice dotazů (popř. i nových klauzulí) a ve druhém podpora pro vyhodnocování.

Pro správné vyhodnocování těchto nových příkazů je pozměněna třída souboru „DataAgent.java“ v balíku „cz.zcu.mre.sparkle.data“. Instance této třídy vytváří rozhraní mezi úložišti a ostatními částmi programu Sparkle. Přičemž je v tomto rozhraní hlavně přidána podpora pro zpracování nově implementovaných příkazů na základě popisu knihovny Jena (viz [56]). Jena je v aplikaci Sparkle používána pro práci s RDF.

Příkaz DELETE DATA

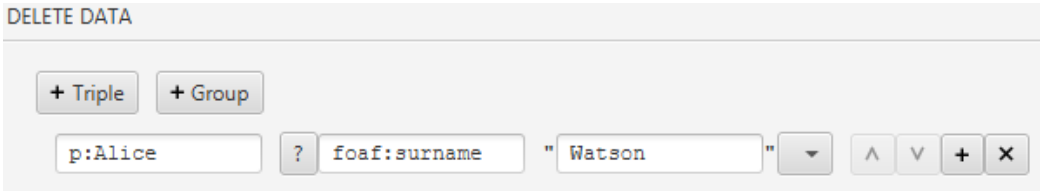
Popis tohoto typu příkazu se nachází v sekci 1.3.2. Samotná implementace se skládá z několika tříd, které zajišťují vhodné prostředí pro vytváření a vyhodnocování příkazu DELETE DATA.

Struktura dotazu je vytvářena instancí třídy „DeleteQueryPane“ (z výše uvedeného balíku tříd), která obsahuje následující části:

- *DELETE DATA* – povinná část dotazu udávající jeho typ,
- *QuadData* – klauzule, která umožňuje vkládání pouze trojic a případné určení pojmenovaného grafu daných trojic klíčovým slovem GRAPH a příslušného prefixu či IRI. Přičemž ve zmíněných trojicích nemohou být použity např. proměnné či tzv. „blank nodes“ (uzly grafů, které nejsou literály a nemají ani URI). Tento typ klauzule je vytvořen modifikováním existující třídy pro WHERE klauzuli (třídy „GroupGraphPatternPane“), která se nachází ve stejném balíku jako definice popisovaného dotazu.

Kromě vytváření SPARQL příkazu je také implementováno skládání aktuální podoby v textové formě (např. pro vyhodnocování), ukládání příkazu do SQF souboru a vše ostatní pro zajištění kompatibility s programem Sparkle.

Pro názornost je uveden příkaz DELETE DATA na obr. 4.7. Příklad je uzpůsoben kódu 1.14.



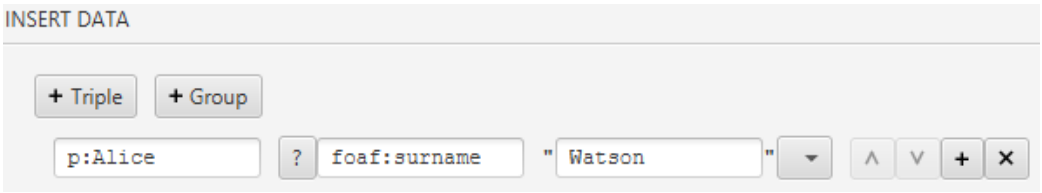
Obrázek 4.7: Náhled příkazu DELETE DATA

Příkaz INSERT DATA

Tento typ příkazu je popsán v sekci 1.3.2 a implementován obdobným způsobem jako příkaz DELETE DATA. Definice obsahu záložky příkazu je především ve třídě „InsertQueryPane“ (nacházející se ve výše uvedeném balíku tříd), přičemž struktura je následující:

- *INSERT DATA* – povinná část, která udává typ dotazu,
- *QuadData* – stejná klauzule jako u příkazu DELETE DATA (viz sekce 4.2.3), jen jsou povolené „blank nodes“.

Na obr. 4.8 je znázorněn příkaz INSERT DATA z kódu 1.13 v aplikace Sparkle.



Obrázek 4.8: Ukázka INSERT DATA příkazu v programu Sparkle na příklad z kódu 1.13

Příkaz DELETE/INSERT

Popis tohoto příkazu ze SPARQL 1.1 se nachází v sekci 1.3.2, kde je uveden i příslušný příklad znázorňující syntaxi a další podrobnosti. Při implementaci této části je využito nově vytvořených klauzulí z příkazů DELETE a INSERT DATA (viz sekce 4.2.3). Obsah záložky příkazu je definován ve třídě „DeleteInsertQueryPane“, která se vyskytuje ve stejném balíku jako např. DELETE DATA. V následující části bude popsána pouze implementace nových klauzulí, jejichž třídy se nacházejí ve stejném balíku tříd, jak popisovaný příkaz.

- *WITH* – jedná se o nově vytvořenou klauzuli v rámci programu Sparkle, jejíž implementace je provedena následujícím způsobem. Vytvořením instance třídy „WithClausePane“ vzniká kontejner, jehož hlavním účelem je uchovávání objektu třídy „SingleWithClausePane“, který je určený k získání prefixu či IRI pro identifikaci požadovaného grafu v rámci klauzule. Dále je vytvořeno ukládání a načítání hodnot klauzule ze souboru, implementováno generování dané části dotazu pro vyhodnocování a další potřebné funkcionality pro její efektivní využívání v programu Sparkle,
- *DELETE* – klauzule využívající implementaci příkazu DELETE DATA – viz sekce 4.2.3,
- *INSERT* – implementováno pomocí částí příkazu INSERT DATA – viz sekce 4.2.3,
- *USING* – implementace této nově přidané klauzule do programu Sparkle je provedena podobným způsobem jako výše uvedená příkazová část WITH. Tedy vytvořením objektu třídy „UsingClausePane“ vzniká kontejner pro uchovávání instancí třídy „SingleUsingClausePane“, které slouží pro získávání hodnot pro identifikaci RDF,
- *WHERE* – stejná klauzule, jako je i u jiných SPARQL dotazů. Popis viz např. sekce 1.3.2.

Příklad DELETE/INSERT je znázorněn v příloze E. Tento příklad zobrazuje vytváření příkazu z kódu 1.15 v programu Sparkle.

V rámci implementace operace DELETE/INSERT je vytvořeno mnoho dalších funkcionalit, např. pro správné generování dotazů při užití pouze některých výše uvedených klauzulí atp. Příkladem může být automatické generování varianty této operace, kterou je DELETE WHERE příkaz. Ten vzniká (podle [27]) když jsou vyplněny pouze klauzule DELETE a WHERE.

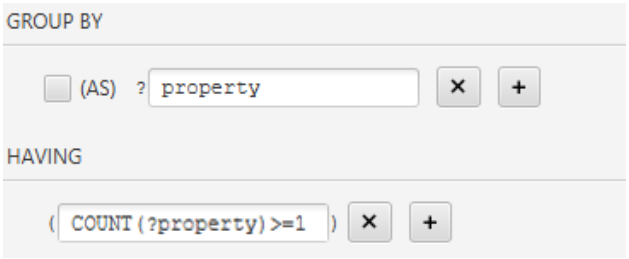
4.2.4 Klauzule GROUP BY a HAVING

K relevantním typům dotazů v programu Sparkle byly přidány klauzule GROUP BY a HAVING. Jejich popis se nachází v sekci 1.3.2. Důležité části těchto klauzulí jsou implementovány v balíku tříd „cz.zcu.mre.sparkle.gui.query“ společně se stávajícími klauzulemi a dalšími komponentami dotazů. Obě tyto klauzule jsou automaticky inicializovány při vytváření SPARQL dotazů, kterých jsou součástí.

Při inicializaci GROUP BY klauzule je vytvářena instance třídy (z výše uvedeného balíku) „GroupByClausePane“, která představuje kontejner pro objekty třídy „GroupByClauseRulePane“. Kromě toho slouží ke komunikaci klauzule s okolím, tudíž zajišťuje správu proměnných, skládá svou část dotazu pro další zpracování atd. Instance třídy „GroupByClauseRulePane“ představují konkrétní podmínky (omezení) vytvářené popisovanou klauzulí.

U HAVING je architektura navržena podobně jako u výše uvedené GROUP BY klauzule. Kontejner je instancí třídy „HavingClausePane“ a omezení se vytvářejí objekty třídy „HavingClauseRulePane“.

Názorná ukázka klauzulí v programu Sparkle pro příklad z kódu 1.19 se nachází na obr. 4.9.



Obrázek 4.9: Klauzule GROUP BY a HAVING pro příklad z kódu 1.19

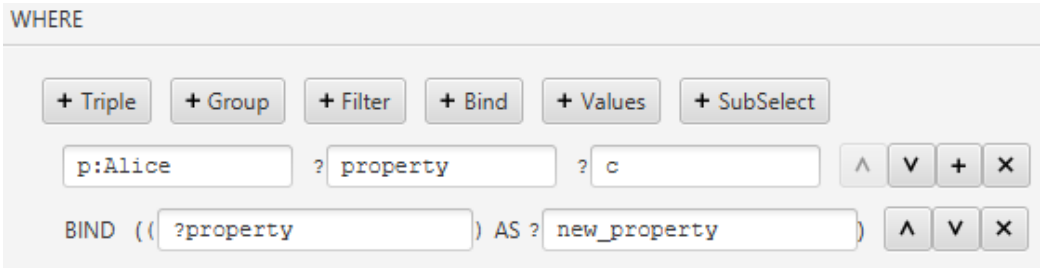
4.2.5 BIND

Popis této novinky SPARQL 1.1 je umístěn v sekci 1.3.2, kde se nachází i příklad užití. Kvůli implementaci této klauzule je do balíku „cz.zcu.mre.sparkle.gui.query“ přidána třída „BindPane“. Hlavním úkolem této třídy je vytvoření prostředí pro možnost přidávání BIND do WHERE klauzulí SPARQL dotazů.

V popisu klauzule BIND je rovněž uvedeno, že ji lze užívat např. v GROUP BY klauzuli či v části dotazu SELECT pro výběr proměnných, které mají být

zohledněny ve výsledcích daného dotazu. Tyto případy užití klauzule BIND mají v programu Sparkle implementována zde uváděná přiřazování vlastním způsobem a třídu „BindPane“ nevyužívají. To z důvodu, že se jedná o oddělené komponenty, při jejichž vyhodnocování (a dalších operacích) je nutný jiný postup v rámci programu Sparkle.

Ve spodní části obr. 4.10 se nachází příklad tvorby BIND ve WHERE klauzuli. Tato část je definována v již několikrát zmiňované třídě „BindPane“, přičemž příklad na obrázku vychází z kódu 1.23.



Obrázek 4.10: Náhled tvorby BIND ve WHERE části příkladu z kódu 1.23 v programu Sparkle

Součástí implementace je vytvoření načítání a ukládání do souboru, zapojení prvku do správy proměnných v rámci dotazu atd. Kromě tohoto jsou dále implementovány další funkcionality pro správné fungování v programu Sparkle s respektováním gramatiky SPARQL.

4.2.6 Klauzule VALUES

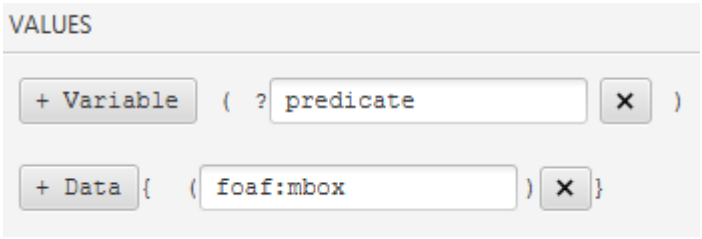
Posledním klíčovým prvkem implementace podpory SPARQL 1.1 do programu Sparkle je přidání klauzule VALUES, jejíž popis se nachází v sekci 1.3.2. Samotná implementace této části je provedena především v následujících třídách balíku „cz.zcu.mre.sparkle.gui.query“:

- *ValuesClausePane* – vytváří především kontejner pro uchovávání instancí třídy „SingleValuesVariableClausePane“ a „SingleValuesDataClausePane“. Popis těchto tříd se nachází níže v tomto textu. Při vytváření uchovávaných instancí kontejner dodržuje určitá pravidla, aby byl zachován soulad s gramatikou SPARQL. Kromě toho spravuje proměnné a předávání hodnot mezi klauzulí a zbytkem programu Sparkle,
- *SingleValuesVariableClausePane* – slouží k uchovávání proměnných, k nimž jsou v této klauzuli přiřazeny hodnoty pro filtrování výsledků SPARQL do-

tazů. Na základě pravidel z třídy „ValuesClausePane“ počet instancí této třídy ovlivňuje počet instancí „SingleValuesDataClausePane“ takovým způsobem, aby byla zachována správná syntaxe,

- *SingleValuesDataClausePane* – třída umožňující zadávání konkrétních hodnot, které slouží pro filtrování výsledků SPARQL dotazů,
- *ValuesPane* – upravuje výše uvedenou třídu „ValuesClausePane“ (tedy celou klauzuli VALUES) tak, aby byla použitelná v klauzuli WHERE. Ve třídě je především pozměněn vzhled, přidána tlačítka pro manipulaci ve WHERE a upraveno předávání hodnot mezi klauzulí a zbytkem programu.

Pro úplnost je níže na obr. 4.11 zobrazena klauzule VALUES, přičemž uvedené hodnoty se vztahují k příkladu z kódu 1.22.



Obrázek 4.11: Náhled klauzule VALUES v programu Sparkle

4.3 Vyhledávání vlastností subjektů

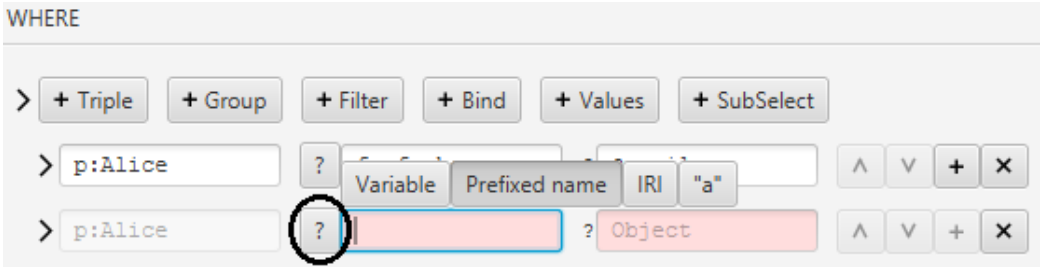
Posledním tématickým celkem praktické části DP je vyhledávání dostupných vlastností subjektů, které jsou (před použitím této nové funkcionality) rozpracovaným dotazem identifikovatelné. Popis funkcionality se nachází v sekci 3.4. V následujícím textu jsou zmíněny nejdůležitější části implementace. Podrobnosti jsou uvedeny v komentářích zdrojových kódů, jejichž stěžejní část se nachází v metodě „findPropertiesButtonOnAction“ třídy „TriplePane“, která je obsažena v balíku tříd „cz.zcu.mre.sparkle.gui.query“.

4.3.1 Podmínky spuštění mechanismu prohledávání

Spuštění a získání vlastností daného subjektu je možné provést tlačítkem s nápisem „?“ (viz obr. 4.12) a za předpokladu, že WHERE klauzule, ve které se daný

subjekt nachází, je vyhodnotitelná (nejsou v ní chyby). To znamená, že pokud jsou vyhledávány vlastnosti subjektu, který je zadán proměnnou, pak musí mít WHERE klauzule (kromě trojice, ve které je vyhledávání spuštěno, v té stačí mít uvedený relevantně pouze subjekt) správnou syntaxi. Správné syntaxi napomáhá např. podbarvování textových polí s chybnými hodnotami v programu Sparkle. Pokud je subjekt, jehož vlastnosti mají být vyhledávány, zadán explicitně např. pomocí prefixu (viz subjekt „p:Alice“ na obr. 4.12), pak na zbytek vytvářeného dotazu není brán zřetel a vyhledávání je hned spuštěno.

Samotné spouštěcí tlačítko je použitelné pouze při relevantně vyplněném subjektu dané trojice a za předpokladu, že je nastaven typ pole hledané vlastnosti na „Prefixed name“ – viz obr. 4.12.



Obrázek 4.12: Tlačítko spuštění vyhledávání vlastností subjektu (označeno v kruhu)

4.3.2 Získávání hloubky prohledávání

Po kliknutí na tlačítko z obr. 4.12 je zobrazeno okno, které slouží pro získání hloubky vyhledávání vlastností (RDF v úložištích lze reprezentovat jako grafy, kde hloubky vyhledávání označují maximální délku cest – nejvyšší možné počty procházených vlastností). Definice obsahu okna pro získávání hloubky je implementována ve třídě „InputDialog“, která se nachází v následujícím balíku tříd „cz.zcu.mre.sparkle.gui.forms“. Jedná se o znovu použitelnou komponentu, jejíž hlavním úkolem je získání hodnot od uživatele. Toto okno je vyvoláváno algoritmem z již zmíněné třídy „TriplePane“. Algoritmus zajišťuje získání celého nezáporného čísla, vyjadřujícího hloubku prohledávanou.

Správně vložená hodnota je vždy ukládána a při dalším spuštění tohoto rozšíření přednastavena. Pokud je program korektně ukončen, pak je poslední uložená hodnota při dalším spuštění aplikace Sparkle a užití této funkcionality rovněž přednastavena v okně pro získání hloubky vyhledávání.

4.3.3 Způsoby identifikace subjektu

Po správném zadání hloubky prohledávání je převzata WHERE klauzule algoritmem, který je umístěný v již zmíněné třídě „TriplePane“. Tento algoritmus na základě hodnoty subjektu, jehož vlastnosti mají být vyhledávány, určí postup zpracování. Pokud je subjekt explicitně zadán (např. pomocí IRI), tak je pouze vytvořena reference na tento uzel. V případě, že je subjekt zadán proměnnou, pak je WHERE klauzule převzata do nově vygenerovaného dotazu, který slouží k nalezení uzlu hledaného subjektu. Pokud není hledaný subjekt dostatečně popsán ve WHERE klauzuli, pak výsledky nově generovaného dotazu mohou čítat více záznamů představujících odpovídající uzly, což má za následek, že budou ve výpisu vlastnosti od všech těchto uzlů.

Již zmíněný vygenerovaný dotaz je vždy typu SELECT, jehož klauzule (určující požadované výsledky) je vygenerována na základě hodnoty hledaného subjektu dané trojice, ze které bylo vyhledávání spuštěno. Pro názornost by proměnná „?subject“ ve WHERE klauzuli příkladu z kódu 1.15 byla vyhledávána vygenerovaným dotazem, který je znázorněn v kódu 4.1. Ještě před vytvořením dotazu, byla do WHERE přidána trojice, kde subjekt tvořila proměnná „?subject“ a ostatní hodnoty trojice byly prázdné. Názvy proměnných „?uOCgjCdADXNp“ a „?dGiVNbiabpmG“ v příkladu kódu 4.1 jsou vygenerované náhodné řetězce písmen pro správné vyhledávání vlastností.

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

SELECT ?subject
WHERE {
  ?subject  foaf:mbox  "alice@email.com" .
  ?subject ?uOCgjCdADXNp ?dGiVNbiabpmG .
}
```

Kód 4.1: Ukázka vygenerovaného dotazu

Po získání relevantních uzlů je přistoupeno k vyhledávání vlastností. Vyhledávání je možné kdykoliv ukončit (návod na ukončení – viz příloha A). Možnost ukončení je vytvořena především kvůli případu příliš dlouhého vyhledávání, které může být zapříčiněno např. uživatelem, který zadá příliš velkou hloubku prohledávání (za předpokladu rozsáhlého úložiště).

4.3.4 Algoritmus vyhledávání vlastností

Vyhledávání vlastností subjektů je vytvořeno na základě algoritmu prohledávání do šířky (BFS), který funguje (jak již bylo naznačeno v sekci 4.1.1) na principu navštívení všech sousedů daného uzlu a poté pokračuje navštěvováním sousedů již navštívených sousedních uzlů. Podrobnější popis BFS je např. v [55].

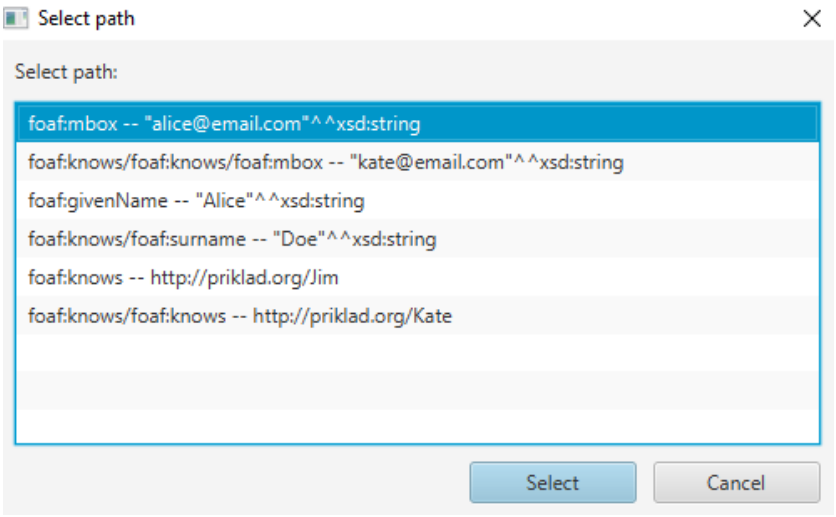
Samotná implementace vyhledávání se nachází v již zmíněné třídě „TriplePane“ – v metodě „getPaths“. Algoritmus vyhledávání postupně prochází graf, přičemž se pokouší převádět získané vlastnosti (kvůli přehlednosti ve výpisu vyhledávání) do prefixového tvaru. Pokud není nějaký prefix znám, pak je příslušná vlastnost vložena do výpisu či cesty ve standardní formě. Nalezené cesty mechanismus ukládá (společně s koncovými hodnotami a jejich datovými typy) do kolekce textových řetězců, které jsou výstupem algoritmu.

Kvůli odstranění možnosti zacyklení jsou již nalezené uzly mechanismem vyhledávání uchovávány. To z důvodu, že daný uzel může disponovat více vlastnostmi, takže existuje více způsobů, jak se k němu při prohledávání dostat.

4.3.5 Výpis a výběr nalezených vlastností

Po vytvoření a naplnění již zmíněné kolekce nalezenými cestami a jejich koncovými hodnotami je zavoláno okno, jehož definice je implementována ve třídě „PropertiesDialog“ balíku „cz.zcu.mre.sparkle.gui.forms“. Toto okno má implementované metody pro správné zobrazení a výběr nalezených cest.

Na obr. 4.13 je znázorněno okno třídy „PropertiesDialog“. Okno obsahuje všechny získané cesty (a jejich koncové hodnoty) proměnné „?subject“ z WHERE klauzule příkazu v kódu 1.15 po aplikaci vyhledávání vlastností na RDF z kódu 1.5.



Obrázek 4.13: Náhled okna se získanými vlastnostmi pro proměnnou „?subject“ příkazu 1.15 při aplikaci na RDF z kódu 1.5

Po zvolení je daná cesta převzata mechanismem třídy „TriplePane“, jehož hlavním úkolem je její zpracování. Zmíněné zpracování spočívá v určení jednotlivých částí cesty (zvolený záznam se skládá z cesty a hodnoty konce cesty). Tyto části jsou poté doplněny do dané trojice, ze které bylo vyhledávání spuštěno.

5 Ověření a testování aplikace

Jednotlivá nově přidaná rozšíření (zmíněná v předchozí kapitole) jsou otestována způsoby, jejichž popisy se nacházejí níže. Pro uvedená testování nejsou zjištěny žádné problémy ve funkčnosti nových rozšíření. V rámci programu Sparkle nejsou získány žádné poznatky o potenciálně vzniklých potížích, které by byly způsobeny nově přidanými funkcionalitami.

5.1 Načítání SPARQL dotazů ze souborů

Načítání SPARQL dotazů z textových souborů je otestováno manuálním způsobem popsaným dále v textu. Samotné testování probíhalo během implementace jednotlivých částí tohoto tématického celku na níže zmíněných SPARQL dotazech, které jsou součástí přiloženého přenosného média, a to včetně načtených dotazů testovaným mechanismem (uložených programem Sparkle do SQF souborů).

Při testování byl SPARQL dotaz nejdříve načten z textového souboru do programu Sparkle. Po načtení bylo spuštěno jeho vyhodnocení, čímž se automaticky provádí mimo jiné i kontrola syntaxe. Pokud byl dotaz úspěšně vyhodnocen, tak byl program Sparkle přepnut do textového režimu. V tomto textovém módu byl vyhodnocený dotaz vizuálně porovnán se SPARQL dotazem z textového souboru.

Pro samotné testování byla získána sada SPARQL dotazů, od vedoucího DP. Tato sestava čítala celkem 45 SPARQL dotazů, přičemž všechny byly typu SELECT. Tři dotazy obsahovaly v klauzulích (určujících proměnné, jejichž záznamy jsou součástí výsledků vyhodnocených dotazů) agregační funkce, 29 dotazů disponovalo skupinami ve WHERE klauzulích (např. OPTIONAL), 37 mělo ORDER BY, tři LIMIT a stejný počet obsahoval i GROUP BY klauzuli.

Jelikož v rámci úkolu vytvoření podpory vybraných částí SPARQL 1.1 bylo nutné přidat i typy příkazů, které nebyly součástí výše uvedené sady SPARQL dotazů, tudíž byla tato tématická část testována na vybraných příkladech z [3, 27]. Při tomto testování byly vždy vzaty příkazy z dané části textu, která se vztahovala k právě zpracovávané funkcionalitě. Tzn., že při přidávání podpory např. příkazu DELETE DATA byly vyzkoušeny vybrané příkazy části 3.1.2 z [27]. Převzaté testovací SPARQL dotazy z [3, 27] jsou součástí přiloženého média.

5.2 Podpora vybraných částí SPARQL 1.1

Testování tohoto tématického celku probíhalo rovněž manuálně během přidávání jednotlivých částí. Byla kontrolována správná funkčnost přidaných či upravovaných prvků programu Sparkle, jestli provedená implementace (s ohledem na možnosti Sparkle popsanými např. v [1]) odpovídá syntaxi SPARQL (viz [3]) a zda nevznikají nějaké problémy se zpracovávanými částmi v rámci celého programu Sparkle.

Dále bylo zkoušeno načítání SPARQL dotazů (popř. příkazů) z textových souborů, které je popsáno v sekci 5.1. Při testování vybraných částí SPARQL 1.1 byly zpravidla načtené dotazy dále uloženy do SQF souborů a poté znovu načteny. Po načtení bylo spuštěno znovu vyhodnocení a provedena vizuální kontrola vyhodnocených SPARQL dotazů. Testovací příklady (včetně SQF souborů) jsou, jak je již zmíněno v sekci 5.1, součástí přiloženého přenosného média.

Při testování nově přidaných příkazů (např. DELETE DATA) bylo také zkoušeno jejich vyhodnocování na konkrétním RDF, které se nacházelo v lokálním úložišti. To z důvodu, že bylo při implementaci nutné přidat podporu pro evaluaci těchto příkazů. Vyhodnocování je otestováno na základě RDF z kódu 1.5 a testovacími příkazy byly příklady z kapitoly 1 (např. kód 1.14 či 1.15). Příkazy v původní podobě i po načtení do programu Sparkle (tedy v SQF souborech) jsou uloženy na přiloženém přenosném médiu.

5.3 Vyhledávání vlastností subjektů trojic

Pro vyzkoušení funkčnosti tohoto rozšíření bylo využito RDF z kódu 1.5. Samotné testování probíhalo rovněž manuálně, přičemž spočívalo v načtení již zmíněného RDF a SPARQL dotazu z kódu 1.6 do programu Sparkle. Poté byl načtený dotaz různě upravován tak, aby bylo možné ověřit, zda se prohledávací algoritmus nezacyklí apod. Již zmíněné RDF a všechny variace SPARQL dotazu (včetně komentářů jejich účelů testování) jsou uloženy na přiloženém médiu.

6 Dosažené výsledky

Do programu Sparkle jsou přidána rozšíření, která zvyšují jeho celkový potenciál a usnadňují práci s vytvářením či upravováním SPARQL dotazů. Díky rozšířením je nyní možné načítat SPARQL dotazy z textových souborů, vyhledávat vlastnosti daných subjektů v úložištích a využívat některých novinek SPARQL 1.1, jejichž hlavní části jsou následující:

- agregační funkce,
- vnořené dotazy,
- příkazy DELETE DATA, INSERT DATA a DELETE/INSERT,
- přidané klauzule BIND, GROUP BY, HAVING, USING, VALUES a WITH.

Načítání SPARQL dotazů z textových souborů zahrnuje i nově přidané části ze SPARQL 1.1.

6.1 Porovnání s programy ostatních autorů

V následující části textu je vždy porovnán program Sparkle s aplikacemi, u nichž byla čerpána inspirace pro tvorbu rozšíření, jejichž popis implementace se nachází v kapitole 4.

Načítání SPARQL dotazů z textových souborů je inspirováno především programem „Twinkle: SPARQL Tools“ (viz sekce 3.1). Načítání u aplikace „Twinkle“ je jednodušší než u Sparkle, jelikož program disponuje pouze textovým editorem, do kterého jsou SPARQL dotazy vkládány ve stejných podobách, v jakých se nacházejí v příslušných textových souborech. Navíc při načítání neprobíhají kontroly, zda jsou vkládané hodnoty skutečně SPARQL dotazy. Naproti tomu v programu Sparkle je nyní implementováno načítání, které provádí lexikální a syntaktické analýzy a případně vypisuje nalezené chyby. Podle načítaných dotazů jsou ve Sparkle automaticky voleny vhodné definice záložek, v nichž jsou dynamicky vytvářeny klauzule pro načítané hodnoty.

Jak již bylo zmíněno v sekci 3.2, rozšíření o podporu vybraných částí SPARQL 1.1 je inspirováno především programem YASGUI. V aplikaci YASGUI je ovšem podpora novinek SPARQL 1.1 mnohem rozsáhlejší, což znamená, že např. podporuje i manipulaci s grafy v úložištích (tedy operace, jako jsou LOAD či CLEAR).

Podle [27] LOAD slouží pro vkládání trojic do daného grafu a již uvedený CLEAR naopak vymazává všechny trojice ze zvoleného grafu). Program Sparkle oproti YASGUI nabízí snazší vytváření SPARQL dotazů. To z důvodu, že po zvolení typu dotazu je již struktura SPARQL operace předem daná a uživatel pouze přidává klauzule a jejich hodnoty (pokud není v textovém režimu, pak se tato výhoda vytrácí).

Inspirace pro vyhledávání vlastností subjektů pochází především z nástroje Gosparqled (viz sekce 3.4). Při pohledu na možnosti Gosparqled z hlediska rozšíření jsou rozdíly oproti programu Sparkle spíše jen ve spouštění mechanismů. Kdy v Gosparqled je mechanismus vyhledávání aktivován klávesovou zkratkou CTRL+SPACE (mezerník), přičemž u Sparkle musí uživatel kliknout na příslušné tlačítko (viz příloha A). Dalším rozdílem je chápání hloubky prohledávání. Pokud je v Gosparqled zadána hodnota hloubky např. dva, pak jsou vypsány vlastnosti pouze z dané hloubky. Naproti tomu u aplikace Sparkle při zadání stejné hodnoty hloubky prohledávání jsou vypsány všechny vlastnosti od těch s nejkratší cestou do zvolené vzdálenosti (tj. pro uvedený příklad jsou vypsány i vlastnosti v hloubce prohledávání jedna).

6.2 Nevýhody realizovaných rozšíření

Tato sekce textu se zaobírá nalezenými nedostatky a nevýhodami realizovaných rozšíření. Dále jsou případně zdůrazňovány důvody těchto obtíží a v některých případech jsou uvedeny i jejich možná řešení.

6.2.1 Načítání SPARQL dotazů z textových souborů

Poměrně velkou nevýhodou realizovaného načítání SPARQL dotazů z textových souborů je, že program nenahlásí případ, kdy je nějaká část dotazu vynechána. Toto vynechání může nastat v případě, kdy je během implementace daná načítaná varianta klauzule (či její část) opomenuta a mechanismus nemá prostředky, jak by s ní mohl naložit. Uživatel tuto skutečnost pozná až při dalším studiu načteného dotazu v programu Sparkle.

Odstranění tohoto problému je při stávající realizaci mechanismu, který je navázán na architekturu aplikace, relativně složité. To z důvodu, že jednotlivé klauzule mají kvůli znovupoužitelnosti v různých typech dotazů vlastní mechanismy načítání a je tak složitější udržovat kontext procházení a případně zjistit, co by nemělo být přeskočeno. Navíc vynechávání např. komentářů SPARQL dotazů je naopak žádoucí.

Dalším zjištěným problémem je načítání tzv. „Prefixed name“ (typ vkládané hodnoty – podobně jako proměnná či IRI), kdy Sparkle obvykle po vložení hodnoty ihned kontroluje regulárním výrazem (výraz udávající možné podoby vkládaných hodnot), zda je vložená položka odpovídající danému výrazu. Pokud není, pak je pole hodnoty barevně zvýrazněno. Problém je u regulárního výrazu pro typ „Prefixed name“, který je relativně složitý a u řetězců délky zhruba 20 znaků a více je již kontrola vloženého řetězce časově náročná. To má za následek, že by se načítání s takto dlouhými hodnotami výrazně zpomalilo. Tudíž je aktuálně při načítání SPARQL dotazů z textových souborů tato kontrola vynechána. Problém nastává tehdy, když je daný načtený prefix chybný, pak není tato chyba v programu zvýrazněna, dokud uživatel neprovede nějakou akci s daným textovým polem.

Řešením tohoto problému by bylo zjednodušení zmiňovaného regulárního výrazu, ovšem kvůli zachování souladu se SPARQL gramatikou je tento krok poměrně problematický.

6.2.2 Podpora vybraných částí SPARQL 1.1

Díky nově přidaným klauzulím jednotlivým typům dotazů narostl problém (tyto potíže jsou již zmiňovány v [1]) s místem na obrazovce (obzvláště u těch typů dotazů, které mají ve své struktuře více klauzulí). To způsobuje, že při vytváření složitějších SPARQL dotazů musí uživatel položky na obrazovce různě posouvat.

Další (obdobný) problém vzniká při větším počtu proměnných v klauzuli VALUES, kdy přestává být přehledná a tím pádem také použitelná pro běžné vytváření SPARQL dotazů. Samotný větší počet proměnných je závislý na velikosti použité obrazovky.

Oba tyto problémy by vyřešila jiná koncepce rozvržení prvků na obrazovce, což by však vyžadovalo rozsáhlý zásah do programu Sparkle.

6.2.3 Vyhledávání vlastností subjektů

Zjištěným problémem u tohoto rozšíření je přidávání typu hodnoty, která se vyskytuje na konci nalezené cesty (vlastnosti), i když daný typ není v RDF deklarován. Tento problém nastává např. v kódu 1.5, kdy je při vyhledání hodnoty „alice@email.com“ přidán typ „xsd:string“.

Důvodem tohoto jevu je pravděpodobně použitá knihovna „Jena“ (v programu Sparkle je používána pro práci s RDF a její popis se nachází na [56]), která při

vytváření modelu přidává typy jednotlivým uzlům. Tento jev nastává rovněž např. ve výsledcích SELECT dotazů.

6.3 Možná další rozšíření

Několik návrhů rozšíření programu Sparkle se nachází v kapitole 3. Cílem této části textu jsou pouze návrhy na úpravy rozšíření implementovaných v rámci DP.

Načítání SPARQL dotazů z textových souborů by bylo vhodné rozšířit zejména o mechanismus, který by kontroloval vynechané části dotazů, zda jsou vynechány oprávněně či jde o chybu načítání. Problém, který nastává absencí takového mechanismus ve Sparkle, je již zmíněn v sekci 6.2.1.

V rámci přidání podpory vybraných částí SPARQL 1.1 je vhodné rozšířit již přidanou podmnožinu novinek o další části. Kromě toho by mohly být lépe kontrolovány pole modifikátorů proměnných a agregačních funkcí v klauzuli pro výběr proměnných, které tvoří výsledky dotazů typu SELECT (popř. vnořených dotazů). Tyto modifikátory by mohly být kontrolovány ohledně správného užití závorek a samozřejmě vkládaných hodnot. Momentálně je kontrola prováděna až při spuštění vyhodnocování daného dotazu.

U vyhledávání vlastností subjektů by bylo vhodné upravit vytváření identifikačních SPARQL dotazů (viz sekce 4.3.3), které jsou užívány především v případech, kdy je subjekt vyhledávaných vlastností zadán pomocí proměnné. Úprava by spočívala ve výběru pouze těch částí WHERE klauzule, které se daného subjektu skutečně týkají. V současné době je zahrnována do identifikačního SPARQL dotazu celá klauzule, což má tu nevýhodu, že když je chyba v částech WHERE se subjektem nesouvisejících, tak zbytečně není dotaz vyhodnocen a subjekt identifikován.

Závěr

Cílem DP bylo vytvoření rozšíření do programu Sparkle. Celkem jsou implementována tři rozšíření, přičemž pokrývají načítání SPARQL dotazů z textových souborů, podporu podmnožiny nových možností SPARQL 1.1 a vyhledávání vlastností subjektů.

Prvním cílem práce bylo seznámení se s problematikou SPARQL a přidružených technologií, jako jsou RDF, OWL atd. Kromě studia těchto problematik bylo nutné dále prozkoumat samotný program Sparkle a další nástroje pro generování SPARQL dotazů. Zpracováním této části byl získán přehled o výše uvedených technologiích, programu Sparkle a dalších obdobných aplikacích z dané oblasti.

Dále byla analyzována aplikace Sparkle, zejména její architektura a funkce. Po provedení analýzy a díky získaným poznatkům z jiných programů pro generování SPARQL dotazů byl vytvořen výčet funkcionalit, kterými program Sparkle na počátku zpracovávání této DP nedisponoval.

Vytvořený seznam výhod ostatních aplikací pro tvorbu SPARQL dotazů, byl následně zkonzultován s vedoucím DP, čímž byly určeny tři rozšíření k realizaci. Realizovaná rozšíření, která jsou již uvedena výše, byla následně (s jistými omezeními, která jsou zmíněna v sekci 6.2) implementována do programu Sparkle.

Na závěr byly zhodnoceny dosažené výsledky, porovnána implementovaná rozšíření programu Sparkle s řešeními v jiných programech pro tvorbu SPARQL dotazů a vytvořen nástin dalšího rozvoje realizovaných rozšíření a samotné aplikace. Veškeré úpravy ve zdrojovém kódu programu Sparkle byly uloženy v repozitáři na serveru projektu MRE.

Použité zkratky

ANTLR – ANother Tool for Language Recognition
ASCII – American Standard Code for Information Interchange
BFS – Breadth-first search
DAML-ONT – DARPA Agent Mark-up Language
DAWG – Data Access Working Group
DFS – Depth-first search
FOAF – Friend of a Friend
IRI – Internationalized Resource Identifier
JavaCC – Java Compiler Compiler
JSON – JavaScript Object Notation
MRE – Medical Research and Education
OIL – Ontology Inference Layer
OWL – Web Ontology Language
RDBMS – Relational database management system
RDF – Resource Description Framework
RDFS – Resource Description Framework Schema
SPARQL – SPARQL Protocol and RDF Query Language
SQF – Sparkle query file
SVG – Scalable Vector Graphics
TSV – Tab-Separated Values
URI – Uniform Resource Identifier
W3C – World Wide Web Consortium
XML – eXtensible Markup Language
XSD – XML Schema Definition

Literatura

[1] ŠMUCR J. *G*rafická tvorba SPARQL. Plzeň, 2014. Diplomová práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd.

[2] RDF 1.1 Primer [online]. World Wide Web Consortium (W3C). [cit. 03.04.2017]. Dostupné z: `https://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`.

[3] SPARQL 1.1 Query Language [online]. World Wide Web Consortium (W3C). [cit. 03.04.2017]. Dostupné z: `https://www.w3.org/TR/sparql11-query/`.

[4] ČADA, Roman, Tomáš KAISER a Zdeněk RYJÁČEK. Diskrétní matematika. Plzeň: Západočeská univerzita v Plzni, 2004. ISBN 80-7082-939-7.

[5] Internationalized Resource Identifiers (IRIs) [online]. DUERST, M. a SUIGNARD, M. [cit. 03.04.2017]. Dostupné z: `http://www.ietf.org/rfc/rfc3987.txt`.

[6] FOAF Vocabulary Specification 0.99. [online]. BRICKLEY, D. a MILLER, L. [cit. 03.04.2017]. Dostupné z: `http://xmlns.com/foaf/spec/`.

[7] RDF/XML Syntax Specification (Revised) [online]. World Wide Web Consortium (W3C). [cit. 04.04.2017]. Dostupné z: `https://www.w3.org/TR/REC-rdf-syntax/`.

[8] Markup Language (XML) 1.0 (Fifth Edition) [online]. World Wide Web Consortium (W3C). [cit. 04.04.2017]. Dostupné z: `https://www.w3.org/TR/2008/REC-xml-20081126/`.

[9] RDF 1.1 Turtle [online]. World Wide Web Consortium (W3C). [cit. 05.04.2017]. Dostupné z: `https://www.w3.org/TR/turtle/`.

[10] RDF 1.1 N-Triples [online]. World Wide Web Consortium (W3C). [cit. 05.04.2017]. Dostupné z: `https://www.w3.org/TR/n-triples/`.

[11] Notation3 (N3): A readable RDF syntax [online]. World Wide Web Consortium (W3C). [cit. 07.04.2017]. Dostupné z: https://www.w3.org/TeamSubmission/n3/.

[12] SVÁTEK, V. Ontologie a WWW. VŠE DATAKON 2002, s. 1–35, ISBN 80-210-2958-7. [cit. 07.04.2017]. Dostupné z: http://nb.vse.cz/~svatek/onto-www.pdf.

[13] OWL 2 Web Ontology Language Primer (Second Edition) [online]. World Wide Web Consortium (W3C). [cit. 07.04.2017]. Dostupné z: https://www.w3.org/TR/2012/REC-owl2-primer-20121211/.

[14] Resource Description Framework (RDF) Model and Syntax Specification [online]. World Wide Web Consortium (W3C). [cit. 07.04.2017]. Dostupné z: https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[15] RDF Vocabulary Description Language 1.0: RDF Schema [online]. World Wide Web Consortium (W3C). [cit. 08.04.2017]. Dostupné z: https://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[16] RDF 1.1 Concepts and Abstract Syntax [online]. World Wide Web Consortium (W3C). [cit. 08.04.2017]. Dostupné z: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[17] RDF Schema 1.1 [online]. World Wide Web Consortium (W3C). [cit. 08.04.2017]. Dostupné z: https://www.w3.org/TR/rdf-schema/.

[18] OWL Web Ontology Language – Overview [online]. World Wide Web Consortium (W3C). [cit. 08.04.2017]. Dostupné z: https://www.w3.org/TR/owl-features/.

[19] OWL Web Ontology Language – Guide [online]. World Wide Web Consortium (W3C). [cit. 08.04.2017]. Dostupné z: https://www.w3.org/TR/owl-guide/.

[20] OWL 2 Web Ontology Language Document Overview (Second Edition) [online]. World Wide Web Consortium (W3C). [cit. 08.04.2017]. Dostupné z: https://www.w3.org/TR/owl2-overview/.

[21] OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition) [online]. World Wide Web Consortium (W3C). [cit. 10.04.2017]. Dostupné z: https://www.w3.org/TR/owl2-syntax/.

[22] OWL 2 Web Ontology Language Profiles (Second Edition) [online]. World Wide Web Consortium (W3C). [cit. 10.04.2017]. Dostupné z: https://www.w3.org/TR/owl2-profiles/.

[23] SPARQL Protocol for RDF [online]. World Wide Web Consortium (W3C). [cit. 10.04.2017]. Dostupné z: https://www.w3.org/TR/rdf-sparql-protocol/.

[24] SPARQL 1.1 Overview [online]. World Wide Web Consortium (W3C). [cit. 10.04.2017]. Dostupné z: https://www.w3.org/TR/sparql11-overview/.

[25] SPARQL Query Language for RDF [online]. World Wide Web Consortium (W3C). [cit. 10.04.2017]. Dostupné z: https://www.w3.org/TR/rdf-sparql-query/.

[26] vCard Ontology [online]. World Wide Web Consortium (W3C). [cit. 10.04.2017]. Dostupné z: https://www.w3.org/TR/2013/WD-vcard-rdf-20130502/.

[27] SPARQL 1.1 Update [online]. World Wide Web Consortium (W3C). [cit. 10.04.2017]. Dostupné z: https://www.w3.org/TR/sparql11-update/.

[28] Medical Research and Education – Sparkle – About [online]. [cit. 11.04.2017]. Dostupné z: https://mre.zcu.cz/sparkle/.

[29] A new version of Flint, the free, feature-rich SPARQL editor, is now available [online]. OpenUp. [cit. 11.04.2017]. Dostupné z: http://openup.tso.co.uk/blog/new-version-flint-free-feature-rich-sparql-editor-now-available.

[30] Describing Linked Datasets with the VoID Vocabulary [online]. World Wide Web Consortium (W3C). [cit. 11.04.2017]. Dostupné z: https://www.w3.org/TR/void/.

[31] RIETVELD, L. a HOEKSTRA, R. YASGUI: Not Just Another SPARQL Client. s. 78 [cit. 17.04.2017]. DOI: 10.1007/978-3-642-41242-4_7. Dostupné z: http://link.springer.com/10.1007/978-3-642-41242-4_7.

[32] OpenTriply/YASGUI: Yet Another Sparql GUI · GitHub [online]. [cit. 17.04.2017]. Dostupné z: https://github.com/OpenTriply/YASGUI.

[33] OAT Interactive SPARQL (iSPARQL) Query Builder [online]. OpenLink Software. [cit. 17.04.2017]. Dostupné z: http://wikis.openlinksw.com/OATWikiWeb/InteractiveSparqlQueryBuilder.

[34] Openlink/iSPARQL · GitHub [online]. [cit. 17.04.2017]. Dostupné z: https://github.com/openlink/iSPARQL/tree/master.

[35] Scalable Vector Graphics (SVG) 1.1 (Second Edition) [online]. World Wide Web Consortium (W3C). [cit. 17.04.2017]. Dostupné z: https://www.w3.org/TR/SVG/.

[36] STÉPHANE, C. Live SPARQL auto-completion. Proceedings of the 2014 International Conference on Posters & Demonstrations Track. p.477-480, October 21, 2014, Riva del Garda, Italy.

[37] Scampi/gosparqled: SPARQL auto-completion · GitHub [online]. [cit. 20.04.2017]. Dostupné z: https://github.com/scampi/gosparqled.

[38] YASR [online]. [cit. 20.04.2017]. Dostupné z: http://yasr.yasgui.org.

[39] Gruff: A Grapher-Based Triple-Store Browser for AllegroGraph. Franz Inc. - Semantic Graph and Common Lisp Solutions [online]. Franz Inc. [cit. 22.04.2017]. Dostupné z: https://franz.com/agraph/gruff/.

[40] Twinkle: A SPARQL Query Tool. Leigh Dodds [online]. [cit. 23.04.2017]. Dostupné z: http://www.ldodds.com/projects/twinkle/.

[41] dotNetRDF [online]. [cit. 23.04.2017]. Dostupné z: http://www.dotnetrdf.org.

[42] dotNetRDF – Home [online]. [cit. 24.04.2017]. Dostupné z: http://dotnetrdf.codeplex.com.

[43] AHO, Alfred V. Compilers: principles, techniques, & tools. 2nd ed. Boston: Pearson/Addison Wesley, c2007. ISBN 0-321-48681-1.

[44] ANTLR [online]. [cit. 10.10.2016]. Dostupné z: http://www.antlr.org.

[45] PARR, T. The definitive ANTLR 4 reference. Pragmatic programmers. ISBN 978-1-93435-699-9.

[46] ANTLR 4 Documentation [online]. [cit. 12.10.2016]. Dostupné z: https://github.com/antlr/antlr4/blob/master/doc/index.md.

[47] JavaCC – The Java Parser Generator [online]. [cit. 25.04.2017]. Dostupné z: https://javacc.org.

[48] SableCC [online]. [cit. 25.04.2017]. Dostupné z: http://www.sablecc.org.

[49] The Compiler Generator Coco/R [online]. System Software. [cit. 25.04.2017]. Dostupné z: http://ssw.jku.at/Coco/.

[50] Using JavaFX UI Controls: Menu – JavaFX 2 Tutorials and Documentation. Moved [online]. [cit. 26.04.2017]. Dostupné z: http://docs.oracle.com/javafx/2/ui_controls/menu_controls.htm.

[51] ANTLR v4 License [online]. [cit. 26.04.2017]. Dostupné z: http://www.antlr.org/license.html.

[52] SPARQL Pretty Printer 4 [online]. [cit. 26.04.2017]. Dostupné z: https://code.google.com/archive/p/sparkle-g/.

[53] Apache License, Version 2.0. Welcome to The Apache Software Foundation! [online]. [cit. 01.05.2017]. Dostupné z: `http://www.apache.org/licenses/LICENSE-2.0.html`.

[54] GRUNE, D. a Ceriel J. H. JACOBS. Parsing techniques: a practical guide. 2nd ed. New York: Springer, c2008. Monographs in computer science. ISBN 978-0-387-20248-8.

[55] CORMEN, T. H. Introduction to algorithms. 2nd ed. Cambridge, Mass.: MIT Press, c2001. ISBN 0-262-03293-7.

[56] Apache Jena [online]. [cit. 02.05.2017]. Dostupné z: `http://jena.apache.org`.

A Uživatelská dokumentace

V této části jsou popsány základní požadavky, postup instalace a možnost spuštění programu Sparkle. Kromě toho je zde uveden pouze popis používání přidaných rozšíření v rámci této DP. Uvedení do užívání ostatních funkcionalit aplikace Sparkle se nachází v [1] – sekce „Uživatelská dokumentace“.

A.1 Požadavky aplikace

Pro zajištění správného fungování programu Sparkle je nutné mít (podle [1]) „Java Runtime Environment[1]“ verze 1.8 nebo vyšší.

A.2 Sestavení a spuštění programu Sparkle

Možností sestavení projektu Sparkle je mnoho, jednou z nich je užití nástrojů „Apache Maven[2]“, „Git[3]“ a „Java Development Kit[4]“ (verze 8 a vyšší). Po nainstalování těchto nástrojů stačí pouze přesunutí příkazovou řádkou do kořenového adresáře projektu a zadat následující příkazy (v uvedeném pořadí):

```
mvn clean

mvn install
```

Po správném dokončení těchto operací je již program Sparkle připraven ke spuštění v adresáři „target“ projektu.

Pro spuštění aplikace Sparkle je nutné se přesunout do již uvedeného adresáře „target“ a zadat níže uvedený příkaz. V příkazu se nachází název souboru aplikace Sparkle „sparkle-6.0-SNAPSHOT“, ten se ovšem může v různých podobách projektu lišit. Zde uvedený název platí za předpokladu užití podoby projektu Sparkle z přiloženého přenosného média DP.

[1] `https://www.java.com/en/download/`

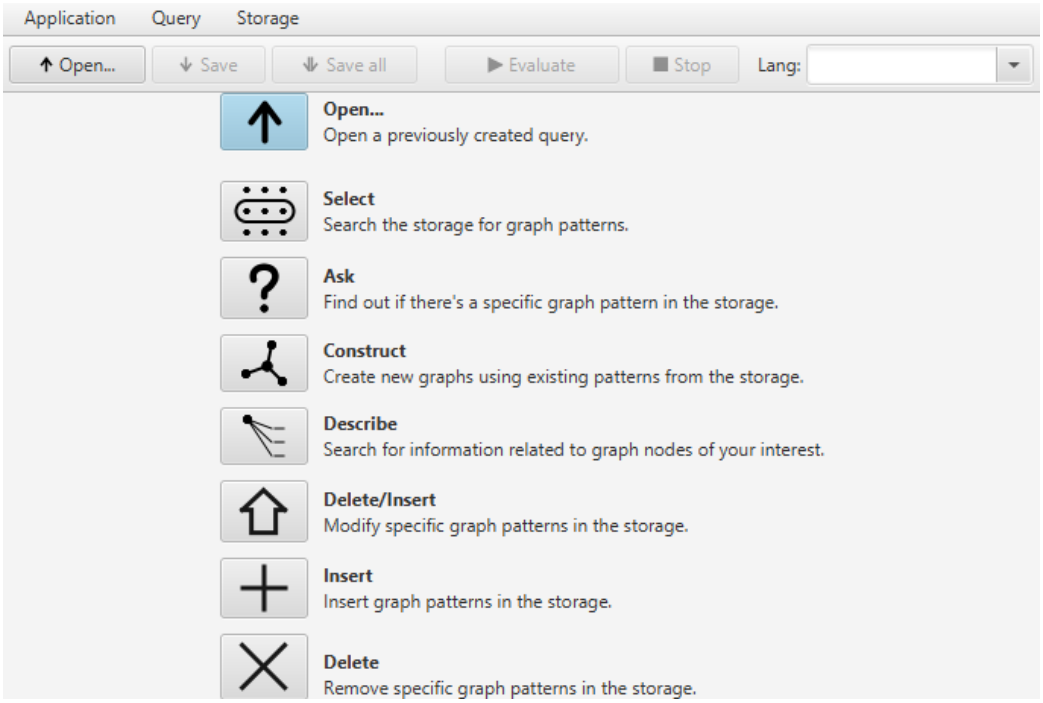
[2] `http://maven.apache.org/download.cgi`

[3] `https://git-scm.com/downloads`

[4] `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`

```
java -jar sparkle-6.0-SNAPSHOT.jar
```

Zpracováním tohoto příkazu je Sparkle spuštěn, přičemž je nejprve zobrazeno okno pro výběr úložiště (pro podrobnosti viz [1]). Po zvolení úložiště je již vyobrazeno hlavní okno aplikace – viz obr. A.1.



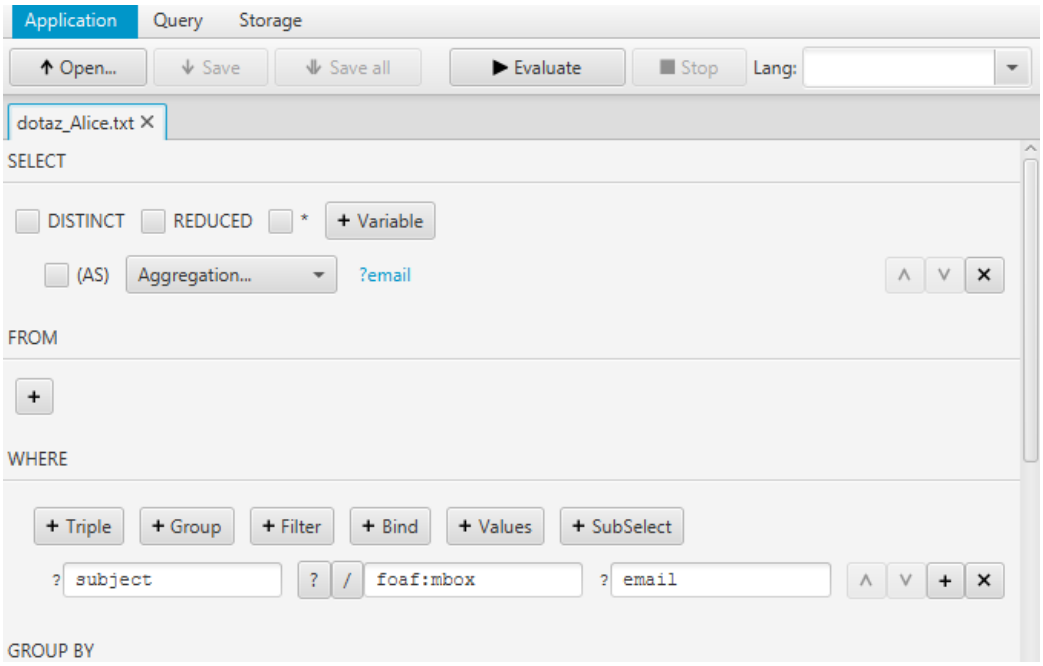
Obrázek A.1: Hlavní okno aplikace Sparkle

A.3 Používání implementovaných rozšíření

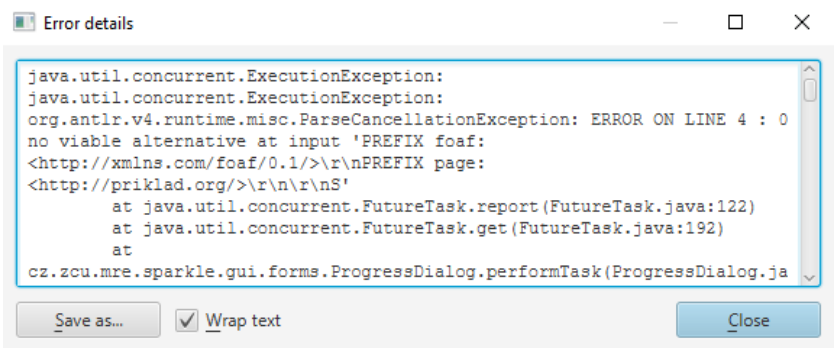
A.3.1 Načítání SPARQL dotazů ze souborů

Užití tohoto rozšíření je velmi jednoduché, jelikož stačí pouze kliknout na tlačítko „Open...“ (viz obr. A.1), čímž je otevřena nabídka pro výběr mimo jiné textového souboru. Výběrem souboru (obsahujícího SPARQL dotaz) je spuštěn mechanismus, jehož výsledkem je buď načtený dotaz či okno s výpisem chyby.

Vzhled načteného SPARQL dotazu z kódu 1.6 v programu Sparkle je zobrazen na obr. A.2. Naopak při zavedení chyby do stejného dotazu je vyobrazeno okno z obr. A.3 (chybová hláška je vždy vztažena ke konkrétní chybě).



Obrázek A.2: Načtený SPARQL dotaz z kódu 1.6



Obrázek A.3: Náhled okna s chybovým hlášením

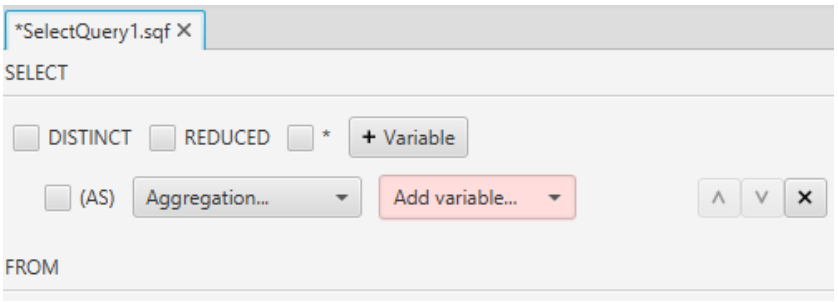
A.3.2 Podpora podmnožiny novinek SPARQL 1.1

V této části je popsáno základní užití podmnožiny implementovaných novinek ze SPARQL 1.1 v programu Sparkle. Pokud existuje více přidaných komponent se stejným užíváním, pak je ukázka pouze na jedné z nich a zbylé jsou jen jmenovány.

Agregační funkce

Do programu Sparkle jsou nově přidané agregační funkce (mezi které např. patří průměrná hodnota – AVG). Použití agregačních funkcí je možné především v dotazech typu SELECT či v nově implementovaných vnořených dotazech.

Pro užití v SELECT dotazu stačí vytvořit dotaz nový (kliknutím na tlačítko „Select“ na úvodní obrazovce programu Sparkle – viz obr. A.1) či načíst již existující z textového souboru. Přidání agregační funkce je možné provést kliknutím na tlačítko „+ Variable“, které se nachází v klauzuli pro výběr proměnných, jejichž záznamy mají být ve výsledcích dotazu (tlačítko je zobrazené na obr. A.4). Po kliknutí se přidá nová komponenta (vyobrazena na obr. A.4 pod tlačítkem „+ Variable“), která obsahuje rozbalovací nabídku s nápisem „Aggregation...“. Zvolením položky z této rozbalovací nabídky je přidána agregační funkce do daného dotazu.



Obrázek A.4: Náhled agregační funkce

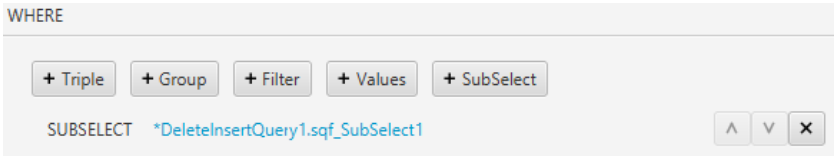
Po přidání agregační funkce je nutné hlídat pořadí závorek, které jsou momentálně automaticky přidávány jen na okrajích celého výrazu s agregační funkcí. To z důvodu maximální obecnosti klauzule při zadávání složitějších výrazů.

Jak již bylo zmíněno, tak agregační funkce je také možné využívat ve vnořených dotazech. Samotná manipulace s těmito komponentami je stejná jako u SELECT dotazu, jen je nutné nejdříve přidat nový vnořený dotaz, což je popsáno v další sekci.

Vnořené dotazy

Vnořené dotazy je možné vytvářet ve SPARQL operacích, které obsahují klauzuli WHERE (tedy např. SELECT či ASK). Ve WHERE klauzuli se nachází tlačítko „+ SubSelect“. Kliknutím na toto tlačítko je vytvořena nová záložka v programu Sparkle a odkaz na ni je ve WHERE klauzuli, ze které bylo vytváření vnořeného dotazu spuštěno.

Již zmíněný odkaz slouží především pro zobrazení relevantní záložky s vnořeným dotazem v programu Sparkle. Dále jím lze daný vnořený dotaz smazat, přičemž jeho smazáním dojde i k odstranění případných vnořených dotazů, které jsou na něj vázané (vysvětleno v sekci 4.2.2). Samotné smazání je možné iniciovat kliknutím na tlačítko označené křížkem u odkazu vpravo – viz obr. A.5, kde je vyobrazen i zmiňovaný odkaz na záložku vnořeného dotazu. Příklad definice záložky vnořeného dotazu se nachází v příloze C.



Obrázek A.5: Náhled odkazu vnořené funkce

Při vytváření dotazu (obsahujícího alespoň jeden vnořený dotaz) je možné využívat nápovědu, která (při napovídání proměnných daného dotazu) může obsahovat i proměnné vnořených dotazů. Do takové nápovědy jsou zařazeny jen ty proměnné, jejichž vnořený dotaz (respektive odkaz) je obsažen ve WHERE klauzuli daného dotazu, ve kterém je nápověda spouštěna. Další podmínkou je, že vnořený dotaz (ze kterého jsou proměnné získávány) musí mít ve své klauzuli pro definování vlastních výsledků (náhled klauzule je vyobrazen např. na obr. A.4) ony napovídané proměnné uvedeny.

Vyhodnocování SPARQL dotazu (obsahujícího vnořené dotazy) je proveditelné kliknutím na tlačítko „Evaluate“ (za předpokladu zobrazené záložky dotazu v programu Sparkle). Po vyhodnocení je vytvořena v programu Sparkle záložka s příslušnými výsledky. Dále je možné vyhodnocovat i samotné vnořené dotazy (vhodné zejména při odstraňování chyb). To lze uskutečnit spuštěním evaluace při zobrazeném vnořeném dotazu. Tím je vyhodnocován daný vnořený dotaz a jeho vnořené dotazy, které jsou na něj vázané (podrobnosti – viz sekce 4.2.2).

Ukládání dotazu s vnořenými dotazy se provádí např. kliknutím na „Save“. Pokud uložení není umožněno, pak je nutné zobrazit jinou část dotazu než vnořený dotaz (např. SELECT či ASK).

Příkazy DELETE DATA, INSERT DATA a DELETE/INSERT

Tyto SPARQL příkazy jsou implementovány tak, aby jejich ovládání bylo co nejvíce shodné s již existujícími typy dotazů v aplikace Sparkle, jejichž popis užití se nachází v [1]. Tudíž pro vytvoření např. operace DELETE/INSERT stačí kliknout na stejně pojmenované tlačítko na úvodní obrazovce programu Sparkle (viz obr. A.1). Po vytvoření se zobrazí struktura příkazu v podobě klauzulí, kterou je možné různě upravovat, ukládat, vyhodnocovat atd. Pro úplnost je v příloze E vyobrazen příklad struktury příkazu DELETE/INSERT pro kód 1.15.

U ostatních nově přidaných příkazů je ovládání totožné, jen tlačítka pro vytvoření mají příslušné názvy podle daného typu dotazu.

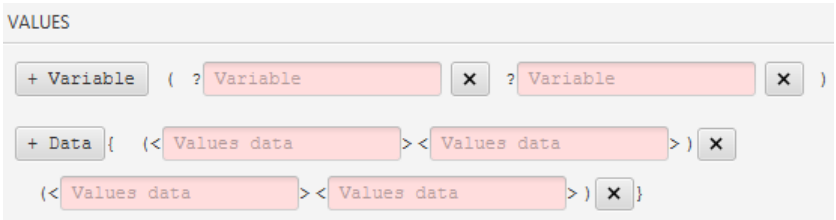
Klauzule VALUES

Do programu Sparkle bylo implementováno několik druhů dalších klauzulí, jejichž ovládání je koncipováno navzájem podobným způsobem, který je popsán níže v sekci A.3.2. Výjimku tvoří pouze klauzule VALUES, která je svým ovládáním odlišná, tudíž je v následující textu uvedena zvlášť.

Pro přidání klauzule VALUES např. do SELECT dotazu je nutné kliknout na tlačítko „+“ v příslušné části struktury vytvořeného dotazu. Poté se automaticky přidá pole pro vložení proměnné a patřičné závorky znázorňující část pro definici proměnných a část pro jejich data (podle [3]). Pro přidaní další proměnné je nutné kliknout na vytvořené tlačítko „+ Variable“.

Přidání nového pole pro data lze uskutečnit tlačítkem „+ Data“. Touto operací není vytvořeno čistě jedno pole, ale celá jedna varianta dat, která odpovídá vytvořeným proměnným.

Příklad na obr. A.6 zobrazuje dvě definované proměnné v klauzuli VALUES, ke kterým byly přidány dvě varianty dat (bylo pouze dvakrát kliknuto na tlačítko „+ Data“).



Obrázek A.6: Příklad klauzule VALUES

Ostatní přidané klauzule

Ve Sparkle jsou nově přidány klauzule, které jsou buď nutné pro vytvoření příkazů z A.3.2 (např. klauzule WITH či USING) nebo např. pro agregační funkce (GROUP BY, HAVING).

Všechny tyto klauzule mají podobné ovládání, kdy stačí kliknout na tlačítko s nápisem „+“, nacházející se v definici příslušné klauzule, čímž se vygeneruje pole pro zadání hodnoty dané klauzule. Některé klauzule umožňují (dle specifikace SPARQL – viz [3, 27]) přidání i více těchto polí. Pokud není v dané klauzuli žádná hodnota, pak je brána jako neexistující v daném dotazu. Odstranění daného pole pro zadání hodnoty klauzule je možné přidruženým tlačítkem s křížkem.

Kromě již uvedených klauzulí byla rovněž implementována např. BIND. Její zavedení do dotazu je možné kliknutím na tlačítko „+ Bind“, které je součástí WHERE klauzule. Z toho vyplývá, že je ji možné užívat pouze u těch typů dotazů (příkazů), které obsahují právě zmíněnou klauzuli WHERE. BIND obsahuje pole pro zadání výrazu a aliasu (názvu proměnné, která vystupuje v rámci dotazu namísto zmíněného výrazu).

Pole pro vyjádření aliasů má kromě BIND i např. klauzule GROUP BY, která je znázorněna níže na obr. A.7. Vyobrazení GROUP BY obsahuje pole pro zadání hodnoty i pro alias.

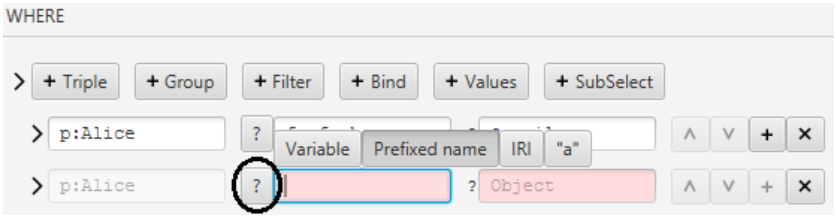
Obrázek A.7: Příklad GROUP BY klauzule

Mezi přidané klauzule patří např. BIND, GROUP BY, HAVING, USING, VALUES či WITH.

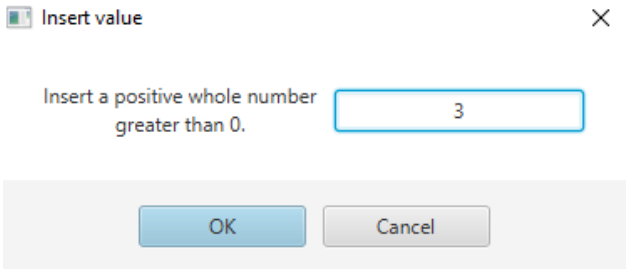
A.3.3 Vyhledávání vlastností subjektů

Vyhledávání vlastností je možné spustit z jakéhokoliv dotazu, který disponuje WHERE klauzulí s alespoň jednou trojicí. Tato trojice musí mít správně vyplněný subjekt a pole pro vlastnost nastaveno na typ „Prefixed name“. Poté již stačí pouze u příslušné trojice kliknout na tlačítko s nápisem „?“ a zadat hloubku prohledávání

(maximální počet vlastností v jedné cestě). Spouštěcí tlačítko je znázorněno na obr. A.8 a okno pro zadání hloubky prohledávání na A.9.

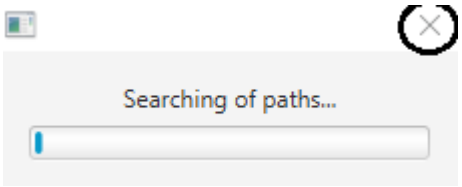


Obrázek A.8: Tlačítko pro spuštění mechanismu vyhledávání



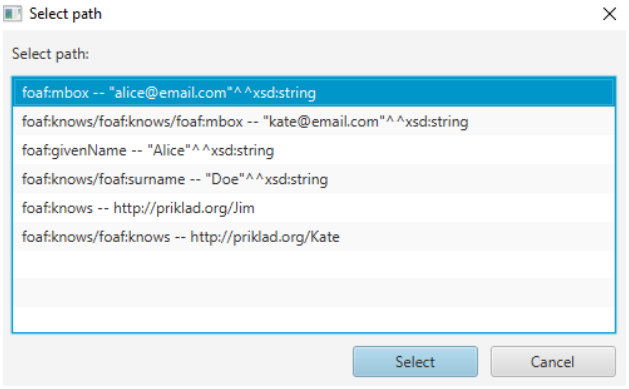
Obrázek A.9: Náhled okna pro zadání hloubky prohledávání

Po zadání hloubky je spuštěn mechanismus prohledávání, který je možné při dlouhotrvajícím zpracovávání ukončit kliknutím na křížek znázorněný na obr. A.10. Při předčasném ukončení jsou nalezené vlastnosti ztraceny.



Obrázek A.10: Průběh vyhledávání vlastností

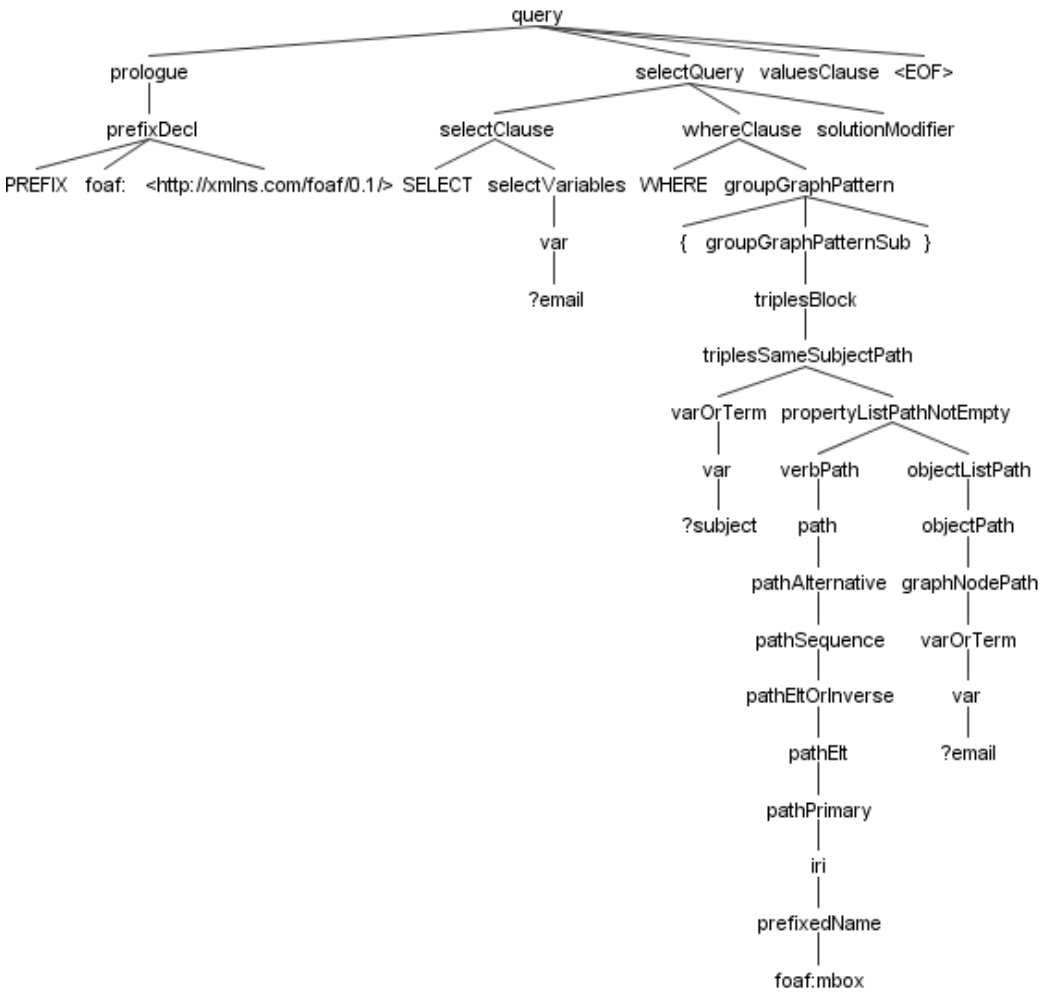
Při správném dokončení vyhledávání vlastností daného subjektu je zobrazeno okno se získanými hodnotami, které je znázorněno na obr. A.11. Pokud nejsou nalezeny žádné vlastnosti, je vráceno patřičné oznámení.



Obrázek A.11: Výpis vlastností pro proměnnou „?subject“ z příkazu 1.15 (při aplikaci na RDF z kódu 1.5)

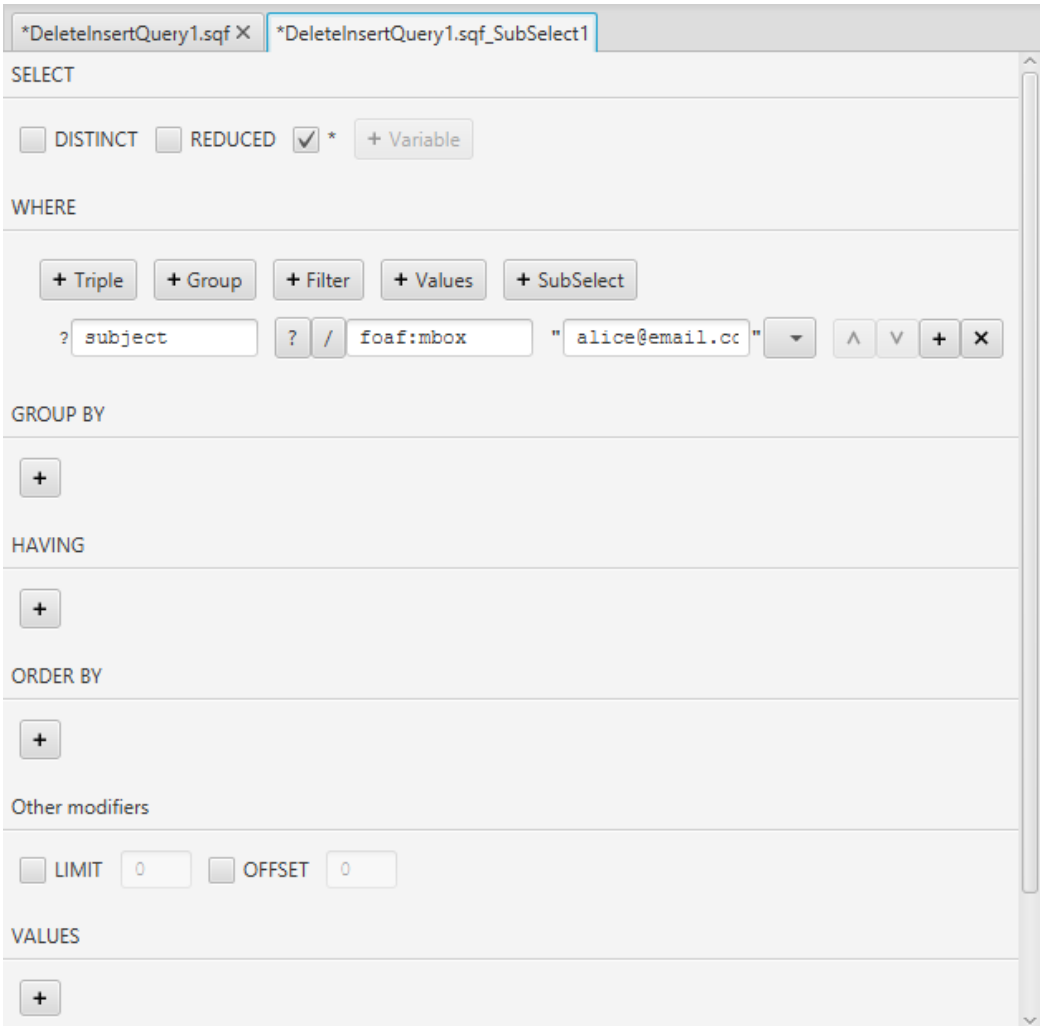
Výběrem nalezeného záznamu, skládajícího se z cesty a koncové hodnoty (odděleno pomocí „- -“), je daná cesta a její koncová hodnota automaticky vložena do příslušných polí trojice, ze které byl mechanismus vyhledávání spuštěn.

B Syntaktický strom



Obrázek B.1: Náhled syntaktického stromu, který je vytvořen (pomocí nástroje „SPARQL Pretty Printer 4“ [52]) na základě dotazu z kódu 1.6

C Záložka vnořeného dotazu



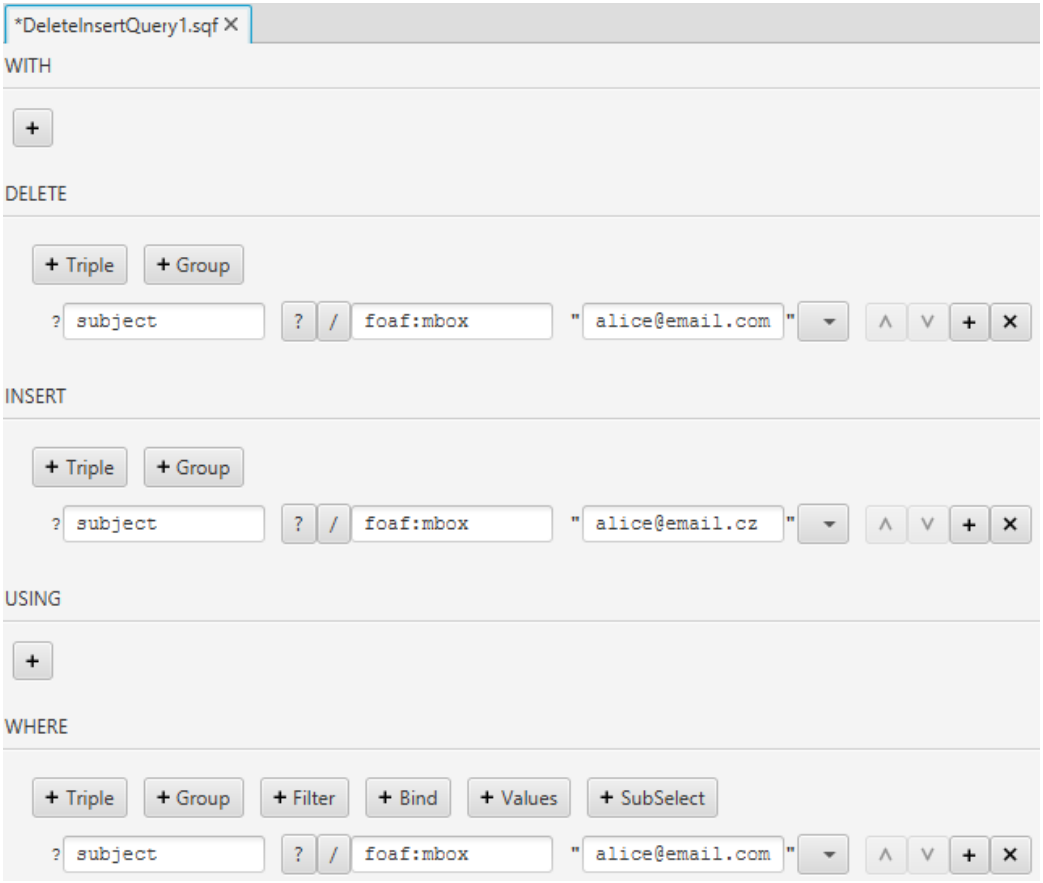
Obrázek C.1: Obsah záložky vnořeného dotazu, který je uzpůsoben příkladu z kódu 1.16

D Ukázka uloženého vnořeného dotazu

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Sparkle deleteWhere="0" tabColor="F4F4F4" type="deleteInsert">
  <WithGroup/>
  <DeleteInsertDelete/>
  <DeleteInsertInsert>
    <Triple>
      <Subject text="subject" type="variable"/>
      <Predicate text="foaf:surname" type="prefixedname"/>
      <Object text="Watson" type="literal"/>
    </Triple>
  </DeleteInsertInsert>
  <UsingGroup/>
  <Where>
    <SubSelect>
      <select>
        <globalVariable all="1"/>
      </select>
      <Where>
        <Triple>
          <Subject text="subject" type="variable"/>
          <Predicate text="foaf:mbox" type="prefixedname"/>
          <Object text="alice@email.com" type="literal"/>
        </Triple>
      </Where>
      <LimitOffset/>
      <OrderBy/>
      <GroupBy/>
      <Having/>
      <Values/>
    </SubSelect>
  </Where>
  <Prefixes>
  ...
```

Kód D.1: Ukázka uloženého příkladu z kódu 1.16

E Náhled DELETE/INSERT příkazu



Obrázek E.1: Náhled příkazu DELETE/INSERT z kódu 1.15 v programu Sparkle