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

Diplomová práce

Rozšíření Sparkle o podporu SPARQL Endpointu a využití titulků při tvorbě dotazu

Plzeň 2018 Klára Hlaváčová

Prohlášení

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

V Plzni dne 26. června 2018

Bc. Klára Hlaváčová

Abstract

The aim of this thesis is the design and implementation of Sparkle extensions. Sparkle is a tool for the creation and evaluation of queries in the SPARQL query language.The first part of this thesis contains a description of relevant technologies. Next the design of the extensions and the techniques for its implementation throught ANTLR and regular expressions are described as well as the current state of the application and an overview of other tools. The last part of this thesis describes the implementation of the extensions. The extensions included an enhancement of the integrated text editor, specifically syntax colouring, context help and query error visualization. Furthermore it contained the support of labels for query evaluation, user preferences setting and the support of SPARQL Endpoint connection. The contribution of this thesis is an increase in application usability, which may lead to higher work effectivity.

Abstrakt

Diplomová práce se zabývá návrhem a implementací rozšíření Sparkle, nástroje pro tvorbu a vyhodnocení dotazů v jazyce SPARQL. V první části je uveden popis relevantních technologií. Dále je proveden návrh rozšíření, popsány techniky použité pro jejich implentaci prostřednictvím ANTLR a regulárních výrazů, popsán současný stav aplikace a uveden přehled konkurenčních nástrojů. Nakonec je popsána implementace rozšíření. Rozšířila jsem integrovaný textový editor, implementovala podporu pro SPARQL Endpointy a používání titulků/popisků napříč celou aplikací. Úprava textového editoru zahrnuje obarvování syntaxe, kontextové napovídání a vizualizaci chyb v dotazu. Přínosem práce je zvýšení použitelnosti aplikace, což může vést k větší efektivitě práce.

Obsah

1 Úvod

SPARQL (SPARQL Protocol and RDF Query Language) je sémantický dotazovací jazyk a protokol pro dotazování a manipulaci s RDF (Resource Description Framework) daty. RDF je jedním z formátů sémantického webu. Sémantický web je reprezentací dat nejen na webu, rozšířením současného webu o metadata specifikujících přesný význam dat. Cílem sémantického webu je zpřístupnit data na webu pro strojové zpracování nebo např. umožnit integrovat data z různých zdrojů. Na specifikacích sémantického webu pracuje, spolu s dalšími, organizace World Wide Web Consortium (W3C)[1].

Sparkle, dostupný na [1], je grafický nástroj pro tvorbu a vyhodnocení dotazů ve SPARQL. Aplikaci začal vyvíjet v rámci své diplomové práce (dále DP) Jan Šmucr [2] v roce 2013 pod vedením Ing. Petra Včeláka, dále pokračovali oborovým projektem Michel Soběhart a DP a oborovým projektem Josef Kazák [3, 4].

Cílem diplomové práce je do aplikace implementovat další funkcionalitu. Konkrétně podporu dotazování na SPARQL Endpoint, nastavení preferencí, použití popisků (např. `rdfs:label` nebo `dc:title`) při konstrukci dotazu. Dále úprava integrovaného textového editoru, která zahrnuje obarvování syntaxe, kontextové napovídání, vložení šablon konstrukcí SPARQL, vizualizace chyb v dotazu.

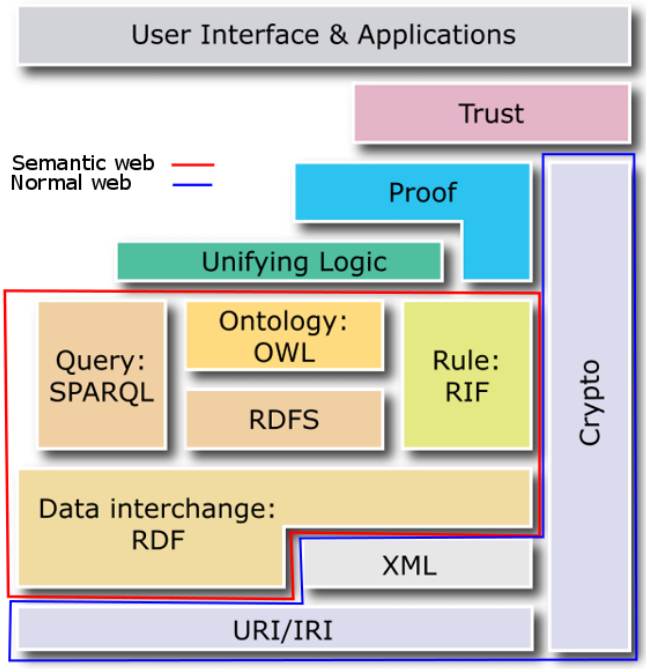
V první kapitolách budou stručně popsány základní technologie, tj. RDF, OWL a jazyk SPARQL a další pojmy, které souvisejí s tématem DP. Dále bude rozepsán současný stav aplikace Sparkle a popsány další existující nástroje pro práci se SPARQL. Následně bude proveden návrh požadovaných rozšíření, popsána jejich implementace a testování. Na závěr budou zhodnoceny dosažené výsledky a splnění požadavků.

[1] https://www.w3.org/

2 Resource Description Framework

Resource Description Framework (RDF, systém pro popis zdrojů) je framework (množina specifikací, model) pro vyjádření informací o zdrojích na webu. Obecně může být zdrojem cokoliv, co je jednoznačně idenfitikovatelné (dokumenty, lidé, fyzické objekty i abstraktní koncepty, čísla, řetězce, ...). Framework je vyvíjený organizací W3C [5, 6].

RDF je jedním z formátů sémantického webu (viz obr. 2.1). Sémantický web je reprezentací dat nejen na webu, rozšířením současného webu o metadata specifikujících přesný význam dat. Cílem je zpřístupnit (popsat) informace na webu pro strojové (automatické) zpracování, ne jenom pro zobrazení člověkem, tak aby se při přenosu dat z různých zdrojů neztrácel jejich význam. Příkladem použití může být automatické propojení několika datových množin do aplikace třetí strany a zpřístupnění tak rozsáhlejších informací pro uživatele z jednoho uživatelského rozhraní např. s možností agregací dat určitého tématu.



Obrázek 2.1: Architektura sémantického webu [1]

[1] Zdroj: http://www.w3.org/2007/03/layerCake.png

2.1 Datový model

RDF je založeno na atomickém prvku označovaném trojice (RDF triple, RDF statement):

`<subject> <predicate> <object>`,

který vyjadřuje (orientovaný) vztah mezi dvěma uzly RDF grafu. RDF graf je definován jako množina RDF trojic [5, 8].

Každá trojice tvoří jedno platné tvrzení. Tvrzení je pravdivé (platné) když existuje vztah (predikát) mezi objektem a subjektem a možina tvrzení je pravdivá, pokud jsou pravdivá všechna tvrzení v ní. Z platných tvrzení lze vyvozovat pravdivost jiných nebo odvodit tvrzení nová, případně ověřit konzistenci dat (s použitím tzv. sémantického reasoneru).

Subjekt (subject, resource, zdroj) je zdroj, který chceme popsat. Je reprezentován IRI (kap. 2.1.3) nebo prázdným uzlem (blank node, kap. 2.1.4).

Predikát (predicate, property, vlastnost) je vlastností subjektu. Je reprezentován IRI.

Objekt (object, property value, hodnota) je konkrétní hodnota vlastnosti. Je reprezentován IRI, literálem (kap. 2.1.2) nebo prázdným uzlem.

Protože stejný zdroj může být jako objekt v jedné trojici a jako subjekt v jiných, můžeme takto mezi trojicemi nacházet spojení. Zdroje musí být jednozačně identifikovatelné.

Množinu RDF trojic lze reprezentovat orientovaným ohodnoceným grafem (ve smyslu teorie grafů), kde uzly jsou zdroje a vlastnost tvoří orientované hrany mezi zdroji (od subjektu k objektu). K tvrzení (které je součástí grafu) můžeme přidat jako čtvrtou hodnotu název grafu a potom mluvíme o „čtveřici“ (RDF quad).

2.1.1 Příklad

Tvrzení „Bob zná Alici.“ a „Alice bydlí v Plzni.“ zapíšeme následovně: hodnota `Bob` je subjekt, vlastností je `knows` (zná) a hodnotou vlastnosti je `Alice`, analogicky i druhé tvrzení (Příklad 2.1):

```
<Bob> <knows> <Alice> .
<Alice> <lives_in> <Plzen> .
```

Příklad 2.1: RDF tvrzení

Na obrázku 2.2 je množina RDF trojic z příkladu 2.1 reprezentovaná jednoduchým grafem.

Obrázek 2.2: Graf množiny RDF trojic z příkladu 2.1

2.1.2 Literál

Literál [7] je obecná hodnota (která není IRI), např. řetězec („Alice“), datum (21. března 2015), číslo (5). Literál má přiřazený datový typ (`rdfs:Datatype`, např.: `"1"^^xs:integer`) a řetězce mohou být lokalizované (`rdf:LangString`, např.: `"hello"@en`).

2.1.3 IRI/URI

IRI/URI [8] je jednoznačný (globálně v daném rozsahu) identifikátor, který je přiřazen každému zdroji. Obecné schéma:

```
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
```

URI (Uniform Resource Identifier)

URI určuje schéma (`http`, ...) a místo v síti. Je-li URI absolutní je zároveň i IRI (naopak to platit nemusí). Př.: *http://neco.cz/str.html*.

IRI (Internationalized Resource Identifier)

IRI je UNICODE řetězec dle normy RFC 3987[2]. IRI je generalizace URI a v RDF musí být absolutní. Kvůli použití na webu je vhodné používat http(s) protokol. Př.: *http://neco.cz/bob*.

Příklad

RDF tvrzení z příkladu 2.1 s doplněnými IRI. RDF tvrzení je zapsáno v notaci N-Triples viz kapitola 2.3.1.

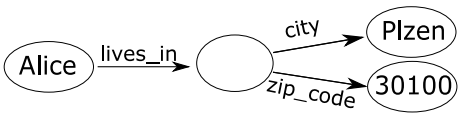
```
<http://example.org/Bob>
  ↪ <http://xmlns.com/foaf/0.1/knows> <http://example.org/Alice> .
```

Příklad 2.2: RDF tvrzení s URI

2.1.4 Anonymní uzel

V RDF grafu je jako prázdný či anonymní uzel (blank node [8]) označován uzel (zdroj), který neposkytuje veřejný identifikátor, lze na něj odkazovat jen přes jiné zdroje. Může se vyskytovat na pozici subjektu a objektu.

Příklad prázdného uzlu je na obrázku 2.3: „Alice žije v Plzni s PSČ 30100“.



Obrázek 2.3: RDF graf s prázdným uzlem

[2] https://www.ietf.org/rfc/rfc3987.txt

2.2 RDF slovníky

RDF datový model neobsahuje žádné informace o významu zdrojů - jejich sémantice. Např. predikáty uvedené v předchozích příkladech („knows“, „live_in“) mohou mít různý význam. Proto je nutné používat prostředky, které poskytnou informace o sémantice a tím zvýšíme znovupoužitelnost dat, např. veřejné slovníky nebo ontologie.

RDF poskytuje základní slovníky a ontologie (RDF built-in Vocabulary, RDF Schema, OWL, viz tabulka 2.1), které určují rámec pro popis zdrojů, tvoří jednoduchý jazyk. RDF slovníky jsou základním stavebním kamenem pro další slovníky/ontologie.

RDF slovníky (RDF vocabularies [5]) jsou kolekce IRI. Ontologie jsou formalizované slovníky, často pokrývající specifickou oblast. Níže je uveden příklad definice třídy `rdf:HTML` [14] z RDF Schema (RDFS).

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

rdf:HTML a rdfs:Datatype ;
  rdfs:subClassOf rdfs:Literal ;
  rdfs:isDefinedBy <http://www.w3.org/1999/02/22-rdf-syntax-ns#> ;
  rdfs:seeAlso <http://www.w3.org/TR/rdf11-concepts/#section-html> ;
  rdfs:label "HTML" ;
  rdfs:comment "The datatype of RDF literals ..." .
```

Příklad 2.3: Část RDFS - definice třídy `HTML`

Přehled základních RDF slovníků je v tabulce 2.1 [8]. Další lze najít např. v [12]. Jedním z prvních slovníku je „Friend of a Friend“ (prefix: `foaf`, jmenný prostor: http://xmlns.com/foaf/0.1/ [9]) sloužící pro popis sociálních sítí.

Tabulka 2.1: RDF slovníky

Prefix	**Jmenný prostor**	**RDF slovník**
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	RDF Vocabulary
rdfs	`http://www.w3.org/2000/01/rdf-schema#`	RDF Schema
skos	`http://www.w3.org/2004/02/skos/core#`	SKOS [10]
dc	`http://purl.org/dc/elements/1.1/`	Dublin Core [11]

2.2.1 RDF Vocabulary

RDF slovník (RDF built-in Vocabulary [5, 7]) obsahuje základní prvky pro popis zdrojů. Prefix a jmenný prostor jsou uvedeny v tabulce 2.1. V tabulce 2.2 a 2.3 jsou vypsány některé konstrukce.

Základní dělení konstrukcí:

- Třída (class): Skupiny souvisejících zdrojů. Jsou také zdrojem a můžou mít přiřazeno IRI. Instance třídy `rdfs:Class`.
- Vlastnost (property): Vztahy mezi subjektem a objektem, instance třídy `rdf:Property`.

Tabulka 2.2: Některé konstrukce RDF Vocabulary - třídy

Název	**Popis**
`rdf:Property`	Třída RDF vlastností.
`rdf:Statemant`	Třída RDF trojic.
`rdf:langString`	Třída řetězců s definovaným jazykem.

Tabulka 2.3: Některé konstrukce RDF Vocabulary - vlastnosti

Název	**Popis**
`rdf:type`	Stanovuje, že subjekt je instancí nějaké třídy.
`rdf:subject`	Subjekt RDF trojice.
`rdf:predicate`	Predikát RDF trojice.
`rdf:object`	Objekt RDF trojice.

2.2.2 RDF Schema

RDF Schema (RDF Schema Language, dále RDFS [5, 7]) je víceúčelový jazyk, který poskytuje podporu pro vytváření slovníků, umožnuje definovat sémantiku RDF dat. Obsahuje prostředky pro popis skupin souvisejících zdrojů (třídy) a vztahy mezi těmito zdroji (vlastnosti). Pomocí konstrukcí `domain` a `range` můžeme stanovit definiční obor a obor hodnot vlastností.

RDF Schema rozšiřuje RDF Vocabulary. Prefix a jmenný prostor jsou uvedeny v tabulce 2.1. V tabulce 2.4 a 2.5 jsou vypsány některé konstrukce. Oddělené jmenné prostory jsou již používány jen z důvodu zpětné kompatibility.

Tabulka 2.4: Některé konstrukce RDFS - třídy

Název	**Popis**
`rdfs:Resource`	Všechny zdroje jsou instance této třídy a je nadřazena ostatním třídám.
`rdfs:Class`	Každý zdroj, který je typu třída, je instancí `rdfs:Class`.
`rdfs:Literal`	Třída literálů - řetězců, čísel, ...

Tabulka 2.5: Některé konstrukce RDFS - vlastnosti

Název	**Popis**
`rdfs:subclassOf`	Modelování hierarchie mezi třídami.
`rdfs:subPropertyOf`	Modelování hierarchie mezi vlastnostmi.
`rdfs:domain`	Stanovuje třídu/y subjektu.
`rdfs:range`	Stanovuje třídu/y objektu.

2.3 Formáty zápisu

Pro zápis (serializaci) RDF dat existuje několik formátů [5]: N-Triples, Turtle, TriG, N-Quads, JSON-LD, RDFa, RDF/XML. K některým formátům je dále uveden stručný popis s příkladem s níže uvedenými trojicemi.

```
<Bob> <type> <Person> .
<Bob> <knows> <Alice> .
```

Příklad 2.4: Trojice v příkladech zápisu RDF dat

2.3.1 N-Triples

Jednoduchý formát, kde každá trojice je na jednom řádku. Jednotlivé prvky tvrzení jsou uzavřeny v <> a trojice je ukončena tečkou.

```
<http://example.org/Bob>
     ↪ <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
          ↪ <http://xmlns.com/foaf/0.1/Person> .
<http://example.org/Bob>
     ↪ <http://xmlns.com/foaf/0.1/knows>
          ↪ <http://example.org/Alice> .
```

Příklad 2.5: Příklad zápisu v N-Triples

2.3.2 Turtle

Turtle je rozšířením N-Triples a umožňuje data výrazně zpřehlednit pomocí prefixů. Relativní IRI (`<Bob>`) jsou vyhodnoceny oproti `BASE`. Množina trojic se stejným subjektem lze zapsat zkráceně, kde každá trojice je oddělena středníkem a konec množiny je označen tečkou.

```
BASE  <http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
<Bob>
    a foaf:Person ;
    foaf:knows <Alice> .
```

Příklad 2.6: Příklad zápisu v Turtle

2.3.3 RDF/XML

Formát RDF/XML vychází z XML syntaxe. XML element `rdf:Description` se používá k definování množiny trojic, které mají jako subjekt IRI uvedenou v atributu `rdf:about`. Každý element univtř `rdf:Description` odpovídá jedné trojici.

```
<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://example.org/Bob">
        <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
        <foaf:knows rdf:resource="http://example.org/Alice"/>
   </rdf:Description>
</rdf:RDF>
```

Příklad 2.7: Příklad zápisu v RDF/XML

3 Web Ontology Language

Web Ontology Language (OWL 1 [17], OWL 2 [15, 16], prefix: `owl`, jmenný prostor: `http://www.w3.org/2002/07/owl#`) je ontologický jazyk založený na deskripční logice poskytující bohatší vyjadřovací schopnosti než RDFS. V závislosti na použité sémantice můžeme na OWL pohlížet jako na jazyk rozšiřující RDFS.

Nejčastěji uváděná definice pojmu ontologie v oblasti informatiky je následující (T. Gruber [18]): „*Ontologie je explicitní specifikace konceptualizace*“ (konceptualizace - systém pojmů modelující určitou část světa). OWL ontologie jsou formalizované slovníky, pokrývající specifickou zájmovou oblast a jsou často šířené v komunitě této oblasti.

OWL oproti RDFS přidává relace mezi třídami, kardinalitu, univerzální a existenční kvantifikátor, rovnost, disjunktnost, ekvivalenci tříd, více typů vlastností, charakteristiky a vyjmenované třídy apod.

Poskytuje dvě specifikace sémantiky *Direct semantic* a *RDF-Based Semantic* a několik syntaxí (formátů) pro zápis (RDF/XML, OWL/XML, Functional Syntax, Manchaster Syntax, Turtle). OWL ontologie může být reprezentována RDF grafem (abstraktní strukturální forma a reprezentace RDF grafem jsou navzájem převoditelné).

Základními kameny OWL jsou axiomy (základní výroky), entity (prvky odkazující na objekty reálného světa) a výrazy (expressions, kombinace axiomů a entit). Nástroj pro odvozování znalostí z faktů definovaných v ontologii se nazývá OWL reasoner (nebo semantický reasoner).

Komponenty OWL:

- Jedinec (individual): Objekt v oblasti zájmu. Může patřit jedné nebo více třídám. Př.: „Karel“, „Tomáš“.

  ```
  :Karel rdf:type :Person; :livesIn :Pilsen .
  ```

- Vlastnost (property): Vztah mezi jedinci. Př.: „liveIn“, „hasChild“.

  ```
  :livesIn rdf:type owl:ObjectProperty .
  ```

- Třída (class): Množina jedinců se stejnými vlastnostmi. Př.: „person“, „man“.

  ```
  :Person rdf:type owl:Class .
  ```

3.1 OWL 1

Jazyk OWL 1 je rozdělen do tří verzí podle vyjadřovací síly. Vyšší verze rozšiřují nižší [17].

- OWL Lite: Nejjednoduší verze, obsahuje jen omezené množství konstrukcí. Pro aplikace, které potřebují jednoduché klasifikační hiearchie a omezení. Např. neumožňuje definovat kardinalitu jinou než 0, 1. Výpočetně efektivní.
- OWL DL: Vhodné pro ty, kteří vyžadují maximální vyjadřovací schopnosti, ale chtějí mít také zaručen výpočetní výkon (všechny výpočty skončí v konečném čase). Nejpoužívanejší jazyk.
- OWL Full: Pro ty, kteří chtěji maximální vyjadřovací shopnosti a nezávislost na syntaxi bez garance výkonnosti a správnosti výsledných tvrzení. Úplné sjednocení RDF a OWL. Verze DL a Full obsahuje stejnou množinu OWL konstrukcí, DL, ale obsahuje omezení v jejich použití a použití RDF konstrukcí.

3.2 OWL 2

Jazyk OWL 2 rozšiřuje OWL 1 a je s ním zpětně kompatibilní. OWL 2 profily jsou podjazyky OWL 2, obsahují různě omezené syntaktické podmnožiny vhodné pro specifické scenáře použití. Všechny jsou více restriktivní než OWL DL [15, 16].

- OWL 2 EL: Pro aplikace, které potřebují velké ontologie (s velkým množstvím tříd a vlastností). Poskytuje polynomiální odvozovací algoritmus. Použití např. v biomedicíně. Založen na deskripční logice EL++.
- OWL 2 QL: Pro jednodušší ontologie s velkým počtem individuálů, které je třeba organizovat, a pokud je nutné se dotazovat na data v RDBMS. Umožňuje dotazování s použitím technologie standardních relačních databází (např. SQL). Poměrně malé vyjadřovací schopnosti.
- OWL 2 RL: Pro systémy založené na odvozovacích pravidlech (rule based), které potřebují výkon, ale nevyžadují přilišné vyjadřovací prostředky. Také pro RDF aplikace, které potřebují nějaké vyjadřovací schopnosti navíc.

4 SPARQL

SPARQL (SPARQL 1.0 [19], SPARQL 1.1 [20]) je sémantický dotazovací jazyk a protokol (verze 1.0 [21], verze 1.1 [22]) pro manipulaci s RDF daty. Byl standardizován pracovní skupinou DAWG (RDF Data Access Working Group), součástí organizace W3C. V následujícím textu je uveden přehled některých možností, které SPARQL poskytuje.

SPARQL definuje dvě skupiny dotazů. Výběrové dotazy jsou dostupné od verze SPARQL 1.0, aktualizační od SPARQL 1.1. Syntaxe je podobná jazyku SQL a formátu `turtle`.

- Výběrové: `SELECT`, `DESCRIBE`, `ASK`, `CONSTRUCT`
- Aktualizační: `INSERT DATA`, `DELETE DATA`, `DELETE/INSERT`, `LOAD`, `CLEAR`

4.1 Schéma dotazu

V příkladu 4.1 je uvedeno schéma SPARQL dotazu, jednotlivé části budou popsány v následujících kapitolách.

```
# deklarace prefixu (prologue)
PREFIX name: <IRI>

# vystup dotazu
SELECT ...

# definice datove mnoziny (dataset)
FROM [NAMED] ...

# podminka - vzor grafu (graph pattern)
WHERE { ... }

# modifikator
LIMIT ...
```

Příklad 4.1: Schéma dotazu

4.1.1 Prologue

Prologue obsahuje definici prefixů, přiřazení prefixů k URI, které jsou použity v dalších částech dotazu.

Příklad

```
PREFIX ds:  <http://mre.zcu.cz/ontology/dasta.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

Příklad 4.2: Prologue

4.1.2 Definice zdroje

SPARQL dotaz se provádí nad RDF datovou kolekcí (RDF dataset), ta obsahuje jeden a více (nejméně jeden nepojmenovaný označovaný jako výchozí a 0-n pojmenovaných grafů) RDF grafů. Různé části dotazu lze vyhodnocovat nad různými grafy.

Konstrukcemi `FROM` a `FROM NAMED` můžeme explicitně definovat datovou kolekci/graf nad kterou se mají dotazy provádět. Výsledná kolekce obsahuje výchozí graf, který je sloučením grafů definovaných ve `FROM` klauzulích, a několik dvojic (IRI, graf) označujících pojmenované grafy z `FROM NAMED` klauzulí. Pokud jsou uvedeny jenom pojmenované grafy, je jako výchozí graf použit prázdný graf.

Syntaxe: `FROM [NAMED] <IRI>`

Příklad

```
SELECT ?nazev ?typ ?komentar
FROM <http://www.linkeddatatools.com/plants>
FROM NAMED <http://www.linkeddatatools.com/plants/orchid>
WHERE { ... }
```

Příklad 4.3: Definice zdroje - FROM klauzule

4.1.3 Graph Patterns

Požadovaný výsledek dotazu je definován množinou RDF trojic, které je označová jako *graph patterns*. Základní vzory lze kombinovat do složitějších. Jako výstup dotazu získáme většinou podgraf nebo množinu výsledků.

- Základní (basic):

  Množina trojic. Všechny trojice musí odpovídat. Další vzory jsou kombinací základního vzoru.

  Syntaxe: `bgp:= triples+`

  Př.: `?n a p:flowers . ?n rdfs:comment ?k .`

- Skupinové (group):

  Celá množina graph patterns musí odpovídat. Tento druh vzoru se vyskytuje nejčastěji v klauzuli `WHERE`. Obsah je uzavřený v {}. Množiny lze vnořovat.

  Syntaxe: `ggp:= '{'(ggp | bgp)* '}'`

  Př.: `{ ?n a p:flowers . ?n rdfs:comment ?k . }`

- Nepovinné (optional):

  Nepovinné vzory, které můžou rozšířit řešení, ale nemusí vždy vyhovovat. Vzor stanovíme klíčovým slovem `OPTIONAL`.

  Syntaxe: `ogp:= 'OPTIONAL' ggp`

  Př.: `{ ?n a p:flowers . OPTIONAL{?n rdfs:comment ?k .} }`

- Alternativní (alternative):

  Je definováno více možností. Musí odpovídat alespoň jedna. Klíčové slovo `UNION`.

  Syntaxe: `ggp:= ggp ('UNION' ggp)*`

  Př.: `{ { ?n a p:flowers . } UNION { ?n a p:scarts . } }`

- Vyloučení

  Podobnou funkcionalitu jako `NOT EXIST` (viz dále) poskytuje klíčové slovo `MINUS`, které odstraní z řešení graph pattern daný na pravé straně výrazu.

  Syntaxe: `ggp:= gp MINUS ggp`

  Př.: `{ ?n a p:flowers . MINUS { ?n a p:scarts . } }`

4.1.4 Filtry

Podgraf (graph pattern) definovaný v klauzuli `WHERE` můžeme dále omezit aplikací filtrů (`FILTR`). Filtr eleminuje ta řešení, pro která podmínka vrátí efektivní logickou hodnotu [19] `false` nebo chybu. Pro sestrojení podmínek SPARQL poskytuje podmnožinu funkcí a operátorů z XQuery[1] a XPath[2]. Kromě standardních operátorů lze, např. pro práci s řetězci, použít regulární výrazy apod.

Filter `NOT EXIST`/ `EXIST` testuje, jestli graph pattern nevyhovuje nebo vyhovuje datové množině vzhledem k proměnným, které se vyskytují v daném podgrafu.

Syntaxe:

```
'FILTR' '(' expresion ')'| BuiltInCall| FunctionCall
'FILTR' ('NOT EXIST'|'EXIST') '{' graphPattern '}'
BuiltInCall, FunctionCall - volani funkce - napr. str, lang, ...
```

Příklad

Následující dotaz vrátí názvy tříd a komentáře, které obsahují podřetězec "plant".

```
SELECT ?nazev ?komentar
WHERE {
  ?nazev rdfs:comment ?komentar .
  FILTER( regex(?komentar, ".[plant]." ))
}
```

Příklad 4.4: Fitrování hodnot

4.1.5 Modifikátory

Dotaz generuje neuspořádanou množinu řešení. Tato množina je pak zpracována jako seznam, na který jsou postupně aplikovány jednotlivé modifikátory (např. řazení) a nakonec je vrácena klientovi. Většina modifikátorů má smysl pouze u `SELECT`u, protože vrací množinu výsledků.

[1] https://www.w3.org/XML/Query/
[2] https://www.w3.org/TR/xpath-31/

- Řazení:

  Seřadí seznam řešení podle daného klíče.

  Syntaxe:

```
'ORDER BY' ( ( 'ASC' | 'DESC' ) '(' expr ')' ) |
    ( '(' expr ')' | constraint | var )
constraint: napr. SPARQL funkce - str, lang, ...
expr - vyraz - constraint, prommena, slozeny vyraz
var - promenna
```

  Některé termy, které nelze řadit uživatelsky, RDF řadí automaticky (RDF literal ← IRI ← Prázdný uzel ← proměnná nebo výraz bez přiřazené hodnoty). Pořadí mezi některými RDF termy není stanoveno (např. "`abc`" a "`abc`"`@en`, "`abc`" a "`abc`"^^`xsd:string`, ...).

- Projekce:

  Omezení řešení stanovením podmnožiny proměnných, které se budou vracet uživateli.

```
SELECT ?X ?Y
```

- Vyřazení/omezení neunikátních řešení:

  Klíčová slova: `DISTINCT, REDUCED`. Distinct neunikátní řešení odstraní, redukce jenom povolí jejich vyřazení (ve výpisu řešení se tedy duplicita může zobrazit).

```
SELECT DISTINCT ?X ?Y
```

- Limit, offset:

  Klíčová slova: `LIMIT, OFFSET`. Stanovení počtu řešení a posun začátku v seznamu řešení.

```
WHERE { ... } LIMIT 10 OFFSET 2
```

- Agregace:

  Klíčové slovo: `GROUP BY`. Určuje, podle kterých proměnných se provádí agregace, tj. sloučení výsledků do skupin. Agregační funkce ve SPARQL 1.1: `MIN, MAX, COUNT, SUM, AVG, GROUP_CONCAT`.

```
SELECT (COUNT(?a) AS ?aCount) WHERE { ... } GROUP BY ?b
```

Příklad

Je provedena projekce vyřazením proměnné `typ` ze seznamu proměnných. Dále budou výsledky seřazeny sestupně podle proměnné `nazev` a při shodě vzestupně podle komentáře a počet výsledků je omezen na 10.

```
SELECT ?nazev ?komentar
WHERE {
  ?nazev a p:flowers ;
         a ?typ .
  OPTIONAL { ?nazev rdfs:comment ?komentar . }
} ORDER BY DESC(?nazev) ?komentar
LIMIT 10
```

Příklad 4.5: Modifikátory

4.2 Základní syntaxe

Kompletní specifikace lze nalézt v [19] pro SPARQL 1.0 a v;[20] pro 1.1, níže je uvedena základní syntaxe.

- Proměnná: Prefix `?` nebo `$` a název: `?nazev`. Může nabývat jakékoliv hodnoty. Globální dosah.
- IRI: `<IRI>`, base adresa: `<>`. IRI může být relativní i absolutní. Př.: `<http://example.com/bob#me>`
- Jméno s prefixem: `prefix:name`
- Literály: „`string`“, „`string`“`@en`, `45`, „`string`“`^^xsd:integer`, „`true`“
- Prázdný uzel: `_:n` nebo `[]`, `n` je lokální název v rámci skupinového graph pattern.

  Př.: `_:s :p ‘o’ .`, `[ :p ‘o’ ] :p1 :o1 .`
- Trojice: RDF trojice. Trojice se stejným subjektem oddělíme ‘;’, se stejným subjektem a predikátem ‘,’. Konec je označen ‘.’.

  `?subjekt ?predikat ?objekt; ?predikat ?objekt .`
- RDF kolekce: `(element1 element2 ...)`, pro kolekci je alokovaný prázdný uzel.

- `rdf:type` můžeme zkrátit na `a`
- Operátory: unární (+,-,!), logické (||,&&), aritmetické (+,-,*,/), relační (<,>, ...).
- SPARQL funkce: `str`, `lang`, `datatype`, ...
- SPARQL testy: `bound`, `isIRI`, ...

4.3 Výběrové dotazy

Výběrové dotazy jsou dostupné od verze SPARQL 1.0. Typy: `SELECT`, `DESCRIBE`, `ASK`, `CONSTRUCT`. Pro příklady dotazů bude použita OWL ontologie z příkladu B.1 (příloha B, str. 79).

4.3.1 Dotaz typu SELECT

Vrací množinu nebo podmnožinu proměnných inicializovaných v části graph pattern. Uvedením '*', místo seznamu proměnných, budou zahrnuty do výsledku všechny proměnné s přiřazenou hodnotou. Pokud chceme provést projekci, vypíšeme jenom určité proměnné.

Syntaxe:

```
'SELECT' (['DISTINCT' | 'REDUCED']) ('*' | ('?'var (' ''?'var)*))
    'WHERE' '{' triples+ '}'
```

Příklad

```
BASE <http://www.linkeddatatools.com>
PREFIX p: <plants#>

SELECT ?nazev ?typ ?komentar
WHERE {
  ?nazev a p:flowers ; a ?typ .
  OPTIONAL { ?nazev rdfs:comment ?komentar . }
} LIMIT 10
```

Příklad 4.6: Dotaz typu SELECT

Dotaz vrátí množinu výsledků ukázanou v tabulce 4.1. IRI jsou kvůli přehlednosti zkráceny.

Tabulka 4.1: Výsledek dotazu SELECT

nazev	**typ**	**komentar**
<orchid>	<flowers>	"Orchidej"@cz
<orchid>	<flowers>	"Orchid"@en
<magnolie>	<flowers>	

4.3.2 Dotaz typu CONSTRUCT

Vrátí graf sestavený podle definovaných šablon. Graf vznikne spojením jednotlivých trojic operací `UNION`. Lze použít pro získání podgrafu, odvození dat, transformace dat. Výsledek lze zobrazit v různých notacích.

Syntaxe:

```
'CONSTRUCT' '{' [ triples+ ] '}' 'WHERE' '{' triples+ '}'
```

Příklad

```
BASE <http://www.linkeddatatools.com>
PREFIX p: <plants#>

CONSTRUCT { } WHERE {
  ?nazev a p:flowers ; a ?typ .
}
```

Dotaz vrátí RDF graf uvedený v příkladu 4.7.

```
@prefix ...

p:orchid   a  p:flowers .
p:magnolia a  p:flowers .
```

Příklad 4.7: Výsledek dotazu CONSTRUCT

4.3.3 Dotaz typu ASK

Vrátí true/false, podle toho jestli vzor dotazu má řešení.

Syntaxe:

```
'ASK' '{' triples+ '}'
```

Příklad

Výsledek dotazu bude logická hodnota `true`.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
BASE <http://www.linkeddatatools.com>
PREFIX p: <plants#>

ASK { p:magnolia p:similarlyPopularTo p:orchid }
```

Příklad 4.8: Dotaz typu ASK

4.3.4 Dotaz typu DESCRIBE

Pro zdroje, uvedené v řešení, vytvoří RDF graf připojením informací z dostupných datových zdrojů. Není potřeba znát strukturu RDF dat, tu určí procesor SPARQL dotazu. Lze uvést i explicitně IRI.

Syntaxe:

```
'DESCRIBE' (('*' | ('?'var (' ''?'var)*)) 'WHERE' '{' triples+ '}')
    | ('<'iri'>')+
```

Příklad

```
BASE <http://www.linkeddatatools.com>
PREFIX p: <plants#>

DESCRIBE ?X
WHERE { p:magnolia p:similarlyPopularTo ?X . }
```

Příklad 4.9: Dotaz typu DESCRIBE

Dotaz vrátí RDF graf uvedený v příkladu 4.10.

```
@prefix ...

p:orchid a                  p:flowers ;
       rdfs:comment         "Orchid"@en , "Orchidej"@cz ;
       p:family             "Orchidaceae" ;
       p:similarlyPopularTo p:magnolia .
```

Příklad 4.10: Výsledek dotazu DESCRIBE

4.4 Aktualizační dotazy

SPARQL 1.1 umožňuje změnu obsahu úložiště [23], tj. smazat, vložit, ... RDF graf / graph pattern. Typy: `INSERT DATA`, `DELETE DATA`, `DELETE/INSERT`, `LOAD`, `CLEAR`

4.4.1 Dotaz typu DELETE DATA

`DELETE DATA` smaže definovaný graph pattern z RDF grafu. Pokud je výsledný graf prázdný, tak ho neodstraní. Graph pattern nemůže obsahovat proměnné a blank node.

Syntaxe:

```
'DELETE DATA' '{' graphPattern '}'
```

Příklad

```
DELETE DATA { p:magnolia a "abc" . }
```

Příklad 4.11: Dotaz typu DELETE DATA

4.4.2 Dotaz typu INSERT DATA

`INSERT DATA` vloží definovaný graph pattern. Pokud graf neexistuje, tak ho založí. `INSERT DATA` může obsah blank node.

Syntaxe:

```
'INSERT DATA' '{' graphPattern '}'
```

Příklad

```
INSERT DATA { p:magnolia a "abc" . }
```

Příklad 4.12: Dotaz typu INSERT DATA

4.4.3 Dotaz typu INSERT/DELETE

`INSERT/DELETE` obsahuje příkazy `INSERT`, `DELETE`, které vloží a smažou graph pattern, ale narozdíl od `INSERT/DELETE DATA` obsahuje i klauzuli `WHERE`. Odstraňované trojice jsou tedy založené na vazbách definovaných vzorem ve `WHERE` klauzuli.

Syntaxe:

```
['WITH' (prefixeName | iri)]
'DELETE' '{' [graph pattern] '}' 'INSERT '{' [graph pattern] '}')
['USING [NAMED] (prefixedName | iri))+ ]
'WHERE '{' graph pattern'}'
```

Příklad

```
WITH <g1> DELETE { p:a p:b p:c } INSERT { p:x p:y ?b }
 WHERE { ?b a "aaa" }
```

Příklad 4.13: Příkaz DELETE DATA

4.5 Další možnosti SPARQL 1.1

Níže jsou stručně uvedeny některé další možnosti, které SPARQL 1.1 nabízí.

- Poddotazy: SPARQL 1.1 umožňuje zanořit `SELECT` poddotaz do dotazu.

  ```
  WHERE { ?dd ds:p ?rc { SELECT * WHERE { ?d ?b ?v }} }
  ```

- `BIND`: dovoluje přiřadit hodnotu proměnné z graph pattern nebo výrazu.

  ```
  BIND (?p*(1-?discount) AS ?price)
  ```

- `VALUES`: dovoluje specifikovat několik proměnný v datovém bloku a přiřadit jim hodnotu/y.

  ```
  VALUES { ?a } { ("ab") ("bc") }
  ```

- `HAVING`: filtruje data, ale narozdíl od `FILTER` pracuje nad group graph pattern.

  ```
  SELECT (AVG(?a) AS varA) WHERE  ?x :a ?a  GROUP BY ?x
  HAVING (AVG(?a) > 5)
  ```

4.6 SPARQL Endpoint

RDF data jsou uchovávána v RDF uložišti (triple store). úložiště mohou podporovat persistentní i nepersistentní (in-memory) modely uchování dat. Příklad RDF uložišť: OpenLink Virtuoso, Sezame, Apache Jena, ...

Většina uložišť podporuje vzdálený přístup přes HTTP protokol. Poskytují službu, SPARQL Protocol Service, která obsluhuje příchozí požadavky. Adresa, na které SPARQL Protocol service naslouchá, se nazývá SPARQL Endpoint.

SPARQL Endpointy jsou webové přístupové body k uložištím. Umožňují vzdálené dotazování, správu úložiště. Existuje řada veřejně dostupných endpointů, které organizace zpřístupňují přes webový editor.

Seznam endpointů lze nalézt např.: http://sparqles.ai.wu.ac.at/. V tabulce 4.2 je uvedeno několik příkladů z toho seznamu.

Tabulka 4.2: Příklady SPARQL Endpointů

Název	**Adresa**
DBpedia	http://dbpedia.org/sparql
Bio2RDF::Wormbase	http://wormbase.bio2rdf.org/sparql
GeoLinkedData	http://geo.linkeddata.es/sparql
RUIAN	http://ruian.linked.opendata.cz/sparql

5 Editory

V následující kapitole budou popsány existující nástroje pro dotazování ve SPARQL, práci s RDF daty a ontologiemi. Pokud není uvedeno jinak, tak byl popis získán vyzkoušením editoru.

5.1 Sparkle

Sparkle [1] je grafický nástroj pro tvorbu a vyhodnocení dotazů ve SPARQL. Aplikace je vyvíjena v rámci projektu MRE (Medical Research and Education) na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Vývoj začal Jan Šmucr [2] v roce 2013 pod vedením Ing. Petra Včeláka, dále pokračovali oborovým projektem Michel Soběhart a DP a oborovým projektem Josef Kazák [3, 4].

Výčet funkcionality (aktuální k 27. 5. 2018, verze 2018.6) je sestaven podle seznamu na webu programu a výše zmíněných pracích. Seznam zahrnuje i rozšíření, které jsem implementovala v oborovém projektu [28] a v semestrální práci předmětu *Databázové systémy a metody zpracování informací 2* [29]. Tyto funkce budou v seznamu označeny zkratkami OP a DBMS.

5.1.1 Stávající funkce

- Zadání dotazu: Vizuální (formulář) nebo po vyhodnocení ručně v textovém editoru.
- Možnosti datového zdroje: lokální (souborový systém, in-memory model), vzdálené (Virtuoso Universal Server).
- Vyhodnocení dotazu přímo v aplikaci (možnost přerušení vyhodnocení).
- Zobrazení výsledků ve formě tabulky (fulltexové vyhledávání, třídění, filtrování), textového výpisu (podle typu dotazu).
- Obarvení záhlaví souvisejících záložek (např. dotaz a výsledek vyhledávání).

- Podporované konstrukce SPARQL 1.0 i 1.1 (výčet v [1]).
- Vyčištění aktuálního grafu (smazání všech trojic).
- Možnosti konfigurace (na úrovni aplikace nebo dotazu): správa zdrojů, prefixů, funkcí, datových typů,
  - správa zdrojů (ontologií/slovníků) - načtení zdrojů ze souboru / lokálního úložiště,
  - předdefinované prefixy, funkce, datové typy a možnost vložit vlastní.
- Formulář pro sestavení dotazu,
  - prvky pro jednotlivé části dotazu,
  - vyhledávání vlastností - našeptávač,
  - kontextová nápověda,
  - zvýraznění chybných částí dotazu.
- Editor (zobrazen až po vyhodnocení dotazu),
  - obarvení syntaxe (částečně),
  - kontrola syntaxe při vyhodnocení dotazu.
- Načtení/uložení dotazu,
  - vlastní formát Sparkle Query File (SQF), dotaz lze uložit a znova načíst i s konfigurací (prefixy, ...),
  - z/do textového souboru, podpora komentářů (#, /**/).
- Datové formáty pro import RDF dat,
  - RDF/XML (*.rdf, *.xml), OWL (*.owl), N3 (*.n3), N-TRIPLE (*.nt), TURTLE (*.ttl).
- Datové formáty pro uložení výsledku dotazu,
  - `SELECT`: zobrazení v aplikaci - tabulka; CSV (kontextová nabídka nad záhlavím sloupců),
  - `CONSTRUCT`, `DESCRIBE`: stejné jako pro import dat.
- *DBMS*: Statistiky (histogram, číselné statistiky, tabulka četností).
- *OP*: Uživatelské nastavení aplikace,
  - jazyk aplikace,

  - možnost změnit typ odvozování v in-memory modelu,
  - preferované typy popisků (viz použití popisků)

- *OP*: Správa ontologií (správa zdrojů a prefixů)
  - správa zdrojů - načtení z IRI (hromadně/jednotlivě) a uložení ontologií/slovníků do souboru (stažení z IRI, hromadně/jednotlivě),
  - správa prefixů - prefix, IRI, alternativní IRI; načtení prefixů z CSV souboru / jednotlivě a uložení do souboru.
- *OP*: Nápověda - dialogy s informacemi o aplikaci, dokumentací, příklady apod.

5.2 Flint SPARQL Editor

Flint SPARQL Editor[1] je webově založený editor poskytující vyhodnocení dotazu a zobrazení výsledků v několika formátech (tabulka nebo sparql-xml, text, json), obarvování syntaxe, jednoduchou kontextovou nápovědu ve formě kontextového menu a tabulky, ve které se zvýrazňují přípustné hodnoty, zvýrazňování chyb. Dále obsahuje tlačítka pro vložení šablon několika typů dotazů, historii a vytvoření nového dotazu. Dotaz je vyhodnocován přes zvolený SPARQL Endpoint (nutné zadat ručně). Dotazy je možné zobrazit v několika záložkách. Pravděpodobně není udržovaný.

- Vyvíjí: TSO[2]
- Verze: 1.0.3, 2012
- Demo: http://fr.dbpedia.org/sparqlEditor/index.html
- Podpora: SPARQL 1.0, 1.1 (Query, Update)

5.3 YASGUI

YASGUI (Yet Another SPARQL GUI)[3] je webově založený editor, který poskytuje vyhodnocení dotazu a zobrazení výsledku v několika formátech

[1] https://github.com/TSO-Openup/FlintSparqlEditor
[2] http://www.tsonline.co.uk/
[3] https://doc.yasgui.org/

(tabulka; responze - json, xml, csv, tsv, rdf/xml, ...; Google Chart; Pivot table; Geo). Uložení výsledku do souboru (txt, csv, json, html). Zvýrazňování syntaxe a chyb.

U tabulkového zobrazení umožňuje fulltextové vyhledávání ve výsledcích, řazení podle sloupců, omezení délky výpisu. Pivot table umožňuje zobrazit různé číselné statistiky, histogram, ... Dále obsahuje zobrazení historie a návrat k předchozím dotazům. Dotazy je možné zobrazit v několika záložkách.

Vyhodnocení je provedeno přes zvolený SPARQL Endpoint, umožňuje výběr ze seznamu endpointů a nastavení připojení.

- Vyvíjí: Triply[4]
- Verze: 2.7.21, 9. únor 2018
- Demo: http://yasgui.org
- Podpora: SPARQL 1.0, 1.1 (Query)

5.4 iSPARQL

Hlavním účelem iSPARQL[5] je poskytovat webovou vrstvu (rozhraní) pro RDF úložiště (např. DBpedia).

iSPARQL editor [25] obsahuje vizuální vytváření dotazu (kreslení grafu) i textový editor. Načtení dotazu ze souboru a jeho uložení. Interaktivní nápovědu (vkládání částí dotazu výběrem z menu), historii. Zobrazení výsledků ve formě *svg* grafu, interaktivní tabulky (záznamy umožňují přejít na podrobnosti), seznamu trojic, klasické tabulky, obrázku a případně mapy podle charakteru dat. Připojení ke vzdálenému uložišti (Virtuoso) nebo přes SPARQL Endpoint (výběr z nabídky).

- Vyvíjí: OpenLink Software[6]
- Verze: verze 1.30, 2014
- Demo: https://www.openlinksw.com/isparql

[4] https://triply.cc/
[5] https://github.com/openlink/iSPARQL
[6] https://www.openlinksw.com/

- Podpora: SPARQL 1.0, 1.1 (Query - SELECT, CONSTRUCT, DESCRIBE; Update - INSERT, DELETE)

5.5 Gruff

Gruff[7] je nástroj, desktopová aplikace, pro správu uložišť a analýzu dat založený na databázi AllegroGraph. Umožňuje zobrazit data vizuálně i v textové podobě. Dotazy mohou být též vytvářeny v textovém i grafickém editoru. Nabízí i API pro použití v aplikaci. Podporuje připojení ke SPARQL Endpointu.

- Vyvíjí: Franz Inc.[8]
- Verze: 7.2.1.
- Demo: https://franz.com/agraph/gruff
- Podpora: SPARQL 1.0, 1.1 (Query, Update); Prolog

5.6 Twinkle: SPARQL Tools

Twinkle[9] [26] je jednoduchý editor, desktopová aplikace, pro vyhodnocování SPARQL dotazů.

Umožňuje načtení dotazu ze souboru, vložení prefixů (včetně konfigurace vlastních namepsace), přerušení dotazu za běhu, uložení dotazu do souboru. Umožňuje se dotazovat na RDF data v lokálních souborech (lokální úložiště) i vzdálených RDF dokumentech (vzdálené úložiště), v relačních databázích, připojení přes SPARQL endpoint (tři pevně dané adresy). Uložení výsledku. Umožňuje použit odvozování (RDF Schema, OWL ontologie, ...) při vyhodnocení dotazu. Obsahuje uživatelskou konfiguraci datových zdrojů.

- Vyvíjí: Leigh Dodds

[7] https://franz.com/agraph/gruff/
[8] https://franz.com/
[9] http://www.ldodds.com/projects/twinkle/

- Verze: 2.0
- Demo: http://www.ldodds.com/projects/twinkle
- Podpora: SPARQL 1.0 (Query);

5.7 Další nástroje

Na webu je dostupná řada dalších nástrojů a webových editorů, které často vychází ze stejného základu.

Některé další nástroje pro správu RDF dat, ontologií a dotazování ve SPARQL:

- Apache Jena - Fuseki (https://jena.apache.org/)
- QueryVOWL (http://vowl.visualdataweb.org/queryvowl/)
- Fluent Editor (http://www.cognitum.eu/semantics/FluentEditor/)
- rdfEditor (https://bitbucket.org/dotnetrdf/dotnetrdf/wiki/UserGuide/Tools/rdfEditor)
- TextMate (https://github.com/peta/turtle.tmbundle)
- protégé (https://protege.stanford.edu/)

6 Analýza

V následujícím textu bude provedena analýza a návrh rozšíření popsaných v předchozí kapitole, tj. podpora SPARQL Endpointu, použití popisků a rozšíření integrovaného textového editoru.

6.1 Požadavky na nové funkce

Níže jsou popsány funkce, které budou implementovány v rámci DP.

- Úložiště/vyhodnocení,
  - podpora dotazování na SPARQL Endpoint,
  - výběr ze seznamu, vyhledávání v seznamu,
  - správa SPARQL Endpointů, načtení ze souboru a ruční zadání,
  - změna úložiště za běhu,
  - zákaz vyhodnocení dotazu při chybě v editoru/formuláři.
- Popisky,
  - použití popisků (`rdfs:label, ...`) při konstrukci dotazu (zobrazení v kontextové nápovědě, volba jazyka a typu popisku),
  - hromadné načtení zdrojů/popisků z url (vylepšení správy prefixů),
  - zobrazení popisků/protypů ve správě zdrojů, funkcí, datových typů,
  - správa preferovaných jazyků popisků v nastavení aplikace.
- Editor,
  - na hlavní obrazovce (v současné verzi je k dispozici až po vyhodnocení dotazu),
  - synchronizace formuláře a editoru,
  - obarvování syntaxe, zvýraznění stejných identifikátorů,
  - vizualizace chyb za běhu - označení a tooltip s popisem chyby,
  - vkládání závorek v páru,
  - kontextová nápověda,

    - editor - podle kontextu před kurzorem a typu dotazu
    - formulář - podle typu vstupního pole (jenom úprava vzhledu),
    - vložení šablon konstrukcí SPARQL,
    - zobrazení popisků a nápovědy pro klíčová slova, ... v kontextové nápovědě,

- Úprava validace polí ve formuláři.

6.2 Podpora SPARQL Endpointu

Program v současnosti dovoluje vykonání dotazu a správu dat v lokálním uložišti (in-memory model se zálohováním na disk) a vzdáleně (Virtuoso). V rámci tohoto rozšíření bude přidáno dotazování na SPARQL Endpoint (viz kapitola 4.6), výběr ze seznamu existujících SPARQL Endpointů a jeho správa. Dále přepnutí typu úložiště za běhu aplikace.

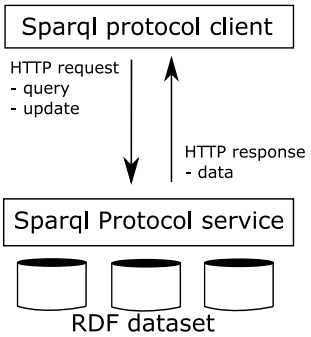
SPARQL Protocol [21], protokol pro komunikaci mezi klientem a SPARQL procesorem, umožňuje vzdálené provedení dotazu přes HTTP protokol. Obsahuje dvě HTTP operace: `query` pro výběrové dotazy a `update` pro aktualizační dotazy.

6.2.1 Princip komunikace

Princip (obr. 6.1): SPARQL Protocol client (aplikace) pošle HTTP request s dotazem, SPARQL Protocol service požadavek zpracuje a pošle zpátky HTTP response s daty. URI, na které SPARQL Protocol service naslouchá se nazývá SPARQL endpoint.

Dotazy (request) lze odesílat pomocí `GET` i `POST`, doporučená je ale metoda `GET`. Odpověď (data) je vrácena ve specifikovaném formátu podle typu dotazu. Např. `SPARQL/XML` pro `SELECT`, `ASK` nebo `Turtle`, `RDF/XML` pro `CONSTRUCT`, `DESCRIBE`.

Při úspěchu/selhání operace používá protokol standardní HTTP stavové kódy (2xx, 3xx, 4xx, 5xx). Pro ochranu úložiště může být nastaven timeout.



Obrázek 6.1: SPARQL Protocol - princip komunikace

6.2.2 Příklad komunikace

Níže je uveden příklad komunikace se SPARQL Endpointem (převzato z [21]) - kód dotazu (6.1), HTTP request (6.2) a HTTP response (6.3).

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?book ?who WHERE { ?book dc:creator ?who }
```

Příklad 6.1: Dotaz

```
GET /sparql/?query=PREFIX...&default-graph-uri=...
HTTP/1.1
Host: www.other.example
User-agent: my-sparql-client/0.1
```

Příklad 6.2: HTTP request

```
HTTP/1.1 200 OK
Date: Fri, 06 May 2005 20:55:12 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.4 DAV/1.0.3
Connection: close
Content-Type: application/sparql-results+xml

<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
 <head><variable name="book"/> <variable name="who"/> </head>
 <results> <result> ... </result> <results>
</sparql>
```

Příklad 6.3: HTTP response

6.2.3 Přepnutí úložiště za běhu aplikace

V předchozí verzi je možné změnit typ úložiště pouze při startu aplikace. Protože, ale bude program umožňovat připojení ke SPARQL Endpointu a je pravděpodobné, že bude docházet k přepínání mezi nima je tento způsob pro uživatele dost nepohodlný. Bude proto přidána možnost přepnutí úložiště za běhu aplikace, aby šlo jednoduše přecházet mezi uložišti i při otevřených dotazech.

Přepnutí lze realizovat např. formou tlačítka, které vyvolá formulář pro připojení k uložišti. Pokud budou při přepnutí otevřeny dotazy, je nutné u nich zajistit aktualizaci propojení s úložištěm.

6.3 Zobrazování popisků

Zdroje ve slovnících/ontologiích je vhodné popsat popisky (anotacemi), které jsou srozumitelnější člověku než pouhý název (resp. IRI) a lze je jednoduše překládat do více jazyků. Existují různé konstrukce pro zápis popisků podle použitého slovníku/ontologie: `dc:title`, `rdfs:label`, `skos:prefLabel`, ...

V příkladu 6.4 je ukázána definice vlastnosti `dasta:contact` z DASTA ontologie[1] s popisky `rdfs:label` v angličtině a češtině.

```
<owl:ObjectProperty rdf:about="http://mre.zcu.cz/ontology/dasta.owl
    #contact">
   <rdfs:domain rdf:resource="http://mre.zcu.cz/ontology/dasta.owl#
      Address"/>
   <rdfs:label xml:lang="en">contact</rdfs:label>
   <rdfs:label xml:lang="cs">kontakt</rdfs:label>
</owl:ObjectProperty>
```

Příklad 6.4: Použití popisků v ontologii

Bude implementována možnost zobrazení popisků (`rdfs:label`, ...) pro všechny třídy a vlastnosti při konstrukci dotazu. Dále nastavení preference jazykové varianty popisků dle seznamu jazyků a typu popisku.

[1] http://mre.kiv.zcu.cz/ontology/dasta.owl

6.3.1 Možnosti použití popisků

Jsou v zásadě dva scénáře, kdy lze popisky použít. Při konstrukci dotazu a při zobrazení výsledků dotazu.

Konstrukce dotazu

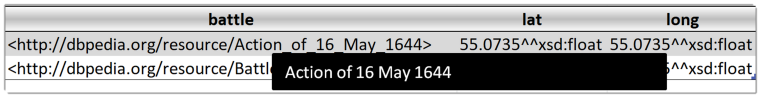
Zobrazení popisků (společně s názvy zdrojů) při konstrukci dotazu umožní uživateli jednodušeji vyhledávat zdroje jenom se znalostí prefixu.

Popisky můžeme standardně zobrazit v kontextové nápovědě. Pokud je předem načteme do úložiště spolu se zdroji, nebude nutné vykonávat žádné dotazy navíc. Kromě popisků zdrojů je vhodné vložit i popis dalšího obsahu nápovědy, jako názvů funkcí, klíčových slov, ... Příklad kontextové nápovědy se zobrazenými popisky je na obrázku 6.2.

Obrázek 6.2: Použití popisků při konstrukci dotazu [2]

Zobrazení výsledků dotazu

Popisky mohou také sloužit jako popis zdrojů v tabulce s výsledky dotazu `SELECT`. Protože všechny zdroje ve výsledku se nemusejí nacházet v lokálním uložišti, byly by nutné další doplňující dotazy. Tento způsob implementován nebude. Příklad puoužití popisku ve výsledcích dotazu je na obrázku 6.3.



Obrázek 6.3: Použití popisků při zobrazení výsledků dotazu

[2] Zdroj: [33]

6.3.2 Preferovaný jazyk a typ popisků

Popisky mohou být v ontologiích/slovnících různého typu (`skos:prefLabel`, `dc:title`, `rdfs:label`, ...) a v různých jazykových variantách (nebo bez explicitně uvedeného jazyka).

Aby byla zajištěna co největší variabilita, bude přidána možnost volby jazyka titulků a typu titulků ve formě seznamů. Při načítání zdrojů do úložiště se poté vybere první existující varianta jazyka a typu titulku a ta se uloží.

6.4 Rozšíření textového editoru

V této kapitole budou popsány možnosti implementace rozšíření textového editoru. Cílem rozšíření je usnadnit práci s editorem, zvýšit přehlednost kódu a poskytnout nápovědu při tvorbě dotazu.

Do editoru bude doplněno: kontextová napověda, možnost vložení šablony vybraných konstrukcí SPARQL, vkládání závorek v páru, obarvování syntaxe a zvýraznění výskytů slova pod kurzorem, zvýraznění chyb v dotazu při psaní. Bude provedeno přenesení textového editoru na hlavní obrazovku (aby šel použít i bez nutnosti předešlého vyhodnocení dotazu) a bude probíhat synchronizace mezi editorem a formulářem, včetně aktualizace prologue při přepnutí z editoru do formuláře.

Synchronizace mezi editorem a formulářem

Editor je v současné verzi k dispozici až po vyhodnocení dotazu. Prvním krokem proto bude sjednocení editoru, formuláře, výsledků a statistik do jedné záložky. Čímž se zvýší celková přehlednost a zároveň použitelnost editoru. Dále je nutné provádět synchronizaci mezi editorem a formulářem. Tj. aktualizaci editoru po přepnutí z formuláře a naopak. Protože dotaz může obsahovat poddotazy (`subselect`), které jsou v samostatných záložkách, je potřeba provádět synchronizaci ve směru editor - formulář při každém přepnutí z editoru na jinou záložku v rámci aktuální záložky s dotazem i mezi založkami dotazů.

Parsování dotazu

Některá uvedená rozšíření vyžadují předzpracování textu dotazu. Pro parsování textu, jeho rozdělení na tokeny (skupiny znaků, které jsou seskupeny na základě sémantického pravidla) a identifikaci typu tokenu (klíčové slovo, url, literál, ...) se nabízejí dvě možnosti.

První možností je použít regulární výrazy, tj. sestavit regulární výrazy pro jednotlivé typy tokenů, které se vyskytují v textu, a poté je použít k jejich nalezení. Druhá možnost je sestrojit překladač a provést lexikální analýzu.

6.4.1 Regulární výrazy

Regulární výrazy slouží jako vzor pro popis sekvence znaků, umožňují jednoduše vyhledávat vzory v textu. W3C poskytuje kompletní popis gramatiky SPARQL, podle které lze jednoduše regulární výrazy pro jednotlivé typy tokenů sestavit.

Regulární výrazy umožňují použití tzv. pojmenovaných skupin (named capture group) a jejich řetězení do alternativ (viz př. 6.5).

```
syntaxRegex
= "(?<KEYWORD>" + keywordPattern + ")"
+ "|(?<COMMENT>" + commentPattern + ")"
+ "|(?<RESOURCES>" + resourcesPattern + ")"
+ "|(?<AGGREGATION>" + aggregationPattern + ")"
+ "|(?<FUNCTION>" + functionPattern + ")"
```

Příklad 6.5: Regulární výraz pro vyhledání skupin vzorů

Tento způsob je vhodný pro vyhledávání jednoho nebo malého množství vzorů, které se od sebe liší. Ve větším rozsahu bude kód hůře udržitelný. Navíc by záleželo na pořadí vyhodnocování jednotlivých skupin (např. pokud vzory začínají stejným prefixem) a některé by se nemuseli identifikovat správně.

Regulární výrazy budou použity k vyhledání jednoho vzoru, ne jako náhrada parseru. Např. při zvýrazňování slov, která jsou stejná jako slovo pod kurzorem.

6.4.2 ANTLR4

Vhodnějším způsobem je sestrojit gramatiku a provést lexikální analýzu a případně i syntaktickou, pokud jsou nutné další informace. Parser je možné sestavit ručně nebo použít nějaký nástroj, který implementaci usnadní, např. ANTLR (Another Tool for Language Recognition).

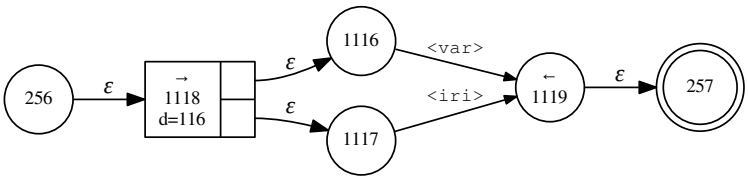
ANTLR[3] [27] je nástroj, který umožňuje ze zadané gramatiky vygenerovat adaptivní LL(*) parser (syntaktický a lexikální analyzátor) pro zpracování textu nebo binárního vstupu. Podporované cílové jazyky jsou C++, Java, C#, JavaScript a další.

Vygenerovný parser poskytuje callback interface (př. 6.6), což umožňuje jednoduše parsovat vstup způsobem řízeným událostmi (event-driven). Metody, které rozhraní poskytuje, umožňují provést akci při vstupu nebo výstupu z kontextu daného pravidla (při probíhající analýze textu).

```
void enterPrologue(SparqlParser.PrologueContext ctx);
void exitPrologue(SparqlParser.PrologueContext ctx);
```

Příklad 6.6: Příklad metod poskytnutých callback rozhraním parseru

V ANTLR4 je struktura gramatiky součástí parseru (ATN - Augmented Transition Network). ATN je hierarchická struktura kterou lze procházet, a která poskytuje kontext jednotlivých pravidel gramatiky a další informace. Na obrázku 6.4 je zobrazen ATN graf pravidla `varOrIRI : var | iri ;`. Čísla představují identifikátory jednotlivých stavů.



Obrázek 6.4: Příklad ATN grafu pravidla gramatiky "varOrIri" [4]

ANTLR parser/lexer bude použit při zvýrazňování syntaxe, filtrování obsahu kontextové nápovědy a vizualizaci chyb a některých dalších činnostech souvisejících s kontrolou syntaxe.

[3] http://www.antlr.org/

[4] Vygenerováno ANTLR4 (https://manpages.debian.org/stretch/antlr4/antlr4.1.en.html)

Zachytávání a zotavení se z chyb

ANTLR parser umožňuje zachytávat lexikální/syntaktické chyby (pokud parser skončí v chybovém stavu je vyhozena výjimka) a také se dokáže z řady chyb zotavit (error recovery).

Zotavování se z chyby je proces, kdy se překladač snaží dostat z chybového stavu do známého stavu (který může v daném kontextu nastat), ze kterého může pokračovat. Korektního stavu můžeme dosáhnout např. konzumací symbolů, změnou vstupního proudu a porovnáváním nového stavu s množinou symbolů, která může následovat. Takovéto jednání může být výhodou i nevýhodou, např. pokud používáme informace o chybě při konstrukci kontextové nápovědy, tak se může stát, že se použije chybná množina symbolů, protože překladač skončí jinde než očekáváme. Při zvýrazňování chyb je naopak zase samozřejmě vhodné, aby překladač dokázal pokračovat co nejdéle.

Pokud implementujeme vlastního posluchače pro zachytávání chyb, můžeme získat řadu informací a díky zotavení chybová hlášení i do jisté míry akumulovat. Toto může být využito při vizualizaci chyb i filtrování kontextové nápovědy.

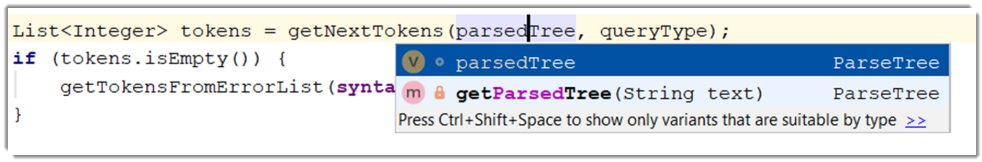
6.4.3 Kontextová nápověda

Kontextová nápověda (nápověda citlivá na kontext) nebo automatické doplňování kódu (code completion) je užitečný doplněk vyskytující se v některých editorech. Umožňuje uživateli sestavit kód/dotaz i bez hlubší znalosti klíčových slov, případně syntaxe. Většinou je implementována ve formě kontextového menu (seznamu), které se objeví při psaní nebo stisku klávesové zkratky *"CTRL+mezerník"* u pozice kurzoru. Příklad nápovědy ve formě kontextového menu je na obrázku 7.6.

Slova, která se mají zobrazit v nápovědě, je vhodné uložit do slovníku, ve kterém se bude následně vyhledávat.

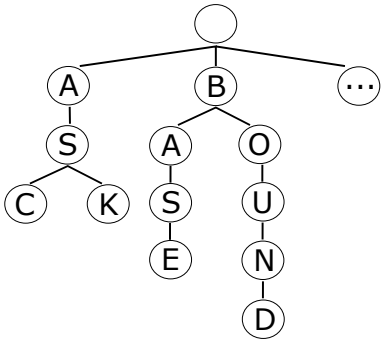
`ADT`[5] `Trie` (obr. 6.6) je stromová struktura, která umožnuje efektivní uložení řetězců a jejich rychlé vyhledávání (v čase úměrném délce vyhledávaného řetězce). Další možností by bylo řetězce ukládat např. do hashovací tabulky

[5] Abstraktní datový typ



Obrázek 6.5: Kontextová nápověda v programu IntelliJ IDEA[6]

(`Hash table`). Vyhledávání a uložení dat v `ADT Trie` je efektivnější (např. lze jednoduše vyhledávat prefixy), proto preferuji tuto variantu.



Obrázek 6.6: Příklad ADT Trie

V kontextové nápovědě budou dostupné prefixy, prefixovaná jména zdrojů, funkce, datové typy, názvy proměnných, klíčová slova, operátory apod. a šablony pro některé konstrukce.

Konstrukce obsahu nápovědy

Sestavení obsahu nápovědy probíhá ve třech fázích. Statický obsah nápovědy (základ, který se mění jen zřídka) je vhodné vytvořit jednou při zobrazení editoru (popsáno výše) a následně přidávat/odebírat hodnoty, které se mění, např. názvy proměnných. Při zobrazení nápovědy, se v tomto slovníku vyhledají slova začínající daným prefixem (částí slova, která je před kurzorem) nebo se načte celý obsah (prefix je prázdný). Nakonec se seznam filtruje podle upřesňujících kritérií - kontextu, sémantiky. Sestavení nápovědy musí být tak rychlé (zvlášť vyhledávání ve slovníku, řazení, apod., protože zpracovávaný seznam může být dlouhý), aby nezdržovalo zobrazení nápovědy uživateli.

[6] Zdroj: screenshot aplikace IntelliJ IDEA

Filtrování seznamu nápovědy podle kontextu

Při hledání kontextu analyzujeme část kódu od začátku do pozice kurzoru nebo bezprostřední okolí pozice kurzoru. Cílem je získat seznam symbolů (typů tokenů), které se mohou vyskytovat na dané pozici.

Nejjednodušším způsobem je udělat analýzu textu (celého nebo po kurzor podle použité strategie) ANTLR parserem. Provedením syntaktické analýzy získáme ATN strukturu (viz výše), kterou je možné procházet. Text můžeme buďto parsovat celý, a poté najít místo kde se nachází kurzor, nebo analyzovat jenom část od začátku do pozice kurzoru, což je vhodnější.

Předpokladem je neúplnost analyzovaného kódu, při dosažení místa kurzoru (konce vstupu) parser vyhodí výjimku, kterou zachytíme, a informace použijeme ke konstrukci seznamu možných následujících symbolů, tj. z hlediska gramatiky `FOLLOW` množinu. Pokud je text dotazu syntakticky správný (např. kurzor je umístěn za kompletním prologuem - částí dotazu s definicí prefixů), musíme tuto situaci ošetřit zvlášť.

Nevýhodou tohoto přístupu je, že se můžou v kódu vyskytovat chyby a nápověda se nemusí zobrazit správně. Aby bylo zajištěno, že zpracovávaná výjimka je ta správná, je nutné brát vždycky poslední hlášení v seznamu chyb.

Získaný seznam symbolů je použit pro filtrování obsahu nápovědy. Dále lze výběr upřesnit využitím sémantických informacích o tokenech, např. filtrováním tříd/vlastností na příslušných pozicích RDF trojice. Ukázka dotazu a množiny symbolů je v příkladu 6.7. Analyzovaná část je označena podtržením a '|' označuje pozici kurzoru.

```
Dotaz: SELECT | var1 var2 WHERE { }
Symboly: ['*', '(', VAR1, VAR2, FROM, DISCTINCT, REDUCED]
```

Příklad 6.7: Příklad kódu a seznamu symbolů

Vložení šablon vybraných konstrukcí SPARQL

Vkládání šablon vybraných konstrukcí představuje další ulehčení pro uživatele. Šablony mohou být nabízeny formou menu, kontextového menu, tlačítek na panelu nástrojů, jako součást kontextové nápovědy, případně jinou formou.

V příkladu 6.8 je ukázka výpisu šablon dotazu `SELECT` tak, jak budou zobrazeny v kontextové nápovědě.

```
SELECT
SELECT * WHERE { ?A ?B ?C }
SELECT ?A ?B ?C WHERE { ?A ?B ?C }
```

Příklad 6.8: Šablona dotazu `SELECT`

6.4.4 Zvýraznění syntaxe

Zvýraznění syntaxe (syntax highlighting, obr. 6.7) představuje rozlišení částí textu různou barvou/stylem. Jedná se o standardní součást každého lepšího editoru, která výrazně usnadňuje orientaci v kódu a nepřímo i vizuální kontrolu překlepů v názvech.

Aby bylo možné text v editoru obarvit, je nutné identifikovat skupiny, které se budou zvýrazňovat stejnou barvou/stylem (tj. klíčová slova, názvy funkcí, literály, . . .). Způsobů, jakými můžeme nalézt tyto skupiny, je několik.

```
String queryPartBeforeCaret = getQueryPartBeforeCaret(query);
ParseTree parsedTree = getParsedTree(queryPartBeforeCaret);

syntaxErrorList = ((SparqlValidationErrorListener) errorListener).getSyntaxErrorList();
if (queryPartBeforeCaret.isEmpty()) {
    return getQueryContextTokens(queryType, includeBaseToken: true);
}
```

Obrázek 6.7: Zvýrazňování syntaxe a výskytů slova pod kurzorem [7]

- Obarvování po slovech

  Nejjednodušší možností je procházet kód po slovech a u každého zvlášť určit do jaké skupiny ho zařadit a rovnou ho zvýraznit. Typ tokenu určíme regulárním výrazem nebo ANTLR lexerem.

- Obarvování po skupinách

  Text v editoru parsujeme regulárními výrazy (př. 6.5 str. 36) nebo ANTLR lexerem a seznam typů tokenů s jejich umístěním uložíme a text obarvíme najednou. Výkonnostně není mezi analýzou ANTLR lexerem a regulárními výrazy rozdíl, ale zpracování lexerem je jednodušší a lépe udržitelné (viz kapitola 6.4.2).

[7] Zdroj: screenshot aplikace IntelliJ IDEA

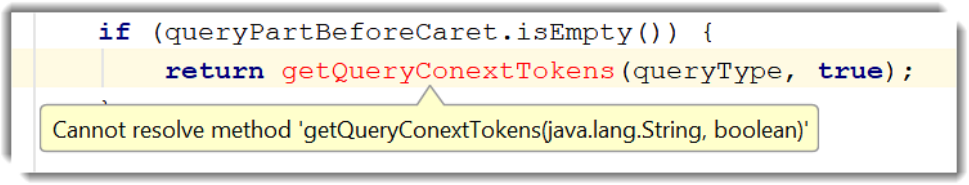
ANTLR lexerem budou identifikovány jednotlivé tokeny a jejich pozice, přirazena barva a nakonec tokeny obarveny.

Zvýraznění výskytů slova pod kurzorem může být užitečné např. při hledání proměnných se stejným názvem. Příklad zvýraznění je na obrázku 6.7. Identifikujeme slovo, které se nachází pod kurzorem, regulárním výrazem najdeme ostatní výskyty a slova zvýrazníme.

6.4.5 Zvýraznění chyb

Během zápisu kódu uživatelem jsou běžně prováděny různé kontroly, které pomáhají udržet kvalitu kódu. Chyby (error) / upozornění (warning) jsou hlášeny zvýrazněním nevalidního kódu a tooltipem s chybovým hlášením (obr. 6.8), značkou ve sloupci s číslem řádku, apod. Základní variantou je kontrola syntaxe, která bude prováděna i zde. Dále je možné provádět kontroly pomocí různých sémantických pravidel a dalších informací, které lze zjistit bez překladu kódu.

Kromě označování syntaktických chyb bude kontrolována (ne)existence použitých prefixů a zdrojů v uložišti a chybějící zdroj/prefix zobrazován ve formě upozornění.



Obrázek 6.8: Zvýraznění chyby a tooltip s popisem [8]

Validace kódu bude prováděna ANTLR parserem. Stejně jako v případě kontextové nápovědy budou během analýzy kódu zachytávány chyby. ANTLR parser se dovede z některých chyb zotavit (error recovery). Chyby je tedy možné ukládat, zpracovat (např. přizpůsobit chybové hlášení) a následně hromadně zobrazit. Protože v chybových hlášeních jsou defaultně zobrazeny názvy terminálních symbolů, je nutné hlášení přizpůsobit a názvy symbolů přepsat do formy čitelnější pro uživatele.

[8] Zdroj: screenshot aplikace IntelliJ IDEA

Alternativou by bylo kód validovat stejným způsobem, jako při vyhodnocování dotazu (vytvoření objektu dotazu funkcemi knihovny Apache Jena, která je použita pro implementaci). Tato možnost ale neumožňuje přizpůsobit chybové hlášení.

7 Implementace

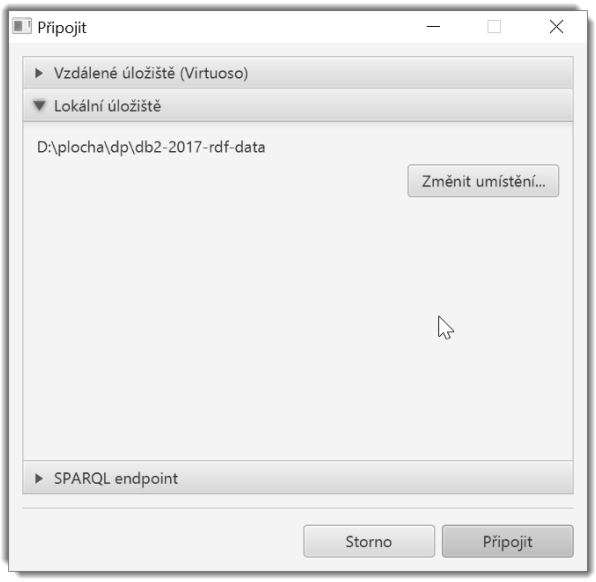
V následujících kapitolách bude popsána implementace navržených rozšíření.

7.1 Podpora SPARQL Endpointu

Implementace podpory dotazování SPARQL Endpointu zahrnuje modifikaci dialogu pro připojení k uložišti, změnu úložiště za běhu aplikace, vytvoření dialogu pro správu SPARQL Endpointů a úpravu vykonání dotazu.

7.1.1 Připojení k uložišti

Připojení k uložišti lze provést při startu aplikace pomocí dialogu s výběrem typu úložiště (obrázek 7.1). Tento dialog je nutné rozšířit o možnost výběru adresy SPARQL Endpointu.



Obrázek 7.1: Dialog pro připojení k uložišti

Před připojením je provedena kontrola pomocí vyhodnocení jednoduchého `SELECT` dotazu. Pokud je adresa z nějakého důvodu nevalidní nebo

nelze navázat spojení je uživatel informován dialogovým oknem. Poslední zadaná adresa se po připojení uloží do konfiguračního souboru aplikace a při novém zobrazení dialogu se zase načte.

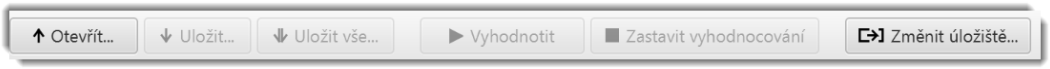
Seznam SPARQL Endpointů

Dialog obsahuje seznam SPARQL Endpointů, které lze spravovat v aplikaci (viz dále). V seznamu SPARQL Endpointů lze vyhledávat psaním do vstupního pole. Tato funkce je implementovaná pomocí knihovny ControlsFX[1].

Komunikaci mezi aplikací a úložištěm (odstínění aplikace od datové vrstvy) zajišťuje abstraktní třída `DataAgent` (viz [2]), resp. její potomci, které je nutné implementovat pro každý typ úložiště, tedy i pro SPARQL Endpoint - třída `SparqlEndpointDataAgent`. Tato třída, mimo jiné, obsahuje adresu úložiště, RDF model (zde je nedefinovaný - `null`), metody pro kontrolu dostupnosti úložiště, metody pro provedení dotazu, ... `DataAgent` je inicializován při připojení k uložišti.

7.1.2 Změna úložiště za běhu aplikace

Změna úložiště za běhu aplikace je realizována tlačítkem na nástrojové liště a položkou v menu, která vyvolá dialog pro připojení k uložišti.



Obrázek 7.2: Panel nástrojů - změna úložiště

Změnu úložiště, tedy změnu instance potomka `DataAgent`, je nutné propagovat do všech míst, kde je použit (např. v otevřeném dotazu). Ke sledování změn slouží rozhraní `Changeable`, které jednoduše umožňuje udržovat seznam posluchačů, kteří se mají informovat při změně na daném objektu. Při připojení k novému uložišti je nejprve odpojeno staré, nastaveno požadované a nakonec poslána notifikace o změně.

[1] http://fxexperience.com/controlsfx/

7.1.3 Vyhodnocení dotazu

Implementace provedení dotazu přes SPARQL Endpoint se téměř neliší od provedení v lokálním uložišti. Knihovna Jena poskytuje metody pro vzdálené i lokální vyhodnocení dotazu i jeho přerušení.

Dotaz je předpřipraven (př. 7.1) voláním metody `sparqlService()` ze třídy `QueryExecutionFactory`, která vrátí instanci třídy `QueryEngineHTTP` (implementující rozhraní `QueryExecution`).

```
queryExecution = QueryExecutionFactory.sparqlService(url, query,
    getHttpClient()) :
```

Příklad 7.1: Příprava dotazu pro vyhodnocení přes SPARQL Endpoint

První parametr představuje URL SPARQL Endpointu, prostřední objekt dotazu (instance třídy `Query`) a poslední parametr umožňuje specifikovat vlastní `Http Client` a zadat tak některé parametry pro HTTP request (např. timeout).

Následně je dotaz vyhodnocen (př. 7.2). Množina výsledků je získána stejně jako u lokálního úložiště. Tj. např. u `SELECT`u je vrácen iterátor a v cyklu jsou staženy jednotlivé řádky množiny výsledků.

```
Iterator<QuerySolution> iter = queryExecution.execSelect()
while (iter.hasNext()) {
     consumer.accept(iter.next());
}
```

Příklad 7.2: Vyhodnocení SELECT dotazu

Při vyhodnocení a sestavování dotazu je nutné rozlišit mezi aktualizačním dotazem (`update`; při připojení ke SPARQL Endpointu není podporováno) a výběrovým dotazem (`query`) a jednotlivými typy dotazu. Vyhodnocování je prováděno asynchronně [2].

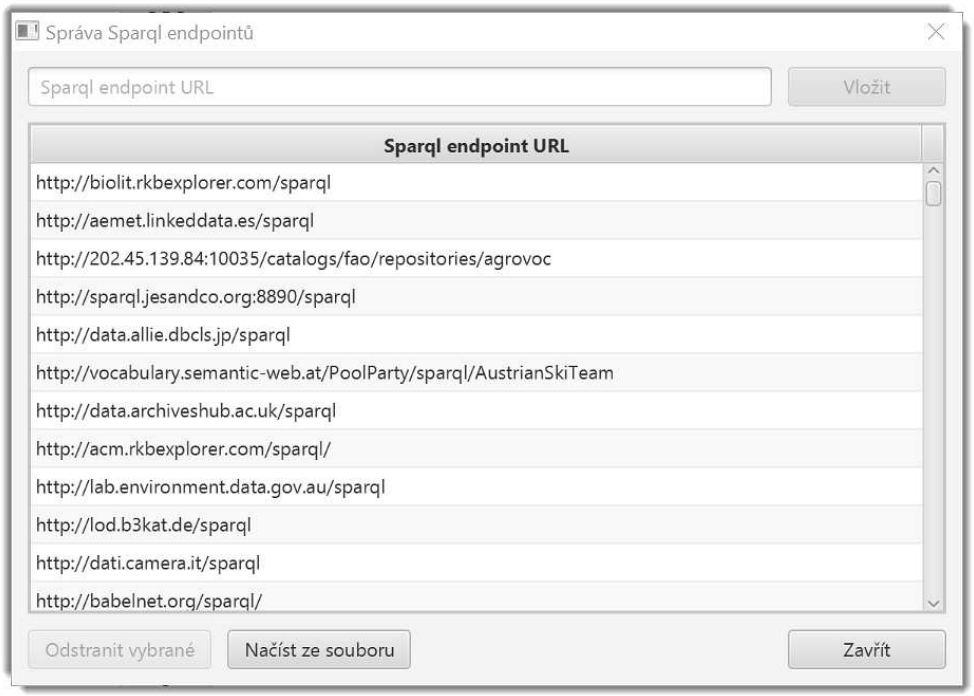
Přerušení vyhodnocení zde není podporováno, protože by byla problematická konzumace dosud nepřijatých dat.

7.1.4 Správa SPARQL Endpointů

Dialog pro správu SPARQL Endpointů (obrázek 7.3), dostupný z menu "Aplikace => Správa SPARQL Endpointů", umožňuje vložit URL endpointů prostřednictvím textového pole v horní části dialogu nebo hromadné načtení z textového souboru (každá adresa na jednom řádku).

Při vkládání je kontrolována funkčnost endpointu stejným způsobem jako při připojení. Při načítání ze souboru ale nejsou zobrazována chybová hlášení kvůli komfortu uživatele.

Při vypnutí aplikace jsou adresy SPARQL Endpointů serializovány do souboru `sparql_endpoints.dat` (stejně jako obsah správy zdrojů, ...) ve složce `.sparkle`, která se vytvoří v domovském adresáři uživatele při prvním použití aplikace.



Obrázek 7.3: Dialog pro správu SPARQL Endpointů

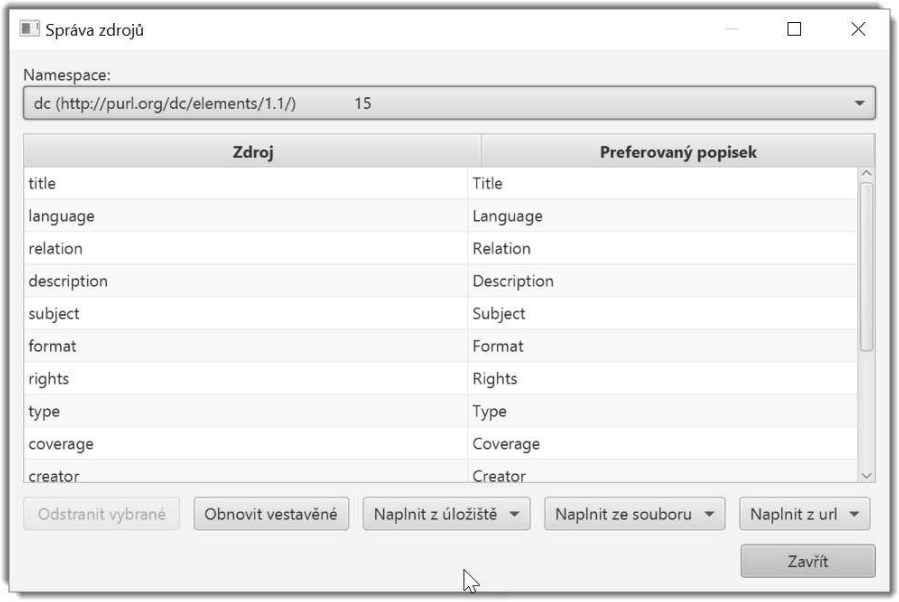
7.2 Zobrazování popisků

Implementace zobrazení popisků zahrnuje načtení popisků do úložiště, zobrazení popisků v kontextové nápovědě a nastavení preferovaných jazyků a typů popisků. Současně je nutné upravit úložiště zdrojů, prefixů a dialog pro jejich správu. Kromě zdrojů byly o popis rozšířeny i funkce a datové typy.

7.2.1 Načtení popisků do úložiště

Aplikace pro potřeby sestavení a vyhodnocení dotazů uchovává seznam prefixů, zdrojů, funkcí a datových typů.

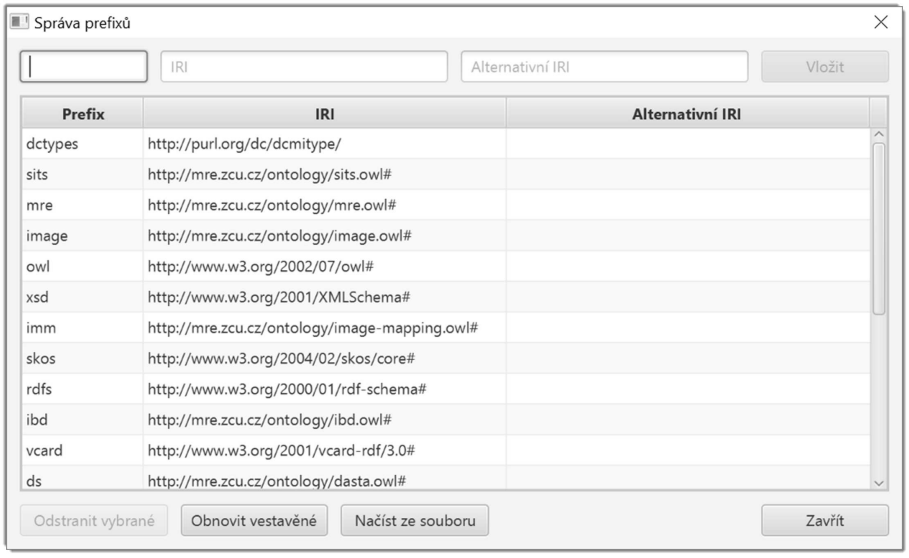
Úložiště zdrojů (třída `ResourcesStorage`, obr. 7.4) slouží k uchování zdrojů patřících do daného jmenného prostoru. Kromě lokálního názvu a jmenného prostoru (namespace) je u zdroje rozlišováno jestli se jedná o třídu nebo vlastnost. Informace o zdrojích potom můžeme použít např. v kontextové nápovědě.



Obrázek 7.4: Dialog pro správu zdrojů

Úložiště prefixů (třída `FunctionsStorage`, obr. 7.5) udržuje seznam prefixů a jmenných prostorů, které jsou použity při sestavení dotazu.

Abychom mohli s popisky efektivně manipulovat, je vhodné je do úložiště



Obrázek 7.5: Dialog pro správu prefixů

zdrojů předem načíst. Poté je můžeme použít kdekoliv v aplikaci.

Třída představující položku úložiště (`StorageEntry`) proto byla rozšířena o další atributy - popisek a alternativní IRI. Alternativní IRI slouží společně se jmenným prostorem ke stažení ontologie/slovníku. Je-li nastavena, načtou se zdroje do úložiště z ní, jinak se použije uvedený jmenný prostor. Tento přístup umožňuje lepší dostupnost zdrojů (namespace nemusí obsahovat stažitelný soubor nebo nemusí být jedna z adres funkční).

S touto změnou souvisí i úprava uložení dotazu v nativním formátu aplikace (`.sqf`) a jeho opětovného načtení, protože spolu s dotazem se ukládá i obsah úložiště zdrojů, prefixů, ... dotazu. Dále bylo vylepšeno rozlišování typu zdroje (třída/vlastnost) v uloženém dotazu, starší soubory proto již nebudou v nové verzi aplikace funkční (resp. nebude správně fungovat kontextová nápověda kvůli chybějící informaci o typu zdroje a popisku).

7.2.2 Zobrazení popisků

Popisky jsou zobrazovány v kontextové nápovědě, která je dostupná v editoru i ve formuláři. Obsah nápovědy je vykreslován jako `ListView` zapouzdřený v komponentě `javafx.stage.Popup` v editoru a `ScrollPane` ve formuláři, která umožňuje jeho zobrazení. Buňka nápovědy je rozdělena na tři části

(`Label`), aby šel obsah jednotlivých sloupců jednoduše formátovat a přizpůsobit šířku. První sloupec obsahuje typ hodnoty (např. “P” jako “Prefix”; ve formuláři je prázdný), druhý hodnotu (název zdroje, . . .) a poslední popisek hodnoty.

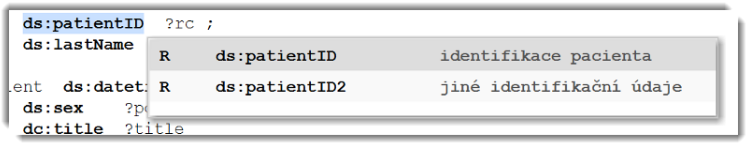
Obrázek 7.6: Kontextová nápověda se zapnutými a vypnutými popisky ve formuláři

Zobrazení ve formuláři

Ve formuláři (obr. 7.6) jsou titulky i data dostupná rovnou ze seznamu návrhů obsahu pro nápovědu. Při plnění jenom musíme rozlišovat, jestli jsou zdrojová data instance `StorageEntry` nebo řetězec, protože se načítá různý obsah podle typu vstupního pole. Pokud jsou instance `StorageEntry`, jednoduše vložíme do nápovědy hodnotu i titulek, jinak je titulek prázdný.

Zobrazení v editoru

V editoru (obr. 7.7) je zobrazení popisků složitější, protože nápověda je plněna (viz dále) ze slovníku (`ADT Trie`), který obsahuje jenom řetězce.

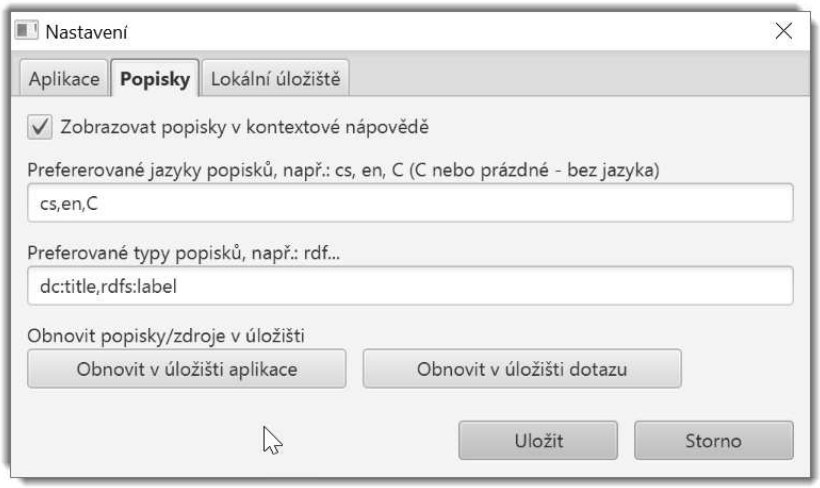


Obrázek 7.7: Kontextová nápověda s popisky v editoru

Postup je následující: V cyklu procházíme seznam návrhů na obsah nápovědy a testujeme typ řetězců (jméno zdroje, funkce, klíčové slovo, . . .). Jedná-li se o zdroj, separujeme jeho prefix a vyhledáme v uložišti seznam všech zdrojů, které k němu náleží. Seznam se vyhledá pouze jednou pro každý prefix, jinak by byl celý proces příliš pomalý. V seznamu zdrojů vyhledáme daný zdroj. Z nalezených dat již sestavíme položku nápovědy jako instanci třídy `ContextHelpEntry`.

7.2.3 Uživatelské nastavení

V dialogu Nastavení (obr. 7.8) můžeme (mimo jiného) vypnout/zapnout zobrazení titulků, nastavit preferovaný jazyk a typ titulků a hromadně nahrát zdroje (titulky) do úložiště aplikace nebo dotazu. Vypnutí/zapnutí zobrazení titulků je možné i z menu ("Zobrazit => Zobrazovat popisky v kontextové nápovědě"). Při ukončení dialogu jsou preference uloženy do souboru s konfigurací aplikace.



Obrázek 7.8: Dialog Nastavení - Popisky

Preferovaný jazyk a typ popisků

Popisky jsou nahrávány do úložiště s ohledem na preferovaný jazyk a typ popisků. Dialog Nastavení obsahuje textová pole pro zadání seznamu jazyků a popisků. Obě pole jsou validována. Nejprve je pomocí regulárního výrazu ověřena struktura (slova oddělená čárkami) seznamu. Jazyky jsou následně kontrolovány vůči seznamu jazyků.

Dvojmístné zkratky jazyků jsou porovnávány s `language tagem` (viz kapitola 2.1.2, např. `"Jméno ulice"@cs`) uvedeným u řetězce popisku, musí tedy splňovat normu ISO 639[2]. Pro zahrnutí popisků bez jazyka slouží znak "C". Seznam zkratek jazyků pro porovnání je načítán ze souboru.

U popisků je kontrolována syntaxe tak, že je na daném řetězci provedena lexikální analýza ANTLR lexerem a získaný typ tokenu je porovnán

[2] http://xml.coverpages.org/iso639a.html

s předpokládaným typem.

Aktualizace úložiště

Typ popisku a jazyk jsou zahrnuty do SPARQL dotazu, kterým jsou extrahovány zdroje do úložiště, ve formě filtru. Uložena je vždy první platná kombinace popisku a jazyka, která se v načítaných datech nachází.

V příkladu 7.3 je uveden dotaz pro načtení zdrojů z Dublin Core[3] ontologie s popisky `dc:title` v češtině.

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?r ?cs0
WHERE
{   ?r  a                   ?c
    FILTER strstarts(str(?r), "http://purl.org/dc/elements/1.1/")
    OPTIONAL
    { ?r  dc:title ?cs0
      FILTER ( lang(?cs0) = "cs" )
    }
}
```

Příklad 7.3: Načtení Dublin Core ontologie do úložiště zdrojů

7.3 Rozšíření textového editoru

Implementace rozšíření textového editoru zahrnuje přesun editoru na hlavní obrazovku, synchronizaci editoru a formuláře, implementaci kontextové nápovědy, obarvování syntaxe a vizualizaci chyb.

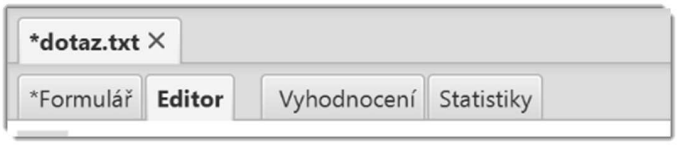
K implementaci textového editoru je použita knihovna RichTextFX[4]. RichTextFX poskytuje textovou oblast (text area) pro JavaFX. Editor umožňuje jednoduše obarvovat text pomocí CSS stylů, spravovat historii textových změn (CTRL+Z/CTRL+Y) apod.

[3] http://dublincore.org/
[4] https://github.com/FXMisc/RichTextFX

7.3.1 Synchronizace editoru a formuláře

V původní je editor k dispozici až po vyhodnocení dotazu a je propojený s formulářem pouze jednosměrně, takže je prakticky k ničemu.

Prvním krokem proto bylo sjednocení záložek s formulářem, editorem, výsledky a statistikami do jedné záložky. Na obrázku 7.9 je ukázka záhlaví záložky dotazu.



Obrázek 7.9: Hlavička záložky s dotazem

Dalším pak synchronizace mezi formulářem a editorem. Synchronizace je prováděna při detekci přepnutí z jedné záložky na jinou. V editoru i fomuláři jsou detekovány změny a při změně je prováděna validace dotazu, takže je možné zabránit vyhodnocení chybného dotazu (tlačítka pro vyhodnocení jsou blokována).

Fomulář → editor

Při přechodu mezi formulářem a editorem je dotaz převeden do textové formy, z řetězce je vytvořen objekt typu `Query` (`DataAgent.createQuery(query)` nebo `UpdateFactory.create(query)` podle typu dotazu) kvůli zformátování a vložen do editoru. Pokud je dotaz chybný a není možné ho přeložit, je zformátován jen nahrubo a uživatel je informován chybovým hlášením.

Editor → formulář

Při přechodu mezi editorem a libovolnou záložkou dojde nejprve k aktualizaci formuláře přepnutím na jeho záložku a zpět. Dotaz je do formuláře načten jako při načítání ze souboru (viz [3]), dojde tedy k obnovení panelu s formulářem a všech případných záložek se subdotazy. Aby se subdotazy korektně obnovily, jsou zavřeny a znovu vytvořeny. Pokud je dotaz chybný k synchronizaci nedojde (k přepnutí ano) a uživatel je informován.

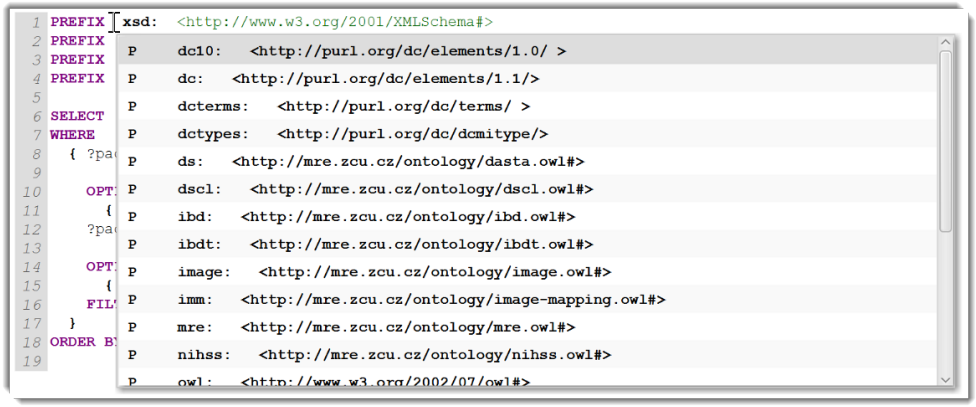
Během synchronizace je sestaven prologue z prefixů, které se nacházejí v kódu a jsou uloženy v uložišti prefixů. Není tedy nutné ho psát ručně.

Pokud se nějaký prefix v uložišti nenalézá, je při přepnutí zobrazeno chybové hlášení.

7.3.2 Kontextová nápověda

Kontextová nápověda (třída `ContextHelp`) je k dispozici v editoru i ve formuláři. Ve formuláři byl upraven její vzhled a mírně obsah kvůli použití popisků. V editoru bylo pouze jednoduché doplňování kódu podle prefixu, takže byla kompletně předělána. Na obrázku 7.10 je příklad kontextové nápovědy v editoru a na obrázku 7.6 (str. 50) ve formuláři.

Nápověda se zobrazí při psaní nebo stisku zkratky "CTRL+mezerník". Vzhled a konstrukce nápovědy byly stručně popsány v kapitole 7.2.2.



Obrázek 7.10: Kontextová nápověda - prologue

Sestavení obsahu nápovědy

Sestavení probíhá ve třech krocích:

- Naplnění slovníku.
- Vyhledání seznamu návrhů ve slovníku podle prefixu.
- Filtrování seznamu návrhů.

Naplnění slovníku

Základ tvoří slovník, který je implentován jako `ADT Trie`[5] (třída `TST`, viz kapitola 6.4.3).

Slovník je naplněn při vytvoření instance nápovědy klíčovými slovy a obsahem uložišť dotazu - jsou nahrány prefixy, zdroje, funkce, datové typy, klíčová slova, proměnné, operátory, závorky, příklad blank node a řetězce a vzory konstrukcí SPARQL.

Při detekci změny v uložištích (kromě úložiště prefixů, které sledování neumožňuje) a detekci změny obsahu editoru je seznam aktualizován. V příkladu 7.4 je kód posluchače pro zachycení změn obsahu editoru.

```
textEditor.richChanges()
  .filter(ch -> !ch.getInserted().equals(ch.getRemoved())).
     subscribe(change -> {
    // aktualizace napovedy, obarveni syntaxe, vizualizace chyb
   }
```

Příklad 7.4: Detekce změn obsahu editoru

Kromě popisků u zdrojů obsahuje nápověda i popis (prototypy) klíčových slov a funkcí. Klíčová slova, vestavěné (built-in) funkce, operátory, ... a jejich popis jsou načítány ze souboru `sparql_keywords.properties`.

Seznam návrhů

Při zobrazení nápovědy se ve slovníku vyhledají slova začínající daným prefixem (částí slova, která je před kurzorem) nebo se načte celý obsah (prefix je prázdný). Pokud je prefix prázdný, tak jsou pro zvýšení přehlednosti zahrnuty do nápovědy jenom prefixy a ne všechny zdroje, které k nim náleží.

Filtrování seznamu návrhů

Poslední krokem je filtrování seznamu návrhů (třída `ListFilter`) podle kontextu v místě kurzoru. Potřebujeme získat seznam symbolů (typů tokenů), které se z hlediska syntaxe mohou nacházet v místě kurzoru (tj. FOLLOW množinu[6]). Kromě syntaxe jsou řešena i další kritéria, např. umístění v rámci RDF trojice a typ dotazu (tj. `SELECT`, ...).

[5] http://algs4.cs.princeton.edu/52trie - Ternary search tries

[6] Množina terminálních symbolů, které mohou následovat za určitým neterminálním symbolem v dané větné formě.

Seznam symbolů získáme syntaktickou analýzou textu od začátku po pozici kurzoru. Postup získání symbolů je ukázán v příkladu 7.5.

```
String queryPartBeforeCaret = getQueryPartBeforeCaret(query);
ParseTree parsedTree = getParsedTree(queryPartBeforeCaret);
syntaxErrorList = ((SparqlValidationSyntaxErrorListener)
                        errorListener).getSyntaxErrorList();
List<Integer> tokens = getNextTokens(parsedTree, queryType);
```

Příklad 7.5: Analýza kontextu kurzoru

Nalezená množina symbolů je použita k filtrování seznamu návrhů. ANTLR lexerem je ke každému slovu ze seznamu návrhů určen typ a ten je porovnán ze seznamem symbolů.

Analýza kontextu kurzoru

Postup analýzy:

- Syntaktická analýza (ANTLR parserem) textu od začátku po kurzor (`parsedTree`).
- Zachycení chyb (`syntaxErrorList`) a získání seznamu symbolů (`tokens`).
- Zpracování výjimek průchodem parsovacího stromu a implementací metod z callback interface ANTLR parseru.

Syntaktická analýza

Prvním krokem je provedení syntaktické analýzy (př. 7.6) textu od začátku do pozice kurzoru.

```
lexer = getSparqlLexer(text, errorListener);
parser = getSparqlParser(errorListener, lexer);
parsedTree = ((SparqlParser) parser).query();
```

Příklad 7.6: Lexikální a syntaktická analýza textu

ANTLR parser byl již použit v aplikaci pro načítání dotazu ze souboru. Použitou gramatiku parseru bylo ale nutné otestovat pravidlo po pravidlu a řadu pravidel upravit (přeformulovat, rozdělit), protože získáná množina symbolů byla nepřesná.

Zachycení chyb

K parseru a lexeru je připojen vlastní posluchač chyb (`SparqlValidationErrorListener`). Posluchač implementuje metodu `syntaxError`, která jako parametry (mimo jiných) poskytuje `Recognizer rg`, tedy parser a výjimku `RecognitionException re`[7]. Ostatní parametry obsahují informace o chybě (pozici apod.).

Recognizer poskytuje ATN, informace o aktuálním stavu ATN (číslo stavu, apod.), kontext pravidla a další a výjimka, kromě již zmiňovaných a standardních informací, množinu terminálních symbolů, které mohou v daném stavu následovat (`re.getExpectedTokens()`)[8].

Poskytnuté informace (množina symbolů) jsou ve většině případech dostačující, některé situace je ale nutné řešit ručně. `syntaxErrorList` (seznam s informacemi o zachycených chybách) může obsahovat více záznamů (v části před kurzorem se může vyskytovat chyba/y), proto je nutné brát vždy poslední záznam, který by měl být teoreticky ten správný.

Zpracování výjimečných situací

Dotaz je zpracováván podle jednotlivých částí (viz kapitola 4.1) a přechody mezi částmi, kdy je text syntakticky správně, je nutné vyřešit samostatně. Některé další situace vyžadují úpravu seznamu získaného z chybového hlášení.

Většina výjimek je řešena procházením parsovacího stromu (`ParseTree`) pomocí `ParseTreeWalker` (př. 7.7). Ten umožňuje procházet parsovací strom a implementací konkrétních metod z callback interface ANTLR parseru řešit jenom dané situace.

```
ParseTreeWalker.DEFAULT.walk(new SparqlParserBaseListener() {
  @Override
  public void enterPrologue(SparqlParser.PrologueContext ctx) { }
}, parseTree);
```

Příklad 7.7: `ParseTreeWalker`

Příklad výjimek: začátek dotazu, přechod mezi prologuem a první klauzulí dotazu, přechod mezi `SELECT` klauzulí a `WHERE` klauzulí, začátek některých

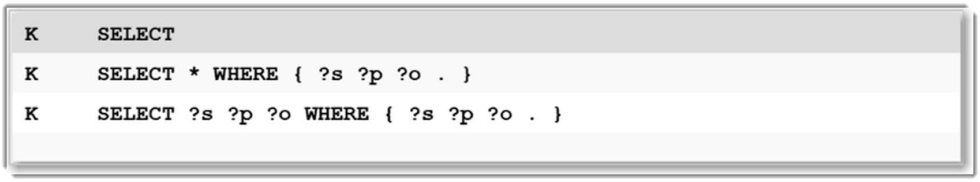
[7] Pozor, pokud dojde k zotavení z chyby může být výjimka `null`, je tedy vhodné ji testovat a použít informace poskytované ostatními parametry k sestavení náhrady.

[8] Množinu lze sestavit i z informací z recognizeru.

klauzulí (`WHERE`, ...) a další. Největší blok tvoří modifikátory řešení na konci dotazu.

Vložení šablony vybraných konstrukcí SPARQL

Do kontextové nápovědy jsou vloženy i kompletní šablony dotazů a částí dotazu včetně závorek. Šablony jsou filtrovány podle typu dotazu. Příklad kontextové nápovědy s šablonami dotazu `SELECT` je na obrázku 7.11.



Obrázek 7.11: Kontextová nápověda - šablona dotazu `SELECT`

7.3.3 Zvýraznění syntaxe a výskytu slova pod kurzorem

Zvýraznění syntaxe (obr. 7.12, třída `SyntaxHighlighter`) představuje rozlišení částí textu různou barvou/stylem.

Text dotazu je ANTLR lexerem rozdělen na tokeny. U každého tokenu známe pozici a typ. Následně jsou v cyklu procházeny jednotlivé tokeny a ke každému tokenu je přiřazen CSS styl podle informací získaných ze souboru `sparql_tokens_css_classes.properties`. Styl je uložen do kolekce stylů editoru `StyleSpansBuilder<Collection<String» spansBuilder` a nakonec je obnoven editor: `setStyleSpans(0, spansBuilder.create())`.

Zvýraznění je aktualizováno při každé změně v editoru.

Zvýraznění výskytu slova pod kurzorem

Zvýraznění výskytu slova pod kurzorem (obr. 7.12) je prováděno při každé detekci změny pozice kurzoru.

Podle pozice kurzoru identifikujeme slovo a sestavíme regulární výraz. Při zvýrazňování syntaxe je při každém cyklu regulární výraz vyhodnocen, a pokud je nalezena shoda, je styl vložen do kolekce.

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

SELECT  ?battle (GROUP_CONCAT(?label ; separator='; ') AS ?mlabel)
        (SUM(?casualty) AS ?totalCasualty) ?lat ?long
        (MAX(?date) AS ?mdate)
WHERE
  { ?battle  <http://purl.org/dc/terms/subject>
             <http://dbpedia.org/resource/Category:Battles_of_the_Thirty_Years'_War>
    OPTIONAL
      { ?battle  rdfs:label  ?label
        FILTER ( ( lang(?label) = "" ) || ( lang(?label) = "en" ) )
      }
    OPTIONAL
      { ?battle  <http://www.w3.org/2003/01/geo/wgs84_pos#lat>  ?lat }
    OPTIONAL
      { ?battle  <http://www.w3.org/2003/01/geo/wgs84_pos#long>  ?long }
  }
GROUP BY ?battle ?lat ?long
ORDER BY ?battle
```

Obrázek 7.12: Zvýrazňování syntaxe

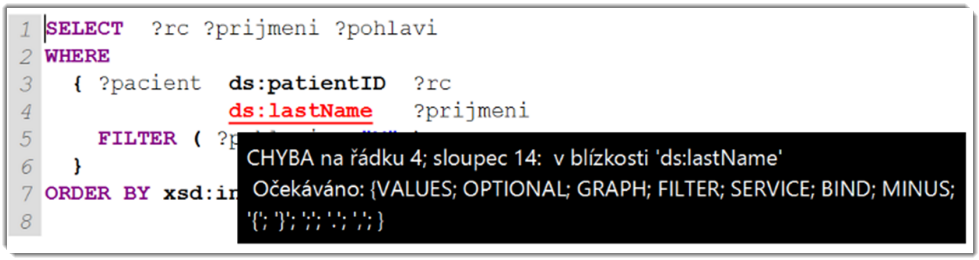
7.3.4 Vizualizace chyb

Validace dotazu za běhu je prováděna ve dvou fázích. Nejprve je parsován ANTLR parserem celý text. Výsledkem této fáze jsou pozice chyb. Potom je pro každou chybu parsován text od začátku do pozice chyby stejným způsobem jako při filtrování kontextové nápovědy. Získáme tak přesnější chybové hlášení.

Seznam očekávaných symbolů obsahuje názvy terminálních symbolů, symboly proto musí být převedeny do čitelnější podoby. Přepis symbolů je uložen v souboru `sparql_tokens_error_labels.properties`.

Samotné zvýraznění chyb je provedeno stejně jako v předchozích kapitolách. Kromě zvýraznění se při ukázání myší na chybu zobrazí tooltip s popisem (obr. 7.13).

Validace je opět prováděna při každé změně v editoru.



Obrázek 7.13: Zvýrazňování chyb

Validace chyb ve formuláři

Ve formuláři probíhala validace textových polí prostřednictvím regulárních výrazů. Protože nefungovala vždy správně a v některých případech byla dost pomalá, byla nahrazena validací ANTLR lexerem.

8 Testování

Nová rozšíření byla manuálně otestována způsobem popsaným níže. Bylo zjištěno pouze několik drobných problémů, které výrazně neomezovaly funkčnost a byly opraveny. Data pro testování jsou na CD (složka `testData`) a část také v přílohách.

Připojení ke SPARQL Endpoint

Funkčnost úložiště SPARQL Endpointů, načítání ze souboru a kontrola chybných adres byly ověřeny načtením skoro dvou set SPARQL Endpointů získaných z http://sparqles.ai.wu.ac.at/. Nad některými z nich byly provedeny dotazy.

Část seznamu SPARQL Endpointů je uvedena v příloze C a několik dotazů pro DBpedia a DBTune je uvedeno v příloze D. Zbytek seznamu se nachází na CD v adresáři `testData` v souboru `endpints.txt`. Dotazy jsou v adresáři `testData/dotazy` v nativním formátu aplikace (*.sqf*).

Zobrazení popisků

Bylo provedeno načtení zdrojů z jednadvaceti ontologií/slovníků a následně vyzkoušena kontextová nápověda v editoru i formuláři. Testována byla především rychlost sestavení nápovědy z pohledu uživatele (jenom vizuálně), bylo sledováno jestli se nápověda neseká u většího množství dat, tj. více než dva tisíce hodnot (např. zdroje z DBpedia). Dále bylo sledováno korektní zobrazení popisků výběrem a zobrazením v nápovědě několika tříd a vlastností z různých ontologií a načítání popisků do úložiště s ohledem na preferovaný jazyk a typ popisku.

Použité nastavení:
Typy popisků: dc:title, rdfs:label, skos:prefLabel
Jazyky: cs, en, C

Seznam ontologií/slovníků se nachází v příloze E a také na CD v adresáři `testData` v souboru `ontologies.csv` ve formě *csv* souboru, ze kterého je možné nahrát IRI do úložiště prefixů.

Rozšíření textového editoru

Během implementace byla kontextová nápověda postupně laděna podle gramatiky. Při testováni ji lze též využít jako pomůcku pro určení, co se má v které části dotazu vyskytovat. V příloze F se nachází `SELECT` dotaz určený k otestování editoru. Další dotazy jsou na CD v adresáři `testData/dotazy/editor`.

Postup: dotaz nahrajeme do editoru. Otevřeme kontextovou nápovědu na začátku dotazu a pak za každým slovem, případně znakem (např. operátorem). Dále uprostřed zvolených slov kvůli kontrole zobrazení nápovědy v rámci RDF trojice a dokončení prefixu. Aby byly vyčerpány všechny možnosti, je vhodné dotaz sestavit ve formuláři a poté jen zkoušet různé části.

V budoucnu by bylo vhodné postup automatizovat pomocí Unit testů. Vzhledem k délce gramatiky SPARQL (přibližně 140 pravidel) je ruční testování náročné. Automatizace testování by značně usnadnilo realizaci případných změn.

Stejným způsobem otestujeme všechny typy dotazů. Např. `WHERE` klauzule je u různých typů dotazů stejná, proto stačí vyzkoušet jen možnosti, které jsou pro daný typ unikátní.

Validaci chyb otestujeme zobrazením dotazu a simulací různých chyb, např. umazáváním částí názvů zdrojů, tečky, středníku apod. Pro validaci chyb i pro kontrolu obarvení syntaxe a zobrazení varování o neexistenci zdroje v uložišti můžeme použít dotazy v přílohách D, F nebo na CD v adresáři `testData/dotazy` a `testData/dotazy/editor`.

9 Diskuze

Kromě implementovaných rozšíření bylo opraveno několik nalezených chyb, např. obarvování záhlaví záložky a chyby ve formuláři. Dále byla ve formuláři upravena validace polí, která byla pro delší řetězce pomalá a způsobovala při zobrazování kontextové nápovědy zamrznutí aplikace i na několik sekund. Nyní se nápověda zobrazí okamžitě. U několika typů polí nefungovala validace zcela správně (např. se označoval chybně název zdroje). A nakonec byla provedena refaktorizace kódu, čímž se zvýšila přehlednost a umožnila snadnější rozšiřitelnost.

Textový editor a použití popisků

Aplikace nyní obsahuje vylepšený integrovaný textový editor, který poskytuje kontextovou nápovědu, vložení šablon konstrukcí SPARQL, obarvování syntaxe, zvýraznění výskytů slova pod kurzorem a validaci chyb při psaní. Editor a formulář jsou nově sjednoceny v jedné záložce a dochází ke vzájemné synchronizaci. Uvedená rozšíření zvyšují použitelnost editoru a komfort uživatele.

Kontextová nápověda a použití popisků zjednodušují tvorbu dotazu poskytnutím nápovědy ve formě klíčových slov, názvů zdrojů apod. Použití popisků dovoluje vyhledávat zdroje jenom se znalostí prefixu. Obarvování syntaxe a zvýrazňování chyb se zobrazením informací o chybě ve formě tooltipu zpřehledňuje dotaz a snižuje možnost výskytu překlepů a chyb.

Z nástrojů popsaných v kapitole 5 obsahuje kontextovou nápovědu jen Flint SPARQL Editor (kap. 5.2). Kontextové menu je ale u Flint SPARQL Editoru omezené a např. názvy zdrojů se zobrazují jen u predikátů.

Zvýrazňování syntaxe obsahuje Flint SPARQL Editor a YASGUI (kap. 5.3). V obou nástrojích je na stejné úrovni jako ve Sparkle.

iSPARQL (kap. 5.4) umožňuje vkládání některých konstrukcí SPARQL z menu a Flint SPARQL Editor dovoluje vkládat klíčová slova, zdroje apod. z tabulky. Způsob použitý u iSPARQL a Flint SPARQL Editoru sice dává větší přehled, ale výběr šablony z kontextové nápovědy je pro uživatele určitě pohodlnější.

Zvýrazňováním chyb disponuje Flint SPARQL Editor (kap. 5.2) a YAS-

GUI (kap. 5.3). Flint SPARQL Editor chyby jen zvýrazňuje, ale neposkytuje žádné dodatečné informace. YASGUI poskytuje informaci o chybě s výpisem návrhů pouze na úrovni celé řádky. Sparkle zobrazuje chybu v místě, kde byla zachycena i s popisem.

Popisky nepoužívá z vybraných SPARQL editorů žádný, ani zvýrazňování výskytů slova pod kurzorem.

Připojení ke SPARQL Endpointu

Připojení ke SPARQL Endpointu umožňuje vyhodnocovat dotazy nad dalšími uložišti bez nutnosti nahrávání objemných dat do lokálního úložiště a také možnost dotazovat a používat data i prostřednictvím existujících veřejně dostupných SPARQL Endpointů. Sparkle nyní obsahuje možnost připojení ke SPARQL Endpointu s výběrem ze seznamu endpointů a jejich správou.

Dotazování přes SPARQL Endpoint podporují všechny editory, i když některé omezeně. YASGUI a iSPARQL poskytují výběr ze seznamu. Flint SPARQL Editor a Gruff pouze textové pole pro zadání. Twinkle obsahuje tři pevně dané endpointy.

Nevýhodou dotazování přes SPARQL Endpoint může být výkon nebo nastavená omezení SPARQL Enpointu, který zavisí na internetovém připojení a výkonu endpointu, ke kterému může být připojena současně řada uživatelů.

Dále je v současnosti zakázáno přerušení vykonávání dotazu, a pokud stahujeme větší množství dat, může aplikace přestat reagovat. Vyhodnocování sice teoreticky přerušit lze (stejně jako u vyhodnocování lokálního), ale pokus o ukončení skončí výjimkou s informací, že dosud nebyla zpracována všechna data, a spojení bude skutečně přerušeno až po jejich přenosu. Jak tuto situaci řešit nevím.

Možná budoucí rozšíření

V dalším vývoji by bylo vhodné přidat panel nástrojů obsahující např. tlačítka pro historii (vpřed/zpět) a další, která jsou běžná v textových editorech. Dále možnost vyhledávání v editoru a synchronizaci úložiště prefixů v editoru a formuláři. Největším problémem je formulář, který není moc přehledný. Jeho předělání by, ale bylo náročné. Možnou náhradou by byla nějaká forma

grafického sestavování dotazu, které jsou součástí editorů YASGUI a Gruff.

Dále, pokud porovnáme aplikaci s uvedenými editory, můžeme uvažovat následující rozšíření:

- Grafické vytváření dotazu (viz editory Gruff, iSPARQL).
- Vylepšení zobrazení výsledků dotazu `SELECT` nebo obecně všech, např. grafické vykreslení výsledku dotazu (viz editor YASGUI, Gruff).
- Lepší správa lokálního úložiště - záloha obsahu na disk, seznam trojic nebo alespoň souborů s daty, které jsou v uložišti nahrány (viz Apache Jena - Fuseki)
- Doplnění podpory zbylých konstrukcí ze SPARQL 1.1. Např. `LOAD` a `CLEAR` (viz YASGUI).

10 Závěr

Přínos diplomové práce spočívá v nově implementovaných možnostech a také ve zvýšeni použitelnosti pro uživatele. Nové funkce mohou současné umožnit efektivnější práci v aplikaci Sparkle při tvorbě dotazu a práci s rozsáhlými daty.

Aplikace nyní obsahuje vylepšený integrovaný textový editor. Realizovaná rozšíření zvyšují použitelnost editoru a komfort uživatele. Kontextová nápověda a použití popisků zjednodušují tvorbu dotazu poskytnutím nápovědy ve formě klíčových slov, názvů zdrojů apod. Popisky v nápovědě dovolují vyhledávat zdroje jenom se znalostí prefixu. Obarvování syntaxe a zvýrazňování chyb se zobrazením informací o chybě ve formě tooltipu zpřehledňuje dotaz a snižuje možnost výskytu překlepů a chyb. Připojení ke SPARQL Endpointu umožňuje vyhodnocovat dotazy nad dalšími uložišti bez nutnosti nahrávání objemných dat do lokálního úložiště a také možnost použít data, ke kterým není jiný přístup.

První část tohoto textu se zabývala teoretickými aspekty, popisovala RDF, OWL a jazyk SPARQL. Dále byl popsán současný stav aplikace Sparkle, vypsán seznam nových rozšíření a popsány další editory pro manipulaci s RDF daty a pro dotazování ve SPARQL. V druhé části byla podrobně rozepsána analýza jednotlivých rozšíření, včetně popisu alternativ řešení. Ve třetí pak byla popsána implementace navržených rozšíření, tj. implementace dotazování na SPARQL Endpoint, použití popisků a rozšíření textového editoru. Dále bylo popsáno testování. A na závěr byly shrnuty výhody a nevýhody implementovaných rozšíření a navržena další s ohledem na popsané alternativní editory.

Seznam zkratek

ANTLR Another Tool for Language Recognition

ATN Augmented Transition Network

FOAF Friend of a friend

IRI International Resource Identificator

ISBN International Standard Book Number

MRE Medical Research and Education

OWL Web Ontology Language

RDBMS Relational database management system

RDF Resource Description Framework

RDFS RDF Schema Language

SKOS Simple Knowledge Organization System

SPARQL SPARQL Protocol and RDF Query Language

URI Uniform Resource Identificator

W3C World Wide Web Consortium

Seznam obrázků

Seznam tabulek

Seznam příkladů

Literatura

[1] Sparkle [online]. [cit. 2017-09-30]. Dostupné z: https://mre.zcu.cz/sparkle/index.html

[2] ŠMUCR, Jan. Grafická tvorba SPARQL. Plzeň, 2014. Diplomová práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd.

[3] KAZÁK, Josef. Sparkle - rozšíření nástroje pro tvorbu SPARQL dotazů. Plzeň, 2017. Diplomová práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd.

[4] KAZÁK, Josef. Rozšíření a úprava programu SPARKLE. Plzeň, 2017. Seminární práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd.

[5] SCREIBER, Guus a Yves RAIMOND, ed. RDF 1.1 Primer [online]. W3C Recommendation. 2004 [cit. 2017-11-03]. Dostupné z: https://www.w3.org/TR/rdf11-primer/

[6] RDF [online]. W3C [cit. 2017-09-30]. Dostupné z: https://www.w3.org/RDF/

[7] BRICKLEY, Dan a R. V. GUHA, ed. RDF Schema 1.1 [online]. W3C Recommendation. 2014 [cit. 2017-11-03]. Dostupné z: http://www.w3.org/TR/rdf-schema/

[8] CYGANIAK, Richard, David WOOD a Markus LANTHALER, ed. RDF 1.1 Concepts and Abstract Syntax [online]. W3C Recommendation. 2014 [cit. 2017-11-03]. Dostupné z: https://www.w3.org/TR/rdf11-concepts/

[9] RICKLEY, Dan a Libby MILLER. FOAF Vocabulary Specification 0.99 [online]. 2014 [cit. 2017-11-03]. Dostupné z: http://xmlns.com/foaf/spec/

[10] MILES, Alistair a Sean BECHHOFER. SKOS Simple Knowledge Organization System Reference [online]. W3C Recommendation. 2009 [cit. 2017-11-03]. Dostupné z: http://www.w3.org/2004/02/skos/core.html

[11] Dublin Core [online]. [cit. 2017-10-18]. Dostupné z: http://dublincore.org/

[12] CYGANIAK, Richard. Top 100 most popular RDF namespace prefixes [online]. [cit. 2017-10-18]. Dostupné z: http://richard.cyganiak.de/blog/2011/02/top-100-most-popular-rdf-namespace-prefixes/

[13] RDF Schema. Ontologies and Semantic Web. [online]. [cit. 2017-10-18]. Dostupné z: http://www.obitko.com/tutorials/ontologies-semantic-web/rdf-schema-rdfs.html

[14] RDF Schema 1.1 namespace [online]. W3C [cit. 2017-11-01]. Dostupné z: http://www.w3.org/2000/01/rdf-schema#

[15] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview [online]. W3C Recommendation . 2012 [cit. 2017-11-03]. Dostupné z: https://www.w3.org/TR/owl2-overview/

[16] HITZLER, Pascal, Markus KRÖTZSCH, Bijan PARSIA, Peter F. PATEL-SCHNEIDER a Sebastian RUDOLPH, ed. OWL 2 Web Ontology Language Primer. W3C Recommendation. 2012 [cit. 2017-11-03]. Dostupné z: https://www.w3.org/TR/owl2-primer/

[17] SMITH, Michael K., WELTY, Chris a Deborah L. MCGUINNESS. OWL Web Ontology Language Guide [online]. W3C Recommendation . 2004 [cit. 2017-11-03]. Dostupné z: https://www.w3.org/TR/owl-guide/

[18] MAŘÍK, Vladimír, Olga ŠTĚPÁNKOVÁ a Jiří LAŽANSKÝ. Umělá inteligence. Praha: Academia, 2013.

[19] PRUD'HOMMEAUX, Eric a Andy SEABORNNE. SPARQL Query Language for RDF [online]. W3C Recommendation. 2008 [cit. 2017-11-13]. Dostupné z: http://www.w3.org/TR/rdf-sparql-query/

[20] HARRIS, Steve a Andy SEABORNNE. SPARQL 1.1 Query Language [online]. W3C Recommendation . 2013 [cit. 2017-11-13]. Dostupné z: https://www.w3.org/TR/sparql11-query/

[21] FEIGENBAUM, Lee, CLARK, Kendall Grant a Elias TORRES. SPARQL Protocol [online]. W3C Recommendation . 2008 [cit. 2017-11-13]. Dostupné z: https://www.w3.org/TR/rdf-sparql-protocol/

[22] FEIGENBAUM, Lee, WILLIAMS, Gregory Todd, CLARK, Kendall Grant a Elias TORRES. SPARQL 1.1 Protocol [online]. W3C Recommendation . 2013 [cit. 2017-11-13]. Dostupné z: https://www.w3.org/TR/sparql11-protocol/

[23] GEARON, Paula, PASSANT, Alexandre a Axel POLLERS. SPARQL 1.1 Update [online]. W3C Recommendation. 2013 [cit. 2018-06-07]. Dostupné z: https://www.w3.org/TR/2013/REC-sparql11-update-20130321/

[24] BRIDGWATER, Adrian. Semantic Query Editor Flint Gains SPARQL 1.1 Support [online]. 2012 [cit. 2018-06-07]. Dostupné z: http://www.drdobbs.com/web-development/semantic-query-editor-flint-gains-sparql/240077536

[25] OAT Interactive SPARQL (iSPARQL) Query Builder [online]. [cit. 2018-06-07]. Dostupné z: http://wikis.openlinksw.com/dataspace/doc/owiki/wiki/OATWikiWeb/InteractiveSparqlQueryBuilder

[26] DODDS, Leigh. Twinkle: A SPARQL Query Tool [online]. [cit. 2018-06-07]. Dostupné z: http://www.ldodds.com/projects/twinkle/

[27] TOMASSETTI, Federico. The ANTLR Mega Tutorial [online]. 2017 [cit. 2018-06-07]. Dostupné z: https://tomassetti.me/antlr-mega-tutorial/

[28] HLAVÁČOVÁ, Klára. Zvýšení uživatelské přívětivosti, konfigurace nastavení a používání ontologií ve Sparkle. Plzeň, 2018. Seminární práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd.

[29] HLAVÁČOVÁ, Klára. Analytická nadstavba nad výsledky SPARQL dotazu (SELECT). Plzeň, 2017. Seminární práce. Západočeská univerzita v Plzni. Fakulta aplikovaných věd.

[30] Tutorial 4: Introducing RDFS OWL. LinkedDataTools.com [online]. [cit. 2018-06-23]. Dostupné z: http://www.linkeddatatools.com/introducing-rdfs-owl

[31] Online Access. DBpedia [online]. [cit. 2018-06-23]. Dostupné z: https://wiki.dbpedia.org/OnlineAccess

[32] Lee FEIGENBAUM. SPARQL By Example. W3C [online]. 2009 [cit. 2018-06-23]. Dostupné z: https://www.w3.org/2009/Talks/0615-qbe/

[33] KRESS, Jonas. SPARQL editor - code completion. In: Wikimedia Commons [online]. 2017 [cit. 2018-06-25]. Dostupné z: https://commons.wikimedia.org/wiki/File:SPARQL_editor_-_code_completion.png

A Přílohy na CD

Na přiloženém CD se nachází tento text, uživatelská dokumentace, zdrojové kódy aplikace a spustitelná verze aplikace.

- *src* - Složka se zdrojovými kódy
- *app* - Složka s aplikací.
- *testData* - Složka s testovacími daty.
- *dp.pdf* - Elektronická verze této práce.
- *doc.pdf* - Uživatelská dokumentace.

B Příklad OWL ontologie

Příklad upraven z [30].

```
@base <http://www.linkeddatatools.com> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix p: <plants#> .

# header
<> rdf:type  owl:Ontology ;
   rdf:label "Example Plant Ontology" .

# Object Properties
p:similarlyPopularTo rdf:type owl:ObjectProperty .

# Data properties
p:family rdf:type owl:DatatypeProperty .

# Classes
p:planttype rdf:type     owl:Class ;
            rdfs:comment "The class of all plant types." .

p:flowers rdf:type        owl:Class ;
          rdfs:subClassOf p:planttype ;
          rdfs:comment    "Flowering plants" .

#Individuals
p:magnolia rdf:type              p:flowers ;
           p:similarlyPopularTo  p:orchid ;
           p:family              "Magnoliaceae" ;
           rdf:label             "Magnolia" .

p:orchid rdf:type              p:flowers ;
         p:similarlyPopularTo  p:magnolia ;
         p:family              "Orchidaceae" ;
         rdfs:comment          Orchidej"@cz, "Orchid"@en .
```

Příklad B.1: Příklad OWL ontologie

C SPARQL Endpointy

Níže je uvedena část seznamu SPARQL enpointů pro testování. Zbytek seznamu se nachází na CD v adresáři `testData` v souboru `endpints.txt`.

`http://dbpedia.org/sparql`
`http://dbtune.org/jamendo/sparql/`

`http://biolit.rkbexplorer.com/sparql`
`http://aemet.linkeddata.es/sparql`
`http://202.45.139.84:10035/catalogs/fao/repositories/agrovoc`
`http://sparql.jesandco.org:8890/sparql`
`http://data.allie.dbcls.jp/sparql`
`http://vocabulary.semantic-web.at/PoolParty/sparql/AustrianSkiTeam`
`http://data.archiveshub.ac.uk/sparql`
`http://lab.environment.data.gov.au/sparql`
`http://lod.b3kat.de/sparql`

`http://dati.camera.it/sparql`
`http://babelnet.org/sparql/`
`https://www.ebi.ac.uk/rdf/services/biomodels/sparql`
`http://budapest.rkbexplorer.com/sparql/`
`http://opendata-bundestag.de/sparql`
`http://data.colinda.org/endpoint.php`
`http://crtm.linkeddata.es/sparql`
`https://www.ebi.ac.uk/rdf/services/chembl/sparql`
`http://citeseer.rkbexplorer.com/sparql/`
`http://cordis.rkbexplorer.com/sparql/`

`http://vocabulary.wolterskluwer.de/PoolParty/sparql/court`
`http://cultura.linkeddata.es/sparql`
`http://dblp.rkbexplorer.com/sparql/`
`http://dblp.l3s.de/d2r/sparql`
`http://dbtune.org/bbc/peel/sparql/`
`http://commons.dbpedia.org/sparql`
`http://wikidata.dbpedia.org/sparql`
`http://eu.dbpedia.org/sparql`
`http://rdf.disgenet.org/sparql/`

`http://italy.rkbexplorer.com/sparql`
`http://linkeddata.finki.ukim.mk/sparql`
`http://dutchshipsandsailors.nl/data/sparql/`
`http://semanticweb.cs.vu.nl/dss/sparql/`

D SPARQL Endpointy - dotazy

Dotazy jsou také v adresáři `testData/dotazy` na CD v nativním formátu aplikace (*.sqf*).

SPARQL Endpoint DBpedia: http://dbpedia.org/sparql
Zdroj příkladů: [31].

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbpediaRes: <http://dbpedia.org/resource/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

Příklad D.1: DBpedia - prologue

```
PREFIX goalkeeper:
  <http://dbpedia.org/resource/Goalkeeper_(association_football)>

SELECT DISTINCT ?soccerplayer ?countryOfBirth ?team ?countryOfTeam
    ?stadiumcapacity
WHERE
  { ?soccerplayer rdf:type dbo:SoccerPlayer .
    ?soccerplayer dbo:position|dbp:position goalkeeper: .
    ?soccerplayer dbo:birthPlace/(dbo:country)* ?countryOfBirth .
    ?soccerplayer
              dbo:team           ?team .
    ?team     dbo:capacity       ?stadiumcapacity ;
              dbo:ground         ?countryOfTeam .
    ?countryOfBirth
              rdf:type           dbo:Country ;
              dbo:populationTotal ?population .
    ?countryOfTeam
              rdf:type           dbo:Country
    FILTER ( ?countryOfTeam != ?countryOfBirth )
    FILTER ( ?stadiumcapacity > 30000 )
    FILTER ( ?population > 10000000 )
  }   ORDER BY ?soccerplayer
```

Příklad D.2: DBpedia: Fotbalisté (`soccerPlayersDBPedia.sqf`)

```
SELECT ?name ?birth ?description ?person WHERE {
   ?person rdf:type       dbo:MusicalArtist ;
           dbo:birthPlace dbpediaRes:Berlin ;
           dbo:birthDate  ?birth ;
           foaf:name      ?name ;
           rdfs:comment   ?description
    FILTER ( lang(?description) = "en" )
}  ORDER BY ?name
```

Příklad D.3: DBpedia: Hudebníci narození v Berlíně (`bornBerlinDBPedia.sqf`)

```
SELECT ?name ?birth ?death ?person
WHERE
  { ?person dbo:birthPlace dbpediaRes:Berlin ;
           dbo:birthDate  ?birth ;
           foaf:name      ?name ;
           dbo:deathDate  ?death
    FILTER ( ?birth < "1900-01-01"^^xsd:date )
  }   ORDER BY ?name
```

Příklad D.4: DBpedia: Lidé narození před rokem 1900 `born1900DBPedia.sqf`

SPARQL Endpoint DBTune: http://dbtune.org/jamendo/sparql/
Zdroj příkladů: [32].

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name ?img ?hp ?loc
WHERE {
  ?a a mo:MusicArtist ;
     foaf:name ?name .
  OPTIONAL { ?a foaf:img ?img }
  OPTIONAL { ?a foaf:homepage ?hp }
  OPTIONAL { ?a foaf:based_near ?loc }
}
```

Příklad D.5: DBTune Jamendo: Umělci (`artistJamendo.sqf`)

E Seznam ontologií/slovníků

V tabulce E.1 je uveden seznam ontologií/slovníků. Ontologie ze serveru mre.zcu.cz lze najít v [1]. Ostatní např. ve [12]. Seznam je také v adresáři `testData` v souboru `ontologies.csv` ve formátu vhodném pro načtení do aplikace.

Tabulka E.1: Prefixy a jmenné prostory ontologií/slovníků

Prefix	**IRI; alternativní IRI**
mre	http://mre.zcu.cz/ontology/mre.owl
ds	http://mre.zcu.cz/ontology/dasta.owl
dscl	http://mre.zcu.cz/ontology/dscl.owl
ibd	http://mre.zcu.cz/ontology/ibd.owl
ibdt	http://mre.zcu.cz/ontology/ibdt.owl
image	http://mre.zcu.cz/ontology/image.owl
imm	http://mre.zcu.cz/ontology/image-mapping.owl
nihss	http://mre.zcu.cz/ontology/nihss.owl
pop	http://mre.zcu.cz/ontology/pop.owl
sits	http://mre.zcu.cz/ontology/sits.owl
dctypes	http://purl.org/dc/dcmitype/
owl	http://www.w3.org/2002/07/owl
xsd	http://www.w3.org/2001/XMLSchema
skos	http://www.w3.org/2004/02/skos/core
rdfs	http://www.w3.org/2000/01/rdf-schema
vcard	http://www.w3.org/2001/vcard-rdf/3.0
dc10	http://purl.org/dc/elements/1.0/
rss	http://purl.org/rss/1.0/; http://web.resource.org/rss/1.0/schema.rdf
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns
dcterms	http://purl.org/dc/terms/
dc	http://purl.org/dc/elements/1.1/

F Dotaz pro testování editoru

Dotaz pro otestování editoru. Zdroj příkladu: [31]. Dotazy jsou také v adresáři `testData/dotazy/editor` na CD v nativním formátu aplikace (*.sqf*).

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX goalkeeper: <http://dbpedia.org/resource/Goalkeeper_(
   association_football)>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT (AVG(xsd:int(?stadiumcapacity)) AS ?avg) ?
   countryOfBirth
FROM <http://abc>
FROM NAMED <http://abc>
WHERE
  { ?soccerplayer rdf:type dbo:SoccerPlayer .
    ?soccerplayer dbo:position|dbp:position goalkeeper: .
    ?soccerplayer dbo:birthPlace/(dbo:country)* ?countryOfBirth .
    ?soccerplayer dbo:team          ?team .
    ?team    dbo:capacity       ?stadiumcapacity ;
             dbo:ground         ?countryOfTeam .
    ?countryOfBirth
             rdf:type           dbo:Country ;
             dbo:populationTotal ?population
    OPTIONAL
      { ?countryOfTeam a dbo:Country
      }
    FILTER ( ( ?countryOfTeam != ?countryOfBirth ) && ( ?
       countryOfTeam = "abc" ) )
    FILTER ( ?stadiumcapacity > 30000 )
    { SELECT *  WHERE { ?a ?b  ?c } }
  }
GROUP BY ?countryOfTeam ?countryOfBirth ?stadiumcapacity
HAVING ( ?countryOfTeam != ?countryOfBirth )
ORDER BY ASC(?soccerplayer) substr(?countryOfBirth, 3)
OFFSET 0
LIMIT  10
VALUES ( ?abc ?ddd ) { ( "123" true ) }
```

Příklad F.1: Dotaz pro otestování editoru - selectEditor.sqf