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

Dimplomová práce

Grafická tvorba SPARQL

Plzeň 2014 Jan Šmucr

Prohlášení

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

V Plzni dne 25. června 2014

Jan Šmucr

Abstract

The main goal of this diploma thesis is to create a visual tool helping its user with creation and evaluation of SPARQL queries. Reader will be acquainted with basics of related technologies and then with the whole process of the application development.

Obsah

Úvod

Hlavním cílem této práce je realizace grafického, uživatelsky přívětivého nástroje poskytujícího uživateli komfort při tvorbě a vyhodnocování dotazů v jazyce SPARQL.

Nejprve bude čtenář zasvěcen do základů technologií, které souvisí s tématem práce, obzvláště pak s metadatovým modelem RDF a jazykem SPARQL (kapitoly 2 a 4). Po teoretickém úvodu přijde na řadu náhled do současného světa aplikací s podobným zaměřením (kapitola 5.1) a identifikace výhod a nevýhod jejich způsobu řešení problému tvorby dotazů, ze kterých bude následně odvozeno další směřování vývoje a požadavky na aplikaci. Poté dokument provede čtenáře fázemi návrhu a implementace aplikace (kapitoly 6 a 7) a na závěr přijde shrnutí dosažených výsledků a nastínění dalších možností rozšíření aplikace.

1 Definice pojmů

1.1 Graf

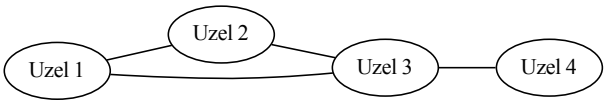
Grafem G je míněna dvojice $G = (V, E)$, kde V je konečná množina a $E \subset \binom{V}{2}$, kde

$$\binom{V}{2} = \{\{x, y\} : x, y \in V \text{ a } x \neq y\}$$

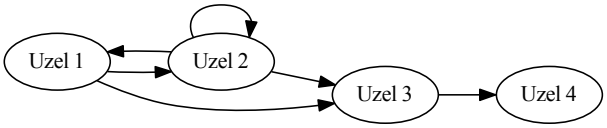
je množina všech dvouprvkových množin prvků množiny V. Prvky množiny V se nazývají *vrcholy* nebo *uzly* a prvkům množiny E říkáme *hrany* grafu G. Vrcholy $x, y \in V$ se nazývají sousední, pokud $\{x, y\} \in E$.[5]

Speciálním případem grafu je graf *orientovaný*, kde množinu $E \subset \binom{V}{2}$ netvoří dvouprvkové množiny ale uspořádané dvojice (viz obr. 1.2).[5]

$$\binom{V}{2} = \{(x, y) : x, y \in V\}$$



Obrázek 1.1: Ukázka neorientovaného grafu.



Obrázek 1.2: Ukázka orientovaného grafu.

1.2 Dalších pojmy

- **Formát** – Způsob organizace dat za účelem jejich uložení či zobrazení. [6]
- **IRI** – Internationalized resource identifier. Zobecnění URI umožňující použít v identifikátoru znaky ze sady Unicode.[9]
- **Ontologie** – Datový model explicitně definující pojmy a vztahy mezi nimi.[24]
- **Slovník** – Seznam pojmů a jejich definic.[7]
- **URI** – Uniform resource identifier. Strukturovaný řetězec sloužící k jednoznačné identifikaci zdroje.[4] Příkladem URI může být webová adresa.
- **XML** – Extensible markup language. Strukturovaný textový formát pro výměnu dat.[32]
- **Zdroj** – Naprosto cokoliv, co lze jednoznačně identifikovat pomocí URI. Například člověk, dokument na webu či jiný konkrétní objekt.[4]

2 Resource Description Framework

Resource Description Framework neboli *systém pro popis zdrojů* je obecný datový model pro reprezentaci informací na webu a organizace W3C jej definuje takto:

> *„RDF is a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed.*
>
> *RDF extends the linking structure of the Web to use URIs to name the relationship between things as well as the two ends of the link (this is usually referred to as a "triple"). Using this simple model, it allows structured and semi-structured data to be mixed, exposed, and shared across different applications.*
>
> *This linking structure forms a directed, labeled graph, where the edges represent the named link between two resources, represented by the graph nodes. This graph view is the easiest possible mental model for RDF and is often used in easy-to-understand visual explanations.“*[25]

Jinak řečeno, RDF je formát umožňující popisovat zdroje strukturovaným způsobem po trojicích *zdroj-vlastnost-hodnota*. Zdroji, který popisujeme, se v trojici říká *subjekt* (resp. *podmět*), jeho vlastnosti *predikát* a hodnotě této vlastnosti *objekt* (nebo *předmět*). Predikát je vždy zdroj definující nějakou vlastnost. Objektem může být zdroj nebo literál (prostá data). Ten může mít navíc stanoven datový typ a nebo – v případě, že jde o řetězec – jazyk.[26]

Trojici samotné se v RDF říká také *tvrzení* (angl. *statement*).

V orientovaném grafu je trojice tvořena výchozím uzlem (subjekt), hranou (predikát) a cílovým uzlem (objekt). Graf je tedy možné vyjádřit jako množinu trojic.

2.1 Příklad

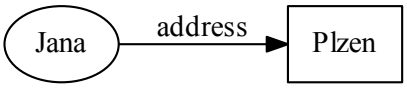
Způsob popisu dat pomocí RDF zkusím ukázat na příkladu využívajícím ontologie DASTA (viz kapitola 3.2). Mějme figurantku Janu a tvrzení (trojici):

Jana bydlí v Plzni.

Nebo lépe:

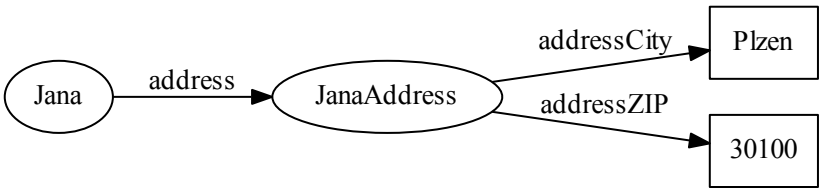
Jana má adresu Plzeň.

Subjektem je zde Jana, vlastností je její adresa a hodnotou vlastnosti je Plzeň. Tvrzení můžeme vyjádřit jako triviální graf na obrázku 2.1.



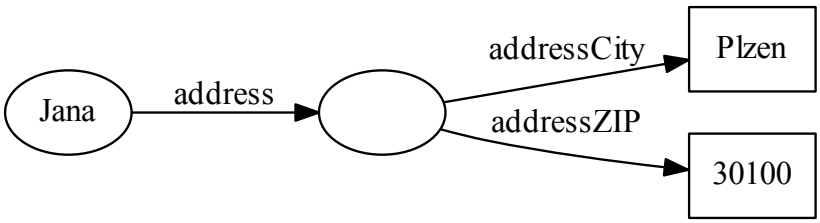
Obrázek 2.1: Jednoduché tvrzení v podobě grafu.

V popisu můžeme pokračovat přidáním dalších tvrzení. Mohli bychom se pokusit o upřesnění Janiny adresy doplněním poštovního směrovacího čísla. Pro zapouzdření informací o adrese vytvoříme nový zdroj, na který lze prostřednictvím jeho IRI odkázat i při jiné příležitosti. Graf bude vypadat jako na obrázku 2.2.



Obrázek 2.2: Rozšířené tvrzení v podobě grafu.

RDF navíc umožňuje tvorbu tzv. *anonymních* zdrojů (či uzlů), které nemají veřejně viditelné IRI a odkazovat na ně lze jen prostřednictvím jiných zdrojů (obr. 2.3). V RDF grafu se takový uzel nazývá *prázdný* (angl. *blank node*, *bnode*).[28]



Obrázek 2.3: Adresa Jany jako anonymní zdroj.

2.2 Zápis

Specifikace RDF[1] k zápisu využívá XML spolu s vlastními jmennými prostory.[29] Jmennými prostory tvořícími základ RDF jsou:

- **RDF**[2] - Základní pojmy pro tvorbu slovníků a ontologií.
- **RDF Schema**[3] **(RDFS)** - Navíc definuje pojmy jako *třída* nebo *datový typ*, zavádí dědičnost a umožňuje určit u vlastností jejich *doménu* (`rdfs:domain`) a *rozsah* (`rdfs:range`). Doména stanovuje, že zdroj s danou vlastností je instancí tříd uvedených v doméně. Jinak řečeno, má-li zdroj `R` vlastnost `P` a vlastnost `P` má v doméně třídy `C1` a `C2`, pak lze odvodit, že `R` je instancí tříd `C1` a `C2`.[27] Rozsah obdobným způsobem stanovuje, že hodnota dané vlastnosti je instancí tříd uvedených v rozsahu.[27]
- **Ontology Web Language**[4] **(OWL)** - Velkou měrou rozšiřuje schopnosti RDF a RDFS a lépe umožňuje popisovat data ve strojově čitelné formě.[24] Zavádí pojmy pro detailnější popis tříd a jejich vlastností.

Zkusme nyní informace z kapitoly 2.1 zapsat v jazyce RDF/XML. IRI používaných pojmů se zde skládá z části identifikující jmenný prostor a z části identifikující samotný pojem. V kódu 2.1 je vidět použití tzv. *prefixu*,

[1] Aktuální verze specifikace je k nalezení na adrese `http://www.w3.org/TR/rdf-syntax-grammar/`

[2] `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

[3] `http://www.w3.org/2000/01/rdf-schema#`

[4] `http://www.w3.org/2002/07/owl#`

kterým lze IRI jmenného prostoru nahradit a IRI celého pojmu tak v případě potřeby zkrátit.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ds="http://mre.kiv.zcu.cz/ontology/dasta.owl#">
  <ds:Female rdf:about="http://foo.com/Jana">
    <ds:address>
      <ds:PermanentAddress>
        <ds:addressCity>Plzen</ds:addressCity>
        <ds:addressZIP>30100</ds:addressZIP>
      </ds:PermanentAddress>
    </ds:address>
  </ds:Female>
</rdf:RDF>
```

Kód 2.1: Ukázka RDF/XML (1).

Z kódu 2.1 je vidět, že jsme Janě přiřadili vlastní IRI a klasifikovali ji jako ženu (`ds:Female`[5]). Jako její adresu jsme uvedli anonymní instanci třídy `ds:PermanentAddress` s vlastnostmi stanovujícími město a PSČ bydliště.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:ds="http://mre.kiv.zcu.cz/ontology/dasta.owl#">
  <ds:Female rdf:about="http://foo.com/Jana">
    <ds:address rdf:resource="http://foo.com/JanaAddress" />
  </ds:Female>
  <ds:PermanentAddress rdf:about="http://foo.com/JanaAddress">
    <ds:addressCity>Plzen</ds:addressCity>
    <ds:addressZIP>30100</ds:addressZIP>
  </ds:PermanentAddress>
</rdf:RDF>
```

Kód 2.2: Ukázka RDF/XML (2).

Kód 2.2 představuje variantu, kdy přiřadíme vlastní IRI i adrese samotné. Způsobů zápisu stejné informace je v RDF/XML více.

[5]Resp. `http://mre.kiv.zcu.cz/ontology/dasta.owl#Female` nezkráceně.

2.2.1 Další formáty

RDF/XML není jediným formátem používaným pro zápis RDF. Za zmínku stojí formát Notation 3[6] (N3), ve kterém jsou v době psaní tohoto dokumentu (červen 2014) popsány i slovníky RDF, RDFS či OWL.

```
@prefix ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#> .

<http://foo.com/Jana> a ds:Female ;
    ds:address <http://example.com/JanaAddress> .

<http://foo.com/JanaAddress> a ds:PermanentAddress ;
    ds:addressCity "Plzen" ;
    ds:addressZIP 30100 .
```

Kód 2.3: Ukázka N3. (1)

Kód 2.3 ukazuje způsob zápisu informace v N3. Varianta s anonymním zdrojem popisujícím adresu by vypadala velice podobně, pouze bychom nahradili původní IRI libovolným názvem s prefixem `_`, tedy například `_:addr`. Totéž by bylo možné zapsat i bez použití identifikátoru prázdného uzlu – viz kód 2.4.

```
@prefix ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#> .

<http://foo.com/Jana> a ds:Female ;
    ds:address [
        ds:addressCity "Plzen" ;
        ds:addressZIP 30100 ] .
```

Kód 2.4: Ukázka N3. (2)

V souvislosti s Notation 3 je na místě zmínit ještě také formáty Turtle a N-Triples, které z N3 vycházejí, ale zejména druhý zmíněný je oproti N3 značně zjednodušený a jde v podstatě pouze o trojice vypsané po řádcích (v nezkrácené podobě – bez užití prefixů).[7][22]

[6] Specifikace zde: `http://www.w3.org/TeamSubmission/n3/`

[7] Příklady užití těchto a dalších formátů lze nalézt na `http://www.w3.org/TR/rdf11-testcases/`.

3 Projekt MRE

Projekt MRE (Medical Research and Education) je jedním ze zaměření výzkumné skupiny medicínských informačních systémů na katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni.

3.1 MRE Ontology

Ontologie popisující obecný koncept výzkumného informačního systému. Definuje výzkumné týmy, jejich členy a řešené projekty a vazbu na datové sady, úložiště, dostupné výpočetní prostředí a další. [21]

Jmenný prostor: `http://mre.kiv.zcu.cz/ontology/mre.owl#`

3.2 DASTA Ontology

Ontologie vycházející z formátu DASTA. DASTA je formát pro předávání dat mezi informačními systémy zdravotnických zařízení standardizovaný ministerstvem zdravotnictví České republiky.[8] Definuje základní pojmy konceptu popisu klinických událostí (cokoliv, co se týká pacienta).

Jmenný prostor: `http://mre.kiv.zcu.cz/ontology/dasta.owl#`
Číselník DASTA: `http://mre.kiv.zcu.cz/ontology/dscl.owl#`

3.3 DICOM Ontology

Ontologie sloužící v projektu MRE jako referenční informační model pro obrazová data. Definuje například pojmy označující pacienta či snímek z počítačové tomografie. Původní implementace ontologie vznikla jako DICOM Ontology Project na Stanford Center for Biomedical Informatics Research (BMIR), ale poslední verze byla zveřejněna v srpnu 2010.

Jmenný prostor: `http://mre.kiv.zcu.cz/ontology/dicom.owl#`

4 SPARQL

Následující kapitola si klade za cíl seznámit čtenáře se základy jazyka SPARQL a měla by být brána pouze jako lehký úvod popisující klíčové způsoby jeho použití.

SPARQL (vyslovováno jako anglické slovo *sparkle*) je strukturovaný dotazovací jazyk pro práci s RDF úložišti (běžně nazýváno také *triplestore*, neboli úložiště trojic). Hlavním případem užití SPARQL je hledání podgrafu, který odpovídá šabloně uvedené v dotazu.[30] Dotazovat se je možné čtyřmi různými způsoby a to pomocí příkazů SELECT, ASK, CONSTRUCT a DESCRIBE.

4.1 Příkaz SELECT

Mějme úložiště trojic obsahující informace z kapitoly 2.2. Kód 4.1 představuje dotaz, který bychom tomuto úložišti kladli, pokud bychom chtěli zjistit PSČ bydliště Jany.

```
1 PREFIX ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#>
2
3 SELECT ?zip
4 WHERE {
5     <http://foo.com/Jana> ds:address ?address .
6     ?address ds:addressZIP ?zip .
7 }
```

Kód 4.1: Ukázka SPARQL (SELECT).

Je zjevné, že šablonu hledaného podgrafu tvoříme stejně jako graf samotný, tedy opět specifikací trojic. Hledané či jen neznámé uzly grafu nahrazujeme v trojicích proměnnými a ty je pak možné zahrnout do výsledku. Výsledek může obsahovat buďto všechny proměnné použité v dotazu – pak píšeme za klíčové slovo `SELECT` hvězdičku (`*`) – nebo výslovným uvedením názvu proměnné v úvodu dotazu tak, jak je to i ve zmíněném příkladu 4.1.[30]

Výstupem příkazu SELECT je tabulka, kde se na každé řádce nachází jeden výsledek pro všechny dotazované proměnné.

zip
30100

Tabulka 4.1: Výsledek příkazu 4.1.

4.2 Příkaz ASK

Příkaz ASK se zápisem velice podobá příkazu SELECT, ale jeho výsledkem je pouze informace o tom, zda hledaný podgraf v grafu existuje či nikoliv.[30] Kód 4.2 ukazuje, jak bychom se dotázali úložiště, zda se v něm nachází popis Jany, resp. zda úložiště obsahuje instanci třídy `ds:Female` s uvedeným IRI. V dotazu použité klíčové slovo `a` je zkratkou pro vlastnost `rdf:type` přiřazující zdroji jeho typ.[30]

```
1 PREFIX ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#>
2
3 ASK
4 WHERE {
5     <http://foo.com/Jana> a ds:Female .
6 }
```

Kód 4.2: Ukázka SPARQL (ASK).

4.3 Příkaz CONSTRUCT

Další příkaz je opět variantou na příkaz SELECT. Každý z výsledků je ale předzpracován a dosazen do šablony podgrafu, kterou uživatel v dotazu uvede. Výstupem je tak nový graf vzniklý transformací výsledků.[30]

Pro potřeby demonstrace užití příkazu CONSTRUCT mějme sadu informací o příbuzenských vztazích (kód 4.3).

Nyní zkusíme přetransformovat uvedená data v sadu trojic přiřazující osobám jejich prarodiče (kód 4.4). Výsledek ukazuje kód 4.5.

```
@prefix : <http://example.com/myfamily#> .
@prefix ft: <http://www.co-ode.org/roberts/family-tree.owl#> .

:Josef a ft:Person .
:Barbora a ft:Person .
:Marie a ft:Person .
:Vladimir a ft:Person .

:Zdenek a ft:Person ;
        ft:isChildOf :Barbora ; ft:isChildOf :Josef .

:Jana a ft:Person ;
        ft:isChildOf :Marie ; ft:isChildOf :Vladimir .

:Jan a ft:Person ;
        ft:isChildOf :Zdenek ; ft:isChildOf :Jana .

:Petr a ft:Person ;
        ft:isChildOf :Zdenek ; ft:isChildOf :Jana .
```

Kód 4.3: Data pro příklad 4.4.

4.4 Příkaz DESCRIBE

Posledním z příkazů, kterými SPARQL disponuje, je DESCRIBE. Ten slouží k získání všech informací, které může služba zpracovávající dotazy o nějakém konkrétním zdroji poskytnout. Výsledkem je – stejně jako v případě CONSTRUCT – podgraf vyhovující předepsané šabloně.[30]

Kód 4.6 ukazuje způsob, jak získat z úložiště kompletní popis všech v něm uložených pacientů.

Parametrem DESCRIBE nemusí být pouze proměnná, ale je možné nechat si zobrazit i informace o konkrétním zdroji, máme-li jeho identifikátor.[30]

```
PREFIX ft: <http://www.co-ode.org/roberts/family-tree.owl#>

CONSTRUCT {
    ?person ft:hasGrandParent ?grandParent .
}
WHERE {
    ?person ft:isChildOf ?parent .
    ?parent ft:isChildOf ?grandParent .
}
```

Kód 4.4: Ukázka SPARQL (CONSTRUCT).

```
@prefix ft: <http://www.co-ode.org/roberts/family-tree.owl#>.

<http://example.com/myfamily#Jan>
        ft:hasGrandParent :Josef ;
        ft:hasGrandParent :Barbora ;
        ft:hasGrandParent :Marie ;
        ft:hasGrandParent :Vladimir .

<http://example.com/myfamily#Petr>
        ft:hasGrandParent :Josef ;
        ft:hasGrandParent :Barbora ;
        ft:hasGrandParent :Marie ;
        ft:hasGrandParent :Vladimir .
```

Kód 4.5: Výsledek dotazu 4.4.

4.5 Filtrování výsledků

Množství podgrafů, kterým vyhovuje šablona v klauzuli `WHERE`, lze dále redukovat pomocí klíčového slova `FILTER`.[30] Parametrem filtru může být podmínka regulující, co vše může být hodnotami proměnných v nalezených podgrafech. Příklad použití filtru ukazuje dotaz 4.7, který se snaží o získání popisu všech pacientů žijících ve městě s názvem začínajícím na ‚J‘.

Funkce `strstarts` je jedna z vestavěných funkcí SPARQL 1.1 (verze 1.0 ji neobsahuje), které ve výrazech použít. Vestavěné funkce jsou výslovně definovány specifikací.[31] Pokud to úložiště podporuje, lze využít i dalších funkcí definovaných v jiných jmenných prostorech.

```
PREFIX ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#>

DESCRIBE ?patient
WHERE {
    ?patient a ds:Patient .
}
```

Kód 4.6: Ukázka SPARQL (DESCRIBE).

```
PREFIX ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#>

DESCRIBE ?patient
WHERE {
    ?patient ds:address [ ds:addressCity ?city ] .
    FILTER (strstarts(?city, "J"))
}
```

Kód 4.7: Ukázka SPARQL (FILTER).

4.6 Řazení a stránkování výsledků

Výsledky je možné řadit pomocí klauzule `ORDER BY` uvedené na konci dotazu. Parametrem je výraz, jehož výsledkem jsou hodnoty, které lze řadit. Lze tak dosadit například proměnnou, která bude ve výsledcích obsahovat čísla či řetězce nebo jiný výraz, který bude mít takový výstup.

Jako příklad poslouží kód 4.8 s dotazem požadujícím seřazení jmen pacientů podle délky jejich příjmení a následně podle křestního jména abecedně. Klíčové slovo `DESC` stanovuje, že budou výsledky řazeny od nejdelšího po nejkratší. Opačný směr by zajistilo použití klíčového slova `ASC`.[1]

Klíčová slova `LIMIT` a `OFFSET` slouží ke stránkování. Parametr prvního z nich určuje maximální počet vrácených výsledků a parametr druhého z nich určuje počet výsledků, které budou přeskočeny.

[1] Dle specifikace je tento způsob řazení výchozí a `ASC` tak není nutné uvádět.[30]

```
PREFIX ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#>

SELECT ?firstName ?lastName
WHERE {
    ?patient a ds:Patient ;
        ds:firstName ?firstName ;
        ds:lastName ?lastName .
}
ORDER BY DESC(strlen(?lastName)) ?firstName
LIMIT 5 OFFSET 3
```

Kód 4.8: Ukázka SPARQL (ORDER BY, LIMIT, OFFSET).

4.7 Skupiny trojic

Trojice uvnitř klauzule `WHERE` lze třídit do skupin a těm následně přiřazovat modifikátory třeba tak, jak to ukazuje kód 4.9.

```
PREFIX ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#>

SELECT ?firstName ?lastName ?city
WHERE {
    ?patient a ds:Patient ;
        ds:firstName ?firstName ;
        ds:lastName ?lastName .

    OPTIONAL {
        ?patient ds:address ?address .
        ?address ds:addressCity ?city .
    }
}
```

Kód 4.9: Ukázka SPARQL (OPTIONAL).

Výsledek dotazu 4.9 obsahuje další sloupec, kde nachází názvy měst, kde mají nalezení pacienti bydliště. Protože jsou ale poslední dvě trojice ve skupině s modifikátorem `OPTIONAL`, platným výsledkem je i pacient, která nemá bydliště vyplněné.[30]

Dalšími modifikátory jsou:

- `UNION` - Umožňuje sjednotit výsledky dvou nebo více šablon, což se hodí třeba v případě, že vyžadujeme, aby ve výsledcích figuroval sloupec „kontakt“ obsahující buďto telefonní čísla **nebo** e-mailové adresy pacientů a záleží pouze na tom, abychom na každého z nich nějaký kontakt získali.[30]
- `GRAPH` - Modifikátor, který si bere za parametr identifikátor grafu, ve kterém se bude hledat shoda se šablonou tvořenou skupinou trojic.[2][30]

4.8 Novinky ve SPARQL 1.1

V březnu 2013 vešla v platnost specifikace SPARQL verze 1.1. Ta přináší oproti původní verzi nespočet vylepšení rozšiřuje a standard o další oblasti týkající-se aplikace technologií spojených s RDF. Následující text se bude věnovat výhradně souhrnu vylepšení samotného jazyka SPARQL.

4.8.1 Agregační funkce

Prvním z vylepšení jsou agregační funkce, které již čtenář může znát z SQL. Ty umožňují transformaci výsledků do jiné formy ještě před jejich zasláním klientovi.[31] Následující ukázka klauzule `SELECT` se může nacházet na začátku dotazu, kterým se úložiště ptáme na jména a příjmení pacientů a chceme, aby výsledky spojilo do jednoho sloupce obsahujícího celá jména.

```
...
SELECT (CONCAT(?firstName, " ", ?lastName) AS ?fullName)
...
```

Lze použít i variantu s klíčovým slovem `BIND`:

```
{
    ?patient a ds:Patient ;
        ds:firstName ?firstName ;
        ds:lastName ?lastName .

    BIND (CONCAT(?firstName, " ", ?lastName) AS ?fullName)
}
```

[2]Pro více informací viz specifikace SPARQL.

4.8.2 Šablony jako filtr

Nově lze jako argument klíčového slova `FILTER` použít i šablonu podgrafu a to s modifikátory:

- `EXISTS { ... }` - Zajistí, že se v podgrafu vyhovujícím dotazu **bude** nacházet podgraf tvořený trojicemi ve skupině.
- `NOT EXISTS { ... }` - Zajistí, že se v podgrafu vyhovujícím dotazu **nebude** nacházet podgraf tvořený trojicemi ve skupině.

Alternativou k `NOT EXISTS` je užití nového modifikátoru skupiny `MINUS`, který ale způsobuje filtrování výsledků až na základě shody v datech.[3]

4.8.3 Řetězce vlastností

Novinkou je také možnost zkráceného zápisu cesty grafem. Kupříkladu zápis

```
{
    ?person foaf:knows ?friend .
    ?friend foaf:knows ?friendOfFriend .
}
```

můžeme zkrátit takto:

```
{
    ?person foaf:knows/foaf:knows ?friendOfFriend .
}
```

Způsobů alternativních zápisu cesty je velice mnoho a čtenáře tedy raději opět odkáži na specifikaci SPARQL 1.1, kde jsou ke všem zápisům ukázky použití.[31]

[3]Pro více informací viz specifikace SPARQL 1.1.

4.8.4 Filtrování dosazením známých hodnot

Další novinkou SPARQL 1.1 je možnost stanovit již v dotazu speciálním klíčovým slovem hodnoty, které mohou určité proměnné mít a výsledky, které toto nesplňují, tak z výstupu vyřadit.

```
{
    ?patient a ds:Patient ;
        ds:firstName ?firstName ;
        ds:lastName ?lastName .

    VALUES (?firstName ?lastName) {
        "Jan" "Novák"
        "Petr" UNDEF
    }
}
```

Kód 4.10: Ukázka užití VALUES.

Výsledek z příkladu 4.10 budou tvořit pouze pacienti jménem Jan Novák a osoby s křestním jménem Petr.

4.8.5 Seskupování výsledků a filtrování skupin

Další funkcionalitou známou z SQL je možnost seskupovat výsledky na základě stejných hodnot v jednom nebo více sloupcích užitím klauzule `GROUP BY`, jak ukazuje kód 4.11.[31]

```
PREFIX ds: <http://mre.kiv.zcu.cz/ontology/dasta.owl#>

SELECT ?firstName (COUNT(?lastName) AS ?count)
WHERE {
    ?patient a ds:Patient ;
        ds:firstName ?firstName ;
        ds:lastName ?lastName .
}
GROUP BY ?firstName
```

Kód 4.11: Ukázka užití GROUP BY.

Uvedený příklad nastiňuje, jak bychom mohli získat z úložiště statistiku četnosti jednotlivých křestních jmen. Přidáním následující řádky poté omezíme statistiku na jména začínající na ‚J‘:

```
HAVING (strstarts(?firstName, "J"))
```

4.8.6 Vnořené dotazy

SPARQL 1.1 umožňuje v rámci dotazu využívat i data, která jsou výsledkem jiného, vnořeného dotazu. Dovolím si uvést příklad takového dotazu přímo z [31] (kód 4.13).

```
@prefix : <http://people.example/> .

:alice :name  "Alice", "Alice Foo", "A. Foo" .
:bob   :name  "Bob", "Bob Bar", "B. Bar" .
:carol :name  "Carol", "Carol Baz", "C. Baz" .
:alice :knows :bob, :carol .
```

Kód 4.12: Data pro příklad 4.13 (převzato z [31]).

```
PREFIX : <http://people.example/>
SELECT ?y ?minName
WHERE {
  :alice :knows ?y .
  {
    SELECT ?y (MIN(?name) AS ?minName)
    WHERE {
      ?y :name ?name .
    } GROUP BY ?y
  }
}
```

Kód 4.13: Ukázka vnořeného dotazu (převzato z [31]).

Vnořený dotaz se provádí jako první a navenek poskytuje pouze proměnné uvedené v klauzuli `SELECT`. Výsledek dotazu ukazuje tabulka 4.2

y	minName
:bob	"B. Bar"
:carol	"C. Baz"

Tabulka 4.2: Výsledky dotazu 4.13 (převzato z [31]).

5 Analýza

5.1 Existující editory SPARQL

5.1.1 Gruff

Mocný nástroj pro správu úložišť založených na databázi AllegroGraph. Disponuje propracovaným grafickým i textovým SPARQL a Prolog editorem.

Poslední verze: 5.2.1 (2014)
Stav vývoje: Aktivní
K dispozici na: `http://franz.com/agraph/gruff/`

5.1.2 iSPARQL

Open source webový SPARQL editor s možností grafické i textové editace dotazů. Jeho primárním použitím je sloužit jako webové rozhraní úložiště. Používá jej například DBpedia.[1]

Poslední verze: 1.30 (březen 2014)
Stav vývoje: Aktivní
K dispozici na: `https://github.com/openlink/iSPARQL`

5.1.3 OWL2Query

Komplexní plugin pro editor ontologií Protégé[2] určený k vizuální tvorbě dotazů s možnou serializací do formátu SPARQL či SPARQL-DL.[3]

[1] `http://dbpedia.org/isparql`
[2] `http://protege.stanford.edu/`
[3] `http://www.derivo.de/en/resources/sparql-dl-api/`

Poslední verze: 0.3.1 (listopad 2013)
Stav vývoje: Neznámý
K dispozici na: `http://franz.com/agraph/gruff/`

5.1.4 Store Manager

Pokročilý textový editor s podporou SPARQL 1.1 Query i Update a několika druhů úložišť. Disponuje zvýrazněním syntaxe, tabulkovým zobrazením výsledků s možností exportu nebo třeba grafickou vizualizací grafu.

Poslední verze: 1.5 (červen 2014)
Stav vývoje: Aktivní
K dispozici na: `https://bitbucket.org/dotnetrdf/dotnetrdf`

5.1.5 Twinkle

Jednoduchý textový editor s textovým či tabulkovým zobrazením výsledků. Podporuje SPARQL 1.0 a uživateli nabízí sadu přednastavených prefixů a uložišť, ke kterým se může připojit. Další zdroje dat lze nastavit prostřednictvím konfiguračních souborů.

Poslední verze: 2.0 (prosinec 2007)
Stav vývoje: Neznámý
K dispozici na: `http://www.ldodds.com/projects/twinkle/`

5.2 Grafická editace SPARQL

Aplikace z kapitoly 5.1 ukazují směry, kterými se současné grafické nástroje pro tvorbu SPARQL ubírají.

- **Dotaz pouze v textové podobě** – První skupina aplikací využívá grafické rozhraní jen jako prostředí pro tvorbu dotazu a dotaz samotný je stále v podobě textu. K dispozici mohou být grafické prvky umožňující vložit na označené místo v textu předpřipravené části dotazu.

- **Kombinace grafického a textového přístupu** – Dotazy lze kromě ručního psaní navrhovat i vizuálně jako grafy s použitím myši. Části dotazu, které nelze vyjádřit graficky, jsou tvořeny jinak.

Výhody psaní dotazu v textové podobě jsou zřejmé – není narušena expresivnost jazyka a lze bez omezení využít všechny jeho možnosti. Nevýhodou je nepřístupnost takového způsobu člověku málo znalému syntaxe SPARQL a určitá míra nepohodlí spojená s ručním psaním kódu. Řešením může být zavedení kontextové nápovědy, která by uživateli radila s psaním dotazu v závislosti na tom, kde se v něm zrovna nachází.

V případě plně grafického provedení je problém neznalosti syntaxe SPARQL alespoň částečně vyřešen a uživatel se může k řešení svého problému pohodlně „doklikat“. Podmínkou je ale to, aby vývojář aplikace dobře odladil design aplikace a zachoval přitom podporu co největšího množství schopností SPARQL.

5.2.1 Editace dotazu v podobě formuláře

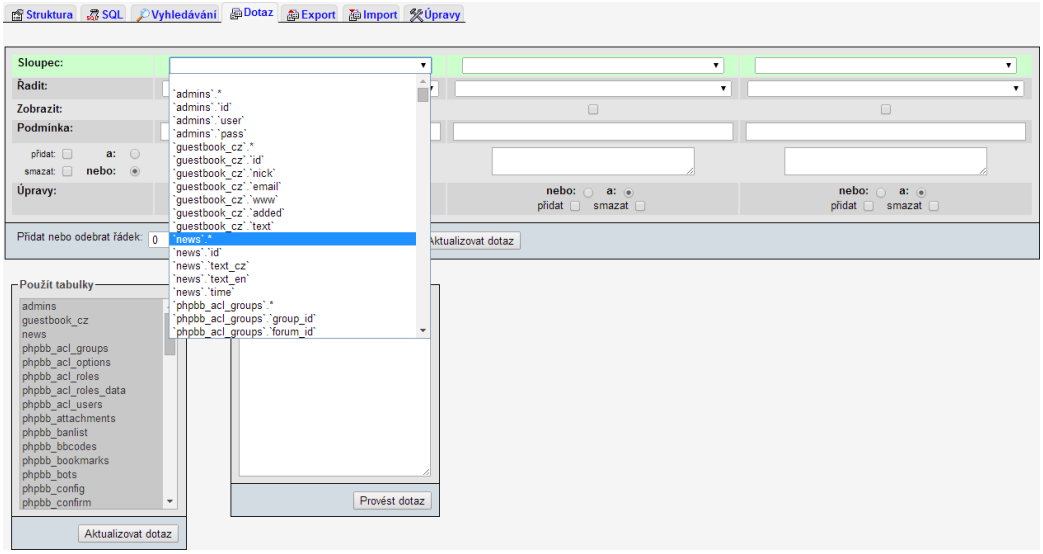
Přístup, který se v 5.1 neobjevil, je založený na metodě, kdy uživatel pouze vyplňuje kolonky formuláře. Ten aplikace před odesláním dotazu zpracuje a sestaví z jeho obsahu dotaz, který následně odešle ke zpracování do databáze.

S takovým přístupem se lze setkat u nástrojů pracujících s relačními databázemi. Příkladem může být webové rozhraní aplikace phpMyAdmin[16] (obr. 5.1).

Toto řešení si bere výhody a nevýhody z obou přístupů z kapitoly 5.2. Z velké části odpadá nutnost znát syntaxi jazyka, políčka formuláře mohou být inteligentní a obsahovat přednastavené hodnoty. Editaci lze částečně kontrolovat myší.

Nevýhodou může být horší rozšiřitelnost a těžkopádnost při implementaci některých vyjadřovacích schopností (jako příklad lze uvést vnořené dotazy) a stále je potřeba mít na paměti i uživatelskou přívětivost, aby množství viditelných ovládacích prvků nepřekročilo únosnou mez.

Tento způsob vidím jako cestu, kterou by se mělo řešení grafické editace SPARQL v této práci ubírat. Formulář nemusí být statický – prvky lze libovolně přidávat i odebírat – a zanoření skupin trojic v dotazu je možné řešit



Obrázek 5.1: Okno pro tvorbu SQL dotazů v phpMyAdmin.

kontejnerovými komponentami.

5.3 Schopnosti a charakteristika aplikace

Se znalostí způsobu editace dotazu v připravované aplikaci je možné pokračovat rozborem problémů s funkcionalitou, kterou by aplikace řešící daný problém měla mít.

5.3.1 Druh výsledné aplikace

Zde je na výběr mezi aplikací webovou, tedy takovou, která by běžela na vzdáleném serveru a uživatel by se k ní připojoval prostřednictvím webového prohlížeče a aplikací klientskou, která by fungovala přímo na zařízení uživatele.

Webová aplikace má výhody v možnosti přístupu odkudkoliv a možnosti ji využívat na prakticky jakémkoliv zařízení s přístupem k Internetu. Nevýhodou jsou ale vysoké nároky na server v případě, že by měla na něm provozovaná aplikace podporovat nejen připojení ke vzdáleným, ale také k

lokálnímu, testovacímu úložišti a měla v jednu chvíli obsluhovat požadavky několika uživatelů najednou.

Další nevýhodou webové aplikace je její obtížnější ladění a uvedení do provozu.

Uvedené problémy v případě klientské aplikace odpadají. Jako kompromis z hlediska možnosti využívat aplikaci všude, se nabízí varianta její realizace v podobě, kterou lze provozovat jako applet – tedy jako drobnou aplikaci běžící v prostředí jiné aplikace, nejčastěji webového prohlížeče.[15]

5.3.2 Tvorba a editace dotazů

Aplikace by měla fungovat jako klasický editor a tento fakt v podstatě definuje sadu základních schopnosti, kterými by měla aplikace disponovat, tedy vytvářet nové a umožňovat úpravu existujících souborů. Samozřejmostí je možnost mít otevřeno více souborů najednou.

Dá se předpokládat, že bude nutnost zavést vlastní formát souboru s dotazem, což přináší požadavek na další funkcionalitu – export dotazu ve formátu využitelném třetí stranou, tedy pravděpodobně přímo ve formě SPARQL.

Nutnost zavedení vlastního formátu je mimo jiné dána tím, že obsah souborů produkovaných aplikací nemusí nutně korelovat s aktuálním standardem SPARQL. Aplikace tak dostane možnost přidat do souborů s dotazem i informace, které nejsou přímo jeho součástí. Zároveň pak nebude nutné řešit problémy s načítání dotazů obsahujících konstrukce, které aplikace nepodporuje.[4]

5.3.3 Pomoc při editaci

Psaní SPARQL dotazů od uživatele obecně vyžaduje znalost jejich struktury a klíčových slov. Aplikace by se měla pokusit uživatele od takové potřeby co nejvíce distancovat. Nabízí se tedy možnost, že by uživatel pouze doplňoval text do předpřipravené šablony či formuláře. Formulář se přitom může skládat ze znovuvyužitelných prvků – kupříkladu klauzule `WHERE` je pro všechny druhy dotazu stejná.[30] Taková část by pak mohla existovat jako samostatná

[4]Další informace o podporované podmnožině SPARQL lze nalézt v kapitole 5.3.5.

komponenta, která by na vyzvání vracela svůj obsah v různých podobách – ať už při ukládání souboru či sestavování dotazu za účelem exportu nebo vyhodnocení.

Další požadavek na tvůrce dotazů je schopnost jejich orientace v používaných slovnících. Ruční psaní dotazu vyžaduje znalost IRI jmenných prostorů a zdrojů. Aplikace by mohla být schopná tyto informace uchovávat stylem *set and forget*. Uživateli by stačilo informace o slovnících nastavit pouze jednou a poté už by je jen používal. Inspirací mohou být dnešní vývojová prostředí, která nenutí uživatele pamatovat si zpaměti názvy balíků, tříd a metod.

5.3.4 Vyhodnocování dotazů

Možnost napsaný dotaz přímo vyhodnotit je stěžejní pro dobrou použitelnost aplikace. Z toho důvodu by měla aplikace poskytovat oddělený mód, kde bude vyhodnocování probíhat a kde se uživatel dozví výsledek (inspirací může být například aplikace Oracle SQL Developer[14]). Samozřejmostí by měla být alespoň základní schopnost výsledek uložit do vhodného formátu.

S tímto souvisí požadavek na napojení k úložišti, kterému by bylo možné dotazy klást. Ze zadání vyplývá, že je důležité, aby aplikace obsahovala možnost připojení k uložišti projektu MRE fungujícím nad databází Oracle.

K dispozici by měla být možnost připojit se i k úložišti čistě testovacímu (tedy např. lokálnímu).

5.3.5 Implementovaná podmnožina SPARQL

Požadavkem ze strany vedoucího práce je, aby aplikace dokázala generovat SPARQL dotazy běžně používané pro práci s ontologiemi projektu MRE, k čemuž poskytl ukázkovou sadu 31 takových dotazů.

Všechny poskytnuté dotazy jsou typu SELECT. Z nich 27 využívá řazení výsledků, tedy klauzuli `ORDER BY`. V té se pokaždé vyskytuje upřesnění směru řazení výsledků (`ASC`, `DESC`) a v osmi případech také přetypování (resp. složitější výraz než prosté uvedení proměnné, podle které se řadí). Devatenáct dotazů používá modifikátor `OPTIONAL` a jeden dotaz používá klauzuli `LIMIT`. Tři z dotazů používají agregační funkce společně s klauzulí `GROUP BY`

ze SPARQL 1.1.

Z předchozího odstavce vyplývá, že prioritou bude implementace podpory SPARQL 1.0 s důrazem kladeným na možnost pozdějšího rozšíření o alespoň základní podmnožinu schopností SPARQL 1.1 v čele s agregačními funkcemi.

6 Návrh

6.1 Požadavky na vývojové nástroje

Ze zadání vyplývá, že musí být aplikace vytvořena v jazyce Java.

6.1.1 Knihovna pro práci s RDF a SPARQL

Z požadavku na podporu uložiště projektu MRE vyplývají dvě možnosti, co se volby knihovny pro pro práci s RDF a SPARQL týče.

První z nich je Sesame[18], open-source řešení, které má dle mých dosavadních zkušeností vše, co bude aplikace ke svému fungování potřebovat, včetně adaptéru pro připojení k Oracle databázi.[13] Nevýhodou je ovšem málo aktivní komunita okolo projektu a neznámá kompatibilita s úložištěm projektu MRE.

Druhou možností, pro kterou jsem se nakonec rozhodl, je knihovna Apache Jena[1] a to právě z důvodu, že kompatibilita konkrétní verze knihovny s úložištěm MRE je známa a není důvod při implementaci očekávat problémy. Komunita okolo Jena je široká a představuje dostatečnou znalostní základnu.

6.1.2 Knihovna pro tvorbu GUI

Pro tvorbu grafického rozhraní aplikace se nabízelo hned několik možností, z nichž každé mělo svá pozitiva i negativa, která se zde pokusím v několika bodech shrnout. Nejdůležitějšími kritérii pro výběr knihovny byly zcela bezproblémový deployment na více OS, možnost oddělit aplikační a prezentační vrstvu a možnost vytvářet vlastní komponenty. O něco méně významná byla přítomnost vizuálního návrháře GUI.

- **Apache Pivot**[2]
  - \+ Definice GUI pomocí XML
  - \+ Stále v aktivním vývoji

  + Jedna knihovna pro všechny OS
  + Nezávislé na platformě
  – Zaměřeno hlavně na applety
  – XML nemá schéma, nelze tedy využít asistované editace v Eclipse
  – Chybí WYSIWYG editor
  – Malá znalostní základna

- **JavaFX 8**[11]
  + Nástupce knihovny Swing v aktivním vývoji
  + Možnost přizpůsobení vzhledu pomocí CSS
  + Definice GUI v XML + WYSIWYG editor
  + Snadná tvorba nových komponent
  + Není třeba externí knihovny (od Java 7)
  + Nezávislé na platformě
  – Nestabilní
  – Problémově vyřešené chování některých komponent

- **Qt Jambi**[17]
  + Rychlost a malé paměťové nároky
  + Možnost přizpůsobení vzhledu pomocí CSS
  + Možnost oddělení aplikační a prezentační vrstvy
  – Složitý deployment
  – Chabá podpora 64-bitových OS
  – Závislost na platformě
  – Dlouho stagnující vývoj

- **Standard Widget Toolkit**[20]
  + Rychlost a malé paměťové nároky
  + Široká znalostní základna
  – Vyžaduje externí knihovny pro každý OS
  – Závislé na platformě (vliv na stabilitu)
  – Problémové oddělení aplikační a prezentační vrstvy

- **Swing**[19]
  - \+ Stabilní, léty prověřené řešení
  - \+ Zcela nezávislé na platformě
  - \+ Široká znalostní základna
  - \+ Není třeba externí knihovny
  - – Již nevyvíjné, zastarávající řešení
  - – Problémové oddělení aplikační a prezentační vrstvy[1]

Ze všech těchto možností jsem se rozhodl pro knihovnu JavaFX, neboť i přes její momentální chybovost a problémovou stabilitu jde o moderní, multiplatformní řešení v aktivním vývoji s dobrým výhledem do budoucna. Není na místě očekávat problémy s deploymentem, neboť jde o součást Javy. Možnost grafické tvorby uživatelského rozhraní ukládaného do XML souboru mimo kód aplikace přináší programátorovi zásadní úsporu času a umožňuje aplikaci lépe strukturovat.

Druhou přijatelnou volbou by bylo použití platformy Swing jakožto stabilního řešení, proti kterému ale mluví nutnost řešit oddělení aplikační a prezentační vrstvy knihovnami třetích stran.

Stanoveným podmínkám částečně vyhovuje také Apache Pivot, nicméně chybějící XML schéma společně s malou znalostní základnou znamená by mohlo tvorbu GUI pomocí této knihovny výrazně ztížit.

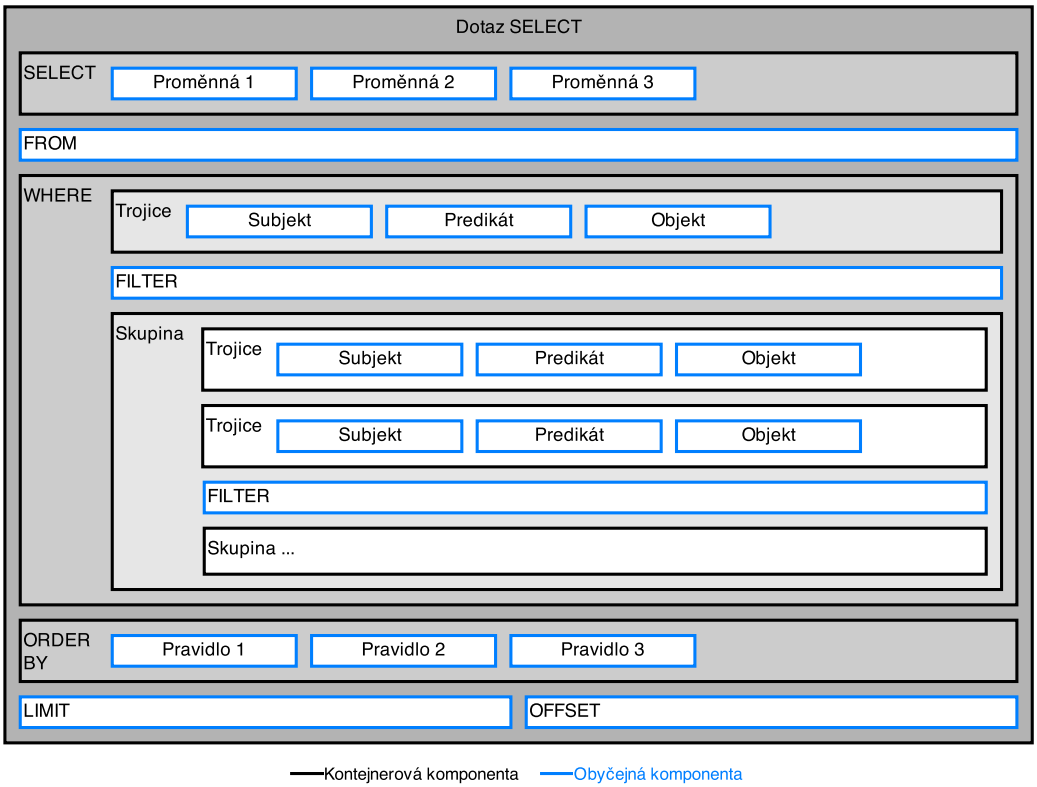
6.2 Návrh způsobu implementace

6.2.1 Vnitřní reprezentace dotazu

Jak lze zjistit pohledem na gramatiku SPARQL, každý dotaz má předem danou strukturu vyjádřitelnou stromem[30] (graf neobsahující cykly [5]). Stejným způsobem je organizováno i grafické rozhraní v JavaFX.[10] To je tvořeno dvěma druhy komponent – těmi, které mohou obsahovat jiné komponenty (kontejnery) a těmi, které ne.

[1] Lze řešit nástroji třetích stran.

V návaznosti na kapitolu 5.3.3 se tedy nabízí myšlenka reprezentovat dotaz stromem viditelných komponent, do nějž by mohl uživatel přímo zasahovat a tím dotaz editovat. Kupříkladu klauzule `WHERE` by mohla být vyjádřena kontejnerem obsahujícím komponenty představující trojici, filtr či další, vnořenou skupinu trojic a ten by se zas nacházel v kontejneru obsahujícím kromě `WHERE` i ostatní části dotazu. Myšlenku ukazuje obrázek 6.1.



Obrázek 6.1: Vnoření jednotlivých částí dotazu.

Komponenty reprezentující části dotazu bude nutné vytvořit. Každá z nich bude tvořena třídou, která bude poskytovat několik druhů výstupu:

- Část SPARQL dotazu, který komponenta obsahuje,
- data, která lze použít pro uložení a pozdější vyvolání dané části dotazu (např. element XML),
- ostatní užitečná data.

Různé části aplikace mohou vyžadovat informace o dění v každé z komponent uvnitř dotazu. Dalším krokem tedy bude vytvoření podmínek pro implementaci návrhového vzoru Observer. Bude tak možné zachytit například informaci o tom, že uživatel provedl v dotazu změnu a na základě toho jej později upozornit, že by měl změny uložit.

Alternativou k Observer pattern je možnost upozorňovat ostatní části aplikace na změny v komponentě událostmi.

6.2.2 Správa prefixů a pojmů

Ve smyslu kapitoly 5.3.3 bude potřeba, aby aplikace sama zjistila, které ze známých prefixů uživatel v dotazu používá a uvedla je v hlavičce dotazu společně s IRI jmenného prostoru, které zastupují.

K tomu lze využít stromové struktury dotazu nastíněné v kapitole 6.2.1. Aplikace jednoduše požádá komponentu na vrcholu stromu o seznam použitých prefixů, ta se podobně dotáže komponent uvnitř a tak dále až se požadavek dostane ke komponentám, které jsou schopné vrátit výsledek.

Kromě prefixů si bude aplikace muset pamatovat alespoň základní seznam datových typů a pojmů pro možnost asistované editace (dále v kapitole 6.2.3).

Pro zachování uživatelské přívětivosti bude nutné tyto informace do souboru s dotazem, který je využívá. V opačném případě by mohlo dojít k tomu, že dotaz vytvořený na jednom zařízení (resp. v jedné instalaci aplikace) nepůjde sestavit na zařízení druhém, protože aplikace na něm instalovaná nebude použité prefixy nebo datové typy znát.

6.2.3 Asistovaná editace

Zásadním přínosem, který by měla aplikace pro tvůrce dotazů mít, je její schopnost uživateli s editací aktivně pomáhat. Aplikace by si měla udržovat seznam známých pojmů, prefixů a proměnných a ten uživateli při psaní nabízet tak, aby stačilo napsat třeba jen začátek názvu pojmu a aplikace mohla na vyžádání doplnit zbytek.

Zde lze využít implementovaný návrhový vzor Observer (viz kapitola

6.2.1). V případě, že kterákoliv z komponent zjistí, že uživatel zavedl do dotazu novou proměnnou, může o tomto informovat své posluchače. Komponenta zaštiťující dotaz by mohla tyto události sledovat a udržovat si seznam proměnných, který by poté poskytovala ostatním komponentám za účelem nápovědy uživateli.

6.2.4 Výstupní formáty

Primárním účelem aplikace je tvorba SPARQL dotazů, ty by tedy měly být jedním z jejích výstupů. Způsob sestavení dotazu byl nastíněn v kapitole 6.2.1: Aplikace se pouze dotáže komponenty zaštiťující dotaz a ta se stejným způsobem dotáže komponent, které sama obsahuje. Získané fragmenty poté pouze spojí a vrátí aplikaci.

Jak zmiňuje kapitola 5.3.2, aplikace bude muset používat pro ukládání dotazů vlastní formát. Vzhledem ke způsobu reprezentace dotazu se nabízejí dva způsoby:

- **Serializace Java objektů** – Podmínkou tohoto způsobu ukládání souborů je *serializovatelnost* ukládaných objektů. Takový objekt musí implementovat rozhraní `java.io.Serializable` a nesmí obsahovat vlastnosti, které nejsou serializovatelné a zároveň nejsou označené klíčovým slovem `transient`.[12] Hlavní nevýhodou tohoto přístupu je znehodnocení již existujících souborů ve chvíli, kdy dojde ke změně kódu serializované třídy. Takové soubory by poté nešly jednoduše otevřít.
- **Uložení každého fragmentu dotazu jako XML elementu** – Ukládaný XML soubor by se sestavil podobným způsobem, jako to lze během tvorby SPARQL. Výhodami oproti klasické serializaci jsou snazší udržení kompatibility aplikace s dříve uloženými soubory a výstup v textové podobě.

Dalším výstupem aplikace budou soubory s výsledky dotazů. Viz kapitola 6.2.6.

6.2.5 Práce s úložištěm

Úkony, které bude aplikace provádět nad úložištěm a které jsou ve výchozím stavu podporovány knihovnou Jena:

- **Připojení** – Ke vzdálenému úložišti projektu MRE se bude aplikace připojovat prostřednictvím adaptéru Jena⇔Oracle. Lokální úložiště pro potřeby testování bude využívat Jena in-memory model se zálohováním na pevný disk uživatele. Pro zachování uživatelské přívětivosti budou údaje nutné k připojení k úložištím ukládány tak, aby uživateli stačilo zadat je pouze jednou.
- **Import dat a čištění uložiště** – Pro potřeby testování.
- **Provádění dotazů** – Vzhledem k tomu, že bude muset aplikace zůstat ovladatelná i při provádění dotazů, bude potřeba tuto činnost přesunout do jiného vlákna a na vzniklé události reagovat asynchronně. Samozřejmostí je možnost probíhající operaci zrušit (alespoň vizuálně).

6.2.6 Reprezentace výsledků dotazu

Výsledky dotazů budou nabývat tří forem:

- **Tabulka** – Bude výstupem dotazu typu SELECT. Řešit bude potřeba způsob výpisu jednotlivých entit (např. zkracování IRI, reprezentace různých datových typů) a následně uložení tabulky do souboru v univerzálně použitelném formátu (např. CSV).
- **Hodnota ANO/NE** – Výstup dotazu typu ASK. Nemá smysl ukládat, stačí zobrazit uživateli.
- **Graf** – Je výsledkem dotazů CONSTRUCT a DESCRIBE. Knihovna Jena vrací výsledný graf v podobě vlastní abstraktní datové struktury `Model`.[2] Tu lze následně převést do textového formátu RDF/XML a dalších [3] a zobrazit či uložit.

[2]Balík `com.hp.hpl.jena.rdf.model`.

7 Implementace

Zjednodušené UML diagramy tříd dělené podle balíků jsou k dipozici v příloze B.

7.1 Vnitřní uspořádání aplikace

Kód aplikace je organizován do balíků s prefixem `cz.zcu.kiv.sparkle`. Balík s tímto názvem obsahuje třídu `Sparkle` implementující vstupní metodu `main()`, která funguje jako vstupní bod a má na starost start aplikace. Ten zahrnuje zobrazení formuláře pro připojení k úložišti a následně hlavního formuláře aplikace, kterému v případě potřeby předá požadavek na otevření souborů uvedených na příkazové řádce. Kromě toho jsou součástí balíku třídy pro lokalizaci a uchování nastavení aplikace, ikony a splash screen.

Obsah vnořených balíků:

- `data` – Třídy značené jako *Storage*, jejichž instance se starají o správu známých prefixů, datových typů a dalších pojmů a třída `DataAgent`, jejíž instance zapouzdřují datový model a slouží jako rozhraní pro operace nad úložištěm.
- `gui.evaluation` – Rozhraní, kontrolery a XML definice komponent (dále viz 7.3.1), které tvoří uživatelské rozhraní pro vyhodnocování dotazů.
- `gui.forms` – Kontrolery a XML definice formulářů a dialogových oken.
- `gui.query` – Rozhraní, kontrolery a XML definice komponent, které tvoří uživatelské rozhraní pro tvorbu dotazů.
- `gui.query.helpers` – Pomocné třídy a rozhraní se silnou vazbou na obsah balíku `gui.query`.
- `gui.tools` – Nástroje a abstraktní třídy s vazbou na uživatelské rozhraní.
- `tools` – Definice, nástroje a utility, které se jinam nevešly.

7.2 Datové rozhraní

7.2.1 Třída DataAgent

Funguje jako nadstavba pro RDF nástroje knihovny Jena. Jejím hlavním účelem je odstínit zbytek aplikace od operací na datové vrstvě v čele s instancemi tříd implementujícími rozhraní `Model`, které Jena poskytuje jako jeden z prostředků pro práci s RDF úložišti.

Poskytuje metody pro asynchronní dotazování, import RDF souborů do úložiště a jeho čištění a metodu pro hledání cesty grafem v připojeném úložišti.

K třídě `DataAgent` se váží také třídy `DataHelper` a `DataAgentFactory` ze stejného balíku. První uvedená poskytuje aplikaci pomocné metody (jako například získání přibližných informací o typu hodnoty proměnné v dotazu) a druhá vytváří instance `DataAgent` v závislosti na druhu úložiště, kam je třeba se připojit.

7.2.2 Storage třídy

Sada serializovatelných tříd uchovávajících informace vyžadované nebo pomáhající při tvorbě dotazů.

- `PrefixesStorage` – Uchovává mapování prefix⇔IRI.
- `AbstractTermsStorage` – Slouží jako základ pro třídy uchovávající známé pojmy, jako jsou datové typy (třída `DataTypesStorage`), funkce (pro použití při filtrování; třída `FunctionsStorage`) a ostatní zdroje (`ResourcesStorage`). Pojmy třídí podle jmenných prostorů, kam spadají.

Všechny uvedené třídy implementují rozhraní `Saveable`[1] a jejich data tak lze vložit do XML souboru společně s dotazem (více v kapitole 7.4.3). Kromě toho poskytují statické metody pro vytvoření nových instancí již obsahujících pojmy a jmenné prostory ze slovníků dodávaných přímo s knihovnou Jena.

[1] Balík `cz.zcu.kiv.sparkle.tools`.

7.3 Grafické rozhraní

Aplikace je tvořena několika formuláři (okny aplikace) a kromě komponent, které jsou součástí JavaFX, používá také sadu vlastních.

Formuláře i komponenty jsou z větší části definovány v šablonách mimo kód aplikace (viz kapitola 7.3.1). Výjimkou jsou vlastní komponenty, které jsou vytvářeny a na formulář umisťovány uvnitř kódu. Stejný postup je používán také při dynamickém generování formuláře pro zápis dotazu (viz kapitola 7.4).

7.3.1 Šablony a kontrolery

Třídy zmíněné v této podkapitole se nachází v balíku `gui.tools`.

Veškeré vizuální části aplikace jsou tvořeny XML definicí (soubory s příponou `.FXML`) a kontrolerem. Kontroler slouží jako aplikační vrstva – obsahuje reakce na uživatelský vstup, případně odkazy na vizuální prvky, obojí s anotací `@FXML`.

V případě formulářů je uvnitř FXML souboru přímo uveden plný název třídy, která bude sloužit jako kontroler. Kontrolery formulářů aplikace musejí být odděděny od třídy `AbstractFormController`.

```
MyWindow wnd = FormControllerFactory.load(ownerWindow,
    MyWindow.class);
wnd.show();
```

Kód 7.1: Načtení okna.

Načtení šablony okna ze souboru a vytvoření instance kontroleru má na starost statická metoda `load()` třídy `FormControllerFactory`. Té je mimo jiné předávána třída kontroleru, podle jejíhož názvu se hledá příslušná FXML šablona.

Komponenty jsou definovány podobným způsobem, ale kontroler je nutné jim přiřadit až při běhu aplikace. To provádí statická metoda `load()` z třídy `Components`. Měla by být volána až uvnitř konstruktoru, tj. instanci kontroleru je třeba vytvářet ručně.

Kontroler v tomto případě přímo stanovuje typ komponenty. Proto například pro vytvoření komponenty rozšiřující možnosti obyčejného tlačítka musí její kontroler dědit od třídy reprezentující ono tlačítko.

```
class MyButton extends Button {
    public MyButton() {
        Components.load(this);
    }
}
// ...
MyButton btn = new MyButton();
```

Kód 7.2: Inicializace komponenty.

7.4 Tvorba dotazu

Jak navrhuje kapitola 6.2.1, dotaz se v aplikaci skládá z do sebe zanořených uživatelsky definovaných komponent z balíku `gui.query`. Zanořením se myslí to, že komponenta představující klauzuli `WHERE` obsahuje libovolné množství komponent reprezentujících trojice (třída `TriplePane`), filtry (`FilterPane`) či skupiny trojic (`GroupGraphPatternPane`). Trojice může obsahovat další trojice sdílející stejný subjekt a skupina trojic může obsahovat opět další trojice, filtry či skupiny.

Každá komponenta zodpovídá v dotazu pouze za tu část, kterou reprezentuje. Pokud se její část dotazu skládá z více kusů tvořených vnořenými komponentami, je její povinností v nich sledovat změny a sbírat z nich data.

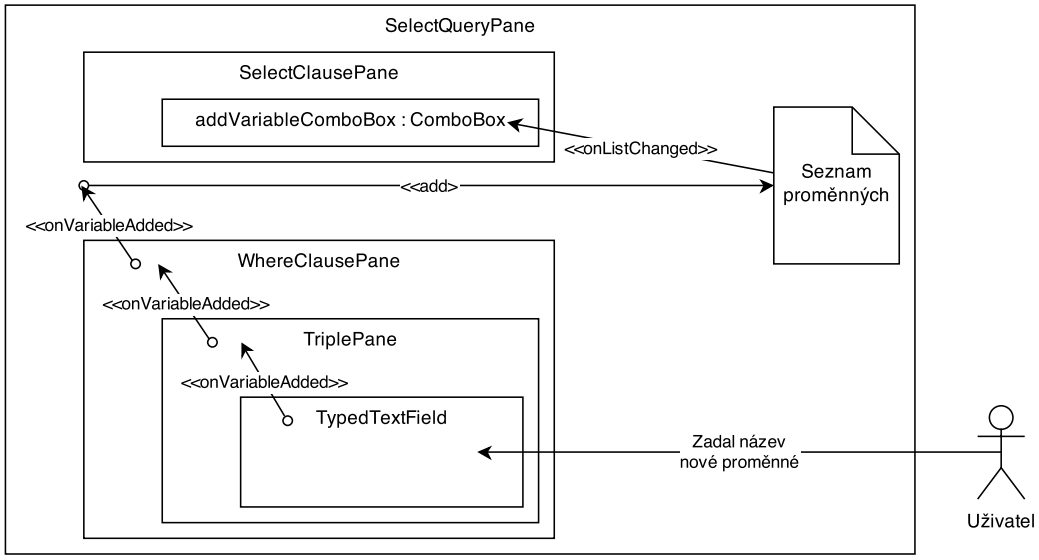
Každý dotaz má jádro – kontejnerovou komponentu s kontrolerem odvozeným od třídy `QueryPane`. Ta zapouzdřuje celý dotaz a slouží jako rozhraní pro komunikaci mezi dotazem a zbytkem aplikace. Uvnitř komponent dědících `QueryPane` mají místo komponenty představující jednotlivé části dotazu, jejichž název vždy končí na `ClausePane`, tedy např. `SelectClausePane`.

7.4.1 Propagace událostí

`QueryPane` naslouchá událostem uvnitř komponent, které obsahuje a příslušným způsobem na ně reaguje. Jako příklad lze uvést situaci, kdy uživatel de-

finuje uvnitř `SelectClausePane` novou proměnnou. Rodičovský `QueryPane` to zjistí a umístí ji do seznamu používaných proměnných. Ten lze využít pro potřeby asistované editace (kapitoly 6.2.3, 7.5.1) nebo při automatickém zavádění nových proměnných.

Komponenty umožňující posluchačům sledovat operace s proměnnými musí implementovat rozhraní `VariablesGenerator`.[2] Posluchač implementuje rozhraní `VariablesCollector`.



Obrázek 7.1: Propagace události použití nové proměnné.

Obrázek 7.1 ukazuje, jak probíhá propagace události zadání názvu proměnné z editačního pole v trojici do jádra dotazu SELECT – instance třídy `SelectQueryPane` – a následně do rozbalovacího seznamu umožňujícího přidat již existující proměnnou do klauzule `SELECT`.

Dalším rozhraním pro sledování změn, je rozhraní `Changeable` z balíku `tools`. To je v dotazech používáno pro sledování obecných změn (změny v editačních polích, přidání trojice atd.) a umožňuje aplikaci určit, zda byl dotaz upraven a je třeba na to uživatele upozornit při zavírání souboru.

[2]Balík `gui.query.helpers`.

7.4.2 Sestavení dotazu

Komponenty poskytující data k sestavení SPARQL dotazu implementují rozhraní `PartialQueryGenerator` z balíku `query.helpers`. To deklaruje metodu `getQueryPart()` vracející textovou podobu části dotazu, kterou komponenta zastupuje.

Dalším druhem dat, který komponenty k sestavení dotazu generují, je seznam používaných prefixů. Komponenta, která umožňuje uživateli použití prefixem zkrácených IRI, implementuje rozhraní `PrefixesUser` (opět z `query.helpers`) deklarující metodu `getPrefixesUsed()`.

Pro sestavení dotazu zavolá aplikace metodu `getQuery()` třídy `QueryPane`. Ta nejprve získá od vnořených komponent seznam použitých prefixů a ty, které zná, vloží do hlavičky vznikajícího dotazu společně s IRI příslušných jmenných prostorů. Následuje volání `getQueryPart()` nad každou částí dotazu a doplnění klíčových slov jako `CONSTRUCT`, `SELECT` nebo `ORDER BY`.

7.4.3 Uložení a vyvolání

Ukládání dotazů do XML funguje na stejném principu jako sestavování dotazu popsané v kapitole 7.4.2. Komponenty, jejichž data lze uložit a následně vyvolat, implementují rozhraní `Saveable` z balíku `tools`. To deklaruje tři metody:

- `getXMLElementName()` – Vrací název XML elementu, pod jakým by měla být data komponenty uložena. Takovou informaci potřebuje znát její rodičovská kontejnerová komponenta pro její uložení a pozdější vyvolání.
- `save(org.w3c.dom.Element e)` – Provede uložení sama sebe do poskytnutého elementu XML dokumentu.
- `load(org.w3c.dom.Element e)` – Načte sama sebe z poskytnutého XML elementu. Nejde o statickou metodu, komponenta již musí být vytvořena.

O uložení dotazu žádá aplikace třídu `QueryPane` voláním metody `save()`, které předává odkaz na výstupní soubor. Ta nejprve vytvoří nový XML dokument, v něm kořenový element s atributem stanovujícím typ dotazu a

následně prochází všechny komponenty, které `QueryPane` obsahuje. Když narazí na komponentu implementující `Saveable`, vytvoří pro ni v dokumentu nový element a ten jí předá v metodě `save()`. Pak volá tutéž metodu i pro s dotazem svázané instance Storage tříd (viz kapitoly 7.2.2, 7.4.4).

Načtení dotazu zůstává v kompetenci `QueryPane`, ale protože není dopředu známo, jaký druh dotazu chce uživatel načíst, je metoda `load()` statická. Soubor otevře, získá z něj kořenový element a jeho atribut uvádějící typ dotazu. Na základě toho vytvoří novou instanci jedné z oděděných tříd a v konstruktoru jí kořenový element předá. Pak už načítání probíhá standardně: Každá kontejnerová komponenta ví, jaké elementy může obsahovat a ty v XML souboru hledá. Pokud nalezne, vytvoří instanci příslušné třídy a nalezený element jí v metodě `load()` předá.

7.4.4 Správa prefixů a pojmů

Návrh na implementaci této funkcionality rozebírá kapitola 6.2.2.

Aplikace udržuje celkem čtyři druhy seznamů (viz kapitola 7.2.2). V běžící aplikaci existuje pro každý z nich jedna instance vázaná na aplikaci (nazývaná *lokální*) a jedna vázaná na každý z otevřených dotazů.

Ke každému nově vytvářenému dotazu se vytváří kopie lokálních seznamů, které jsou s ním následně distribuovány. Kopie jsou synchronizovány s lokálními seznamy a to tak, aby lokální seznam *doplňoval* seznam vázaný na dotaz.

Synchronizaci řeší každá instance třídy `QueryPane` tím, že naslouchá změnám v lokálním seznamu a udržuje si seznam těch mapování či pojmů, která do něj byla doplněna. Změny (v lokálních seznamech) se pak promítnou i v dotazu pouze v případě, že jsou v tomto seznamu zmíněna.

Data z lokálních seznamů aplikace ukládají v adresáři s uživatelskými daty. Jeho umístění je závislé na operačním systému. Pokud některý ze seznamů nelze načíst, aplikace automaticky vytvoří prázdný a naplní jej přednastavenými hodnotami.

Pro uživatelskou správu seznamů disponuje aplikace čtyřmi formuláři.[3] Ty dovolují uživateli ručně přidávat mapování, funkce či datové typy. Výjimku tvoří ostatní zdroje (v aplikaci pojmenované jednoduše *resources*),

[3] `PrefixesDialog`, `DataTypesDialog`, `FunctionsDialog` a `ResourcesDialog`.

které lze pouze importovat z RDF souborů, případně rovnou z právě připojeného úložiště. K oběma druhům importu je používán SPARQL dotaz, jehož vyhodnocení bohužel bývá časově velice náročné.

Pokud se uživatel pokusí prostřednictvím dialogu odstranit nějaký prefix, aplikace nejprve kontroluje, zda není v příslušném dotazu používán. Pokud je, k odstranění nedojde.

7.4.5 Chytrá editační pole

Třída `TypedTextField` z balíku `gui.query.helpers` představuje komponentu funkčně výrazně rozšířeného editačního pole. Ve způsobu jakým aplikace reprezentuje uživateli SPARQL dotaz, může zastávat jeden z těchto SPARQL elementů:

- **Proměnná** – Uživatel může v tomto poli uvést název proměnné. Komponenta se sama postará o přidání prefixu ‚`?`‘ a na každou změnu upozorní své posluchače (implementuje rozhraní `VariablesGenerator` (viz kapitola 7.4.1). Pokud je název proměnné neplatný, zbarví se pozadí pole do červena.
- **Prefixem zkrácený IRI (*Prefixed name*)** – Uživatel může uvést IRI ve zkrácené podobě `prefix:název_zdroje`. Použití prefixu je evidováno (implementuje rozhraní `PrefixesUser`, viz kapitola 7.4.2). V případě neplatné syntaxe se pole zbarví do červena.
- **Plný IRI** – Pro uvedení nezkráceného identifikátoru. Prefix ‚`<`‘ a suffix ‚`>`‘ je doplněn automaticky. V případě neplatné syntaxe se pole zbarví do červena.
- **Klíčové slovo** `a` – V tomto případě je pole pouze pro čtení.
- **Literál** – Pro uvedení literálu. Žádná kontrola syntaxe se neprovádí. Prefix a suffix ‚`"`‘ je doplněn automaticky a pokud se vyskytuje v textu, je escapován.
- **Výraz** – V tomto poli může uživatel napsat libovolný výraz. Závorky kolem výrazu jsou doplněny automaticky. Syntaxe je kontrolována pomocí metody `parse()` třídy `ExprUtils`[4] knihovny Jena.

[4]Balík `com.hp.hpl.jena.sparql.util`.

Který element v dotazu pole zastává, si může uživatel vybrat pomocí tlačítek zobrazených nad polem v případě, že je zaměřené. Sada dostupných tlačítek je závislá na pozici pole v dotazu. Např. zastupuje-li pole subjekt, nemá smysl zobrazovat tlačítko pro přepnutí pole na typ obsahující literál.

V případě typů pro zápis zkráceného IRI, proměnné a výrazu pole uživateli s editací pomáhá – viz kapitola 7.5.1.

7.5 Doplňková funkcionalita

7.5.1 Asistovaná editace

Chytrá editační pole umějí v některých případech zobrazit uživateli seznam návrhů, co do pole zadat a to v závislosti na textu, který se v poli nachází za kurzorem.

Pokud je pole nastaveno na zadávání proměnných, seznam dává uživateli na výběr z dříve použitých proměnných.

Pokud je typ pole nastaven na zadávání zkrácených IRI, seznam s návrhy obsahuje nejprve známé prefixy a pokud se již v poli nějaký prefix nachází, seznam obsahuje jména známých zdrojů z příslušného jmenného prostoru.

V případě, že je pole nastaveno na zadávání výrazů, seznam návrhů je závislý na aktuálním kontextu a nabízí kombinaci uvedených možností a navíc seznam známých funkcí a datových typů.

Plovoucí seznam s návrhy se zobrazuje přímo pod editačním polem. Zobrazen je buďto automaticky nebo na výzvu uživatele (viz tabulka klávesových zkratek A.1) nebo jako reakce na napsání konkrétního znaku – otazník zobrazí seznam proměnných, dvojtečka seznam zdrojů k prefixu vlevo od pozice kurzoru.

Funkcionalitu asistované editace zajišťuje třída `AutoCompleteWrapper` z balíku `gui.query`. Ta v konstruktoru očekává mimo jiné textové pole, ke kterému se má funkcionalita vázat a instanci třídy implementující rozhraní `AutoCompleteListHandler`.[5]

[5]Definováno uvnitř `AutoCompleteWrapper`.

`AutoCompleteListHandler` deklaruje metodu `getAutoCompleteList()` vracející aktuální seznam návrhů. Wrapper ji volá pokaždé, když je potřeba seznam obnovit.

Aplikace obsahuje tři handlery, vždy jeden pro každý případ uvedený výše. Handler mající na starost návrh zkrácených IRI[6] lze navíc nastavit tak, aby navrhoval pouze vlastnosti, tedy zdroje, které jsou typu `rdf:Property`. Toho využívají editační pole zastávající v trojici úlohu predikátu.

7.5.2 Hledání cest grafem

Při dodržení určitých podmínek umožňuje aplikace nalézt cestu mezi zdrojem a vlastností jiného zdroje a příslušné mezikroky doplnit do dotazu. Lze tak například nalézt ontologií definovaný vztah mezi pacientem a vlastností mající za hodnotu směrovací číslo jeho bydliště.

Vstupem pro hledání cesty je název proměnné stojící na začátku cesty, IRI vlastnosti tvořící předposlední článek řetězce cesty a název proměnné stojící na konci cesty. Výsledkem operace má být doplnění všech chybějících článků mezi počáteční proměnnou a vlastností stojící před koncovou proměnnou.

Algoritmus začíná identifikací proměnné, při které se aplikace snaží zjistit, jakého budou její hodnoty typu. Čerpá přitom ze dvou zdrojů informací. Prvním zdrojem je dotaz samotný, kde aplikace hledá trojici s počáteční proměnnou v subjektu následovanou predikátem `rdf:type` a IRI zdroje na pozici objektu. Pokud takovou trojici nalezne, přidá ji na seznam.

Druhým zdrojem je úložiště. Aplikace vyextrahuje z dotazu ty části, které přímo či nepřímo odkazují na počáteční proměnnou. Z nich složí dotaz. Odpovědí něj by byly konkrétní hodnoty, kterých může počáteční proměnná nabývat. Aplikace tak navíc doplní trojici zjišťující jejich typ. Dotaz odešle a z výsledku získané typy doplní do seznamu.

Seznam nyní obsahuje zdroje, kde začíná hledaná cesta. Aplikace postupně pro každý z nich volá metodu `findForwardOntologyPath()` třídy `DataAgent`. Ta prohledává graf metodou *do šířky*.

Způsob hledání se dělí na dva způsoby podle toho, zda je či není rozbalovaný uzel definovaný jako vlastnost. Pokud ano, aplikace získá všechny hod-

[6] Třída `PrefixedNamesAutoCompleteListHandler`.

noty jeho vlastnosti `rdfs:range` a ty umístí do fronty k prohledávání. V opačném případě jsou do fronty umístěny všechny hodnoty vlastnosti `owl:hasKey` a také všechny další zdroje, které mají právě prohledávaný uzel jako hodnotu vlastnosti `rdfs:domain`.

Rozbalený uzel je odebrán z fronty a vložen do kolekce již prohledaných uzlů a to včetně odkazu na jeho předchůdce. V případě nalezení koncového uzlu je tak možné vystopovat cestu zpět a zároveň existuje evidence toho, kudy už prohledávání prošlo a lze zabránit zacyklení, pokud graf obsahuje cykly.

Funkční omezení

Výsledky prohledávání jsou bohužel poznamenány definicí `rdfs:domain` a `rdfs:range` (viz kapitola 2.2). V jejím smyslu mohou být do domény a rozsahu vlastností (např. inferencerem[7] doplněny zdroje, které prohledávání nasměrují špatnou cestou.

Problém lze ukázat na jednoduchém příkladu: Vlastnost `clinicalEvent` definovaná v ontologii DASTA má v rozsahu třídu `ClinicalEvent`, která je zároveň zdrojem a do rozsahu tedy patří i třída `rdfs:Resource`, která se později dostane mezi prohledávané uzly.

Při pokusu o rozbalení uzlu `rdfs:Resource` algoritmus nejprve zkusí přidat do fronty všechny hodnoty vlastnosti `owl:hasKey`. Žádné takové samozřejmě nenalezne a pokračuje tím, že přidá do fronty všechny vlastnosti, které mají `rdfs:Resource` v doméně. A zde vzniká problém, protože tato podmínka platí pro všechny vlastnosti, které mají v doméně uvedenu libovolnou jednu nebo více tříd.

7.5.3 Vyhodnocování dotazů

K vyhodnocování dotazů slouží metody `select()`, `ask()`, `construct()` a `describe()` definované v třídě `DataAgent`. Těm je kromě dotazu ke zpracování předávána instance třídy `CancellableConsumer` z balíku `tools`, které jsou během vyhodnocování dotazu předávány výsledky. Tímto je docíleno

[7] Od angl. *infer – odvodit*. Aplikace odvozující nová fakta na základě existujících tvrzení.

možnosti vyhodnocovat dotaz asynchronně (aplikace nemusí čekat na návratovou hodnotu metody).

`CancellableConsumer` se může nacházet ve třech stavech.[8] Stav `Normal` je výchozím stavem, do stavu `Finished` se třída dostane po obdržení posledního výsledku a ve stavu `Cancelled` se nachází poté, co byla jinou částí aplikace požádána o přerušení vyhodnocování.

Interně mají na starost zpracování dotazu instance tříd implementující rozhraní `QueryExecution` knihovny Jena. To deklaruje metody pro vyhodnocení jednotlivých druhů dotazů a také metodu `abort()` umožňující vykonávání dotazu přerušit, volanou právě ve chvíli, kdy se dostane instance `CancellableConsumer` do stavu `Cancelled`.

Zobrazení výsledku

Zobrazování výsledků je v GUI řešeno odděleně od psaní dotazu – v samostatné záložce. Ke každé záložce s dotazem (třída `QueryTab` z balíku `gui.query`) lze otevřít maximálně jednu záložku s výsledky (`EvaluationTab` z balíku `gui.evaluation`).

Tabulku, která je výsledkem dotazu typu SELECT, zobrazí komponenta `SelectResultsPane` oddědděná od komponenty `TableView` z knihovny JavaFX. Oproti ní disponuje kontextovým menu s volbami pro uložení celého výsledku do formátu CSV. Po dvojitém kliknutí na buňku je uživateli zobrazeno okno s s její hodnotou tak, aby ji mohl například zkopírovat do schránky nebo uložit do souboru.

Hodnotu ano/ne, která je výsledkem dotazu ASK zobrazuje komponenta `BooleanResultsPane`.

Třetí formou výsledku je graf (resp. instance třídy implementující rozhraní `Model` z knihovny Jena). Ten se uživateli zobrazuje v textové podobě uvnitř komponenty `ModelResultsPane`. Na výběr je z pěti formátů a výsledek lze uložit do souboru.

[8] Dle hodnot výčtového typu `State` definovaného uvnitř `CancellableConsumer`.

Manuální úprava vygenerovaného dotazu

Součástí záložky pro vyhodnocení dotazu (viz kapitola 7.5.3) je také editační pole, kde je možné vyhodnocovaný dotaz upravit ručně. Lze tak nechat vyhodnotit i dotazy, jejichž psaní aplikace neumožňuje.

8 Dosažené výsledky

Aktuální verze aplikace je dobře použitelná pro psaní jednoduchých a středně složitých dotazů využívajících převážně schopností jazyka SPARQL 1.0 (viz dále 8.1). Snaží se odstínit uživatele od povinností s tím spojených a poskytnout mu přitom jistou míru komfortu.

8.1 Implementovaná podmnožina SPARQL

Výsledná aplikace pokrývá svými schopnostmi všechny výrazové prostředky jazyka SPARQL 1.0, s výjimkou klauzule `BASE`, která je snadno nahraditelná využitím zkrácených IRI.

Implementované výrazové prostředky SPARQL 1.1 Query jsou následující:

- `MINUS` – Bez omezení.
- `FILTER EXIST`, `FILTER NOT EXIST` – Implementováno pouze jako modifikátor skupiny.
- Vestavěné funkce s výjimkou agregačních.

Protože lze vygenerovaný dotaz v aplikaci ručně upravit, je možné získat i výsledky dotazu využívajícího nepodporovanou podmnožinu SPARQL.

8.2 Známá omezení a možnosti rozšíření

Aplikace byla napsána s ohledem na další vývoj. Díky systému komponent s funkčností definovanou rozhraními, je možné v malém čase implementovat podporu dalších výrazových prostředků SPARQL. Vývojář bude nicméně stále častěji narážet na limity způsobené omezenou velikostí okna aplikace a bude muset řešit otázku, zda vyměnit uživatelskou přívětivost za funkčnost.

8.2.1 Agregace

Prioritou při dalším vývoji by měla být implementace podpory agregačních funkcí, které mohou uživateli nyní citelně chybět. Pro ně nutné vytvořit novou třídu, která by je odlišila od běžných vestavěných a uživatelských funkcí v seznamu `FunctionsStorage` (viz kapitola 7.2.2). Jistá forma odlišení bude potřeba také v dialogu pro správu funkcí. Souvisejícím, méně náročným úkolem je implementace klauzulí `GROUP BY` a `HAVING`, případně `BIND`.

8.2.2 Vnořené výrazy

Stěžejním vylepšením SPARLQ 1.1 Query jsou vnořené výrazy (viz kapitola 4.8.6). Aplikace je v současné dobře nepodporuje a jejich implementace by znamenala znatelný posun v její využitelnosti.

U vnitřní reprezentace vnořeného dotazu se lze inspirovat u ostatních komponent, ze kterých se dotaz skládá. Větším problémem je vizuální stránka věci. Vývojář se bude muset rozhodnout, zda chce zobrazovat vnořený dotaz celý nebo jen komponentu, která jej zastupuje a umožnit uživateli editovat vnořený dotaz v jiném okně.

8.2.3 Drag'n'drop

Jedním z konkrétních navrhovaných vylepšení je možnost přesouvat trojice a další komponenty mezi skupinami tak, aby bylo kupříkladu možné zbavit některou z trojic modifikátoru `OPTIONAL`. Tato funkce je v současné chvíli předpřipravena v podobě drag'n'drop. Libovolnou trojici lze uchopit myší a přesunout ji do jiné skupiny.

V tuto chvíli lze přesouvat pouze trojice. Ty lze „upustit" do jiné skupiny a přesunutá trojice je umístěna na konec. Serializace a deserializace komponenty před a po upuštění je řešena metodami `save()` a `load()` (viz kapitola 7.4.3).

Návrhy na vylepšení drag'n'drop:

- **Implementovat pro více prvků** – Přidat možnost přesunu skupin či filtrů.

- **Umožnit přesun jinam než na konec skupiny a zvýraznit místo, kam bude komponenta přesunuta**
- **Přidat „madlo"** – Vizuální komponenta, za kterou bude uživatel moci přesouvaný prvek uchopit.
- **Doplnit vizualizaci přesunu** – Nyní zůstává při přetahování komponenta na místě a probíhající přesun signalizuje pouze kurzor myši.

8.2.4 Tlačítka zpět a vpřed

Stěžejní funkce každého editoru, jejíž implementace bude ale v tomto případě poměrně obtížná. Nabízí se využití rozhraní `Changeable`, díky kterému se aplikace dozvídá o kterékoliv změně v dotazu a o komponentě, kde se změna stala. Rozhraní by mělo být možné upravit tak, aby přenášelo stav komponenty před a po změně a aplikace by si mohla tyto informace uchovávat. V případě kliknutí na tlačítko zpět by pak mohla vyvolat původní stav komponenty.

8.2.5 Optimalizace asistované editace a hledání cest

Seznam návrhů (viz 7.5.1) by mohl být tvořen inteligentně, např. na základě znalosti ontologie, nad kterou uživatel pracuje. Nabízí se také možnost položky v seznamu obarvovat podle jejich typu.

Pod tvorbou seznamu se znalostí ontologie si lze představit např. algoritmus, který na místo objektu navrhuje pouze hodnoty z domény vlastnosti na místě predikátu editované trojice. Z kapitoly 7.5.2 nicméně vyplývá, že `rdfs:domain` k tomuto účelu využít nelze.

S tím souvisí další návrh na vylepšení aplikace a sice implementace spolehlivějšího algoritmu hledání cest (kapitola 7.5.2).

8.2.6 Další vylepšení

- **Připojení k jinému úložišti** – Položka v menu, která by umožnila

uživateli okamžité přepojení na jiné úložiště bez nutnosti vypnout aplikaci.

- **Jeden dialog při zavírání více souborů** – V současné době znamená uzavření aplikace s N otevřenými a pozměněnými soubory dotaz na uložení každého z nich.
- **Seznam nedávno otevřených souborů**
- **Další druhy úložišť** – Přidání možnosti připojit se k online či lokálnímu datasetu a k dalším službám nezaloženým na Oracle databázi, např. ke SPARQL Endpoint.[23]
- **Zvýrazňování syntaxe** – Při ruční modifikaci dotazu, resp. při zobrazování výsledků dotazů CONSTRUCT a DESCRIBE.
- **Ovládání reasoneru** – V současné době je reasoner nad lokálním úložištěm vždy zapnutý. Mělo by být možné jej vypnout, případně nastavit.

8.2.7 Jiná omezení

Uživatel se může čas od času setkat s anomáliemi v uživatelském rozhraní, které si mohou v krajním případě vyžádat restart aplikace. Tyto problémy jsou způsobeny použitím knihovny JavaFX 8, která v době psaní práce neoplývá stabilitou a je stále v aktivním vývoji (viz kapitola 6.1.2). Uživatelský zážitek se tak může lišit podle verze knihovny, kterou uživatel disponuje.

Na problémy lze narazit také v případě použité knihovny Jena, obzvlášť v případech, kdy se uživatel rozhodne, že nechce čekat na dokončení operace zahrnující vyhodnocení dotazu a zruší ji. Vizuálně ke zrušení operace dojde, ale pod povrchem vše stojí na ochotě knihovny současnou aktivitu přerušit a není výjimkou, že k tomu nedojde v rozumném čase. Z důvodu tendence knihovny zpracovávat požadavky sekvenčně pak může jiná operace čekat na dokončení té právě běžící.

Druhým, souvisejícím problémem, je fakt, že použitá verze Jena nepodporuje paralelní zpracování dotazu v místním úložišti. Zatíženo tak zůstane vždy jen jedno jádro procesoru, což se negativně projeví na výkonu aplikace.

V obou případech může být řešením upgrade na novější verzi knihovny Jena.

Závěr

Výsledkem této práce je multiplatformní nástroj umožňující uživateli pohodlné psaní dotazů v jazyce SPARQL. Jeho mottem je odstínit uživatele od formalit, které s sebou tato činnost přináší a s úkony, které nelze udělat automaticky, mu pomoci. Jeho silnou stránkou je rychlá tvorba krátkých dotazů s možností je okamžitě vyhodnotit. Díky podpoře lokálního úložiště se hodí pro testovací účely.

V současné době aplikaci nicméně schází některé schopnosti, které by usnadnily její nasazení jakožto pomocného nástroje v produkčním prostředí, v čele s podporou agregačních funkcí nebo možností připojení k více druhům úložišť. Lépe tak poslouží tak spíše jako demonstrace způsobu tvorby dotazu, který se u aktuálně dostupných aplikací řešících stejný problém příliš nevidí. Práce byla nicméně psána s ohledem na možnost budoucího rozšíření a doplnění nových funkcí či rozšíření podpory standardu SPARQL by tak nemělo znamenat větší problém. Obsáhlou sadu podnětů na rozšíření funkcionality lze nalézt v kapitole 8.2.

Seznam užitých zkratek

DASTA	DAta STAndard
GUI	Graphical User Interface
IRI	Internationalized Resource Identifier
MRE	Medical Research and Education
N3	Notation 3
OWL	Ontology Web Language
RDF	Resource Description Framework
RDFS	RDF Schema
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
URI	Uniform Resource Identifier
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language

Literatura

[1] *Apache Jena* [online]. [cit. 24.6.2014]. Dostupné z: http://jena.apache.org/.

[2] *Apache Pivot* [online]. [cit. 24.6.2014]. Dostupné z: https://pivot.apache.org/.

[3] APACHE.ORG. *Model* [online]. [cit. 18.6.2014]. Dostupné z: http://jena.apache.org/documentation/javadoc/jena/com/hp/hpl/jena/rdf/model/Model.html.

[4] BERNERS-LEE, T. – FIELDING, R. – MASINTER, L. *Uniform Resource Identifier (URI): Generic Syntax* [online]. IETF, January 2005. Dostupné z: http://www.ietf.org/rfc/rfc3986.txt.

[5] ČADA, R. – KAISER, T. – RYJÁČEK, Z. *Diskrétní matematika*. Plzeň, Česká Republika : Západočeská univerzita v Plzni, 2004. ISBN 80-7082-939-7.

[6] COMPANY, H. M. H. P. *The American Heritage® Dictionary of the English Language*. Boston, Massachusetts, USA : Houghton Mifflin Harcourt Publishing Company, 4th edition, 2011. Dostupné z: http://ahdictionary.com/word/search.html?q=format.

[7] COMPANY, H. M. H. P. *The American Heritage® Dictionary of the English Language*. Boston, Massachusetts, USA : Houghton Mifflin Harcourt Publishing Company, 5th edition, 2014. Dostupné z: http://ahdictionary.com/word/search.html?q=vocabulary.

[8] *DASTA - Datový standard pro předávání dat mezi informačními systémy zdravotnických zařízení* [online]. 2014. [cit. 22.6.2014]. Dostupné z: http://www.dastacr.cz/.

[9] DUERST, M. – SUIGNARD, M. *Internationalized Resource Identifiers (IRIs)* [online]. IETF, January 2005. Dostupné z: http://www.ietf.org/rfc/rfc3987.txt.

[10] HOMMEL, S. *Working with the JavaFX Scene Graph*, 2013. Dostupné z: http://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm.

[11] ORACLE. *JavaFX Developer Home* [online]. [cit. 24.6.2014]. Dostupné z: `http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html`.

[12] ORACLE. *Interface Serializable* [online]. [cit. 18.6.2014]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html`.

[13] ORACLE. *Oracle Semantic Technologies Downloads* [online]. 2013. [cit. 18.6.2014]. Dostupné z: `http://www.oracle.com/technetwork/database-options/spatialandgraph/downloads/index-156999.html`.

[14] *Oracle SQL Developer* [online]. [cit. 24.6.2014]. Dostupné z: `http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html`.

[15] PER CHRISTENSSON, T. *Applet* [online]. 2012. [cit. 18.6.2014]. Dostupné z: `http://www.techterms.com/definition/applet`.

[16] *phpMyAdmin* [online]. [cit. 24.6.2014]. Dostupné z: `http://www.phpmyadmin.net/`.

[17] *Qt Jambi - Qt for Java* [online]. [cit. 24.6.2014]. Dostupné z: `http://qt-jambi.org/`.

[18] *openRDF.com ...home of Sesame* [online]. [cit. 24.6.2014]. Dostupné z: `http://www.openrdf.org/`.

[19] *Package javax.swing* [online]. [cit. 24.6.2014]. Dostupné z: `http://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html`.

[20] *SWT: The Standard Widget Toolkit* [online]. [cit. 24.6.2014]. Dostupné z: `http://www.eclipse.org/swt/`.

[21] VCELAK, P. – KRATOCHVIL, M. – KLECKOVA, J. Collaboration and Research Information System Used for Sustainable Long-Term Research. In *2013 Fourth World Congress on Software Engineering*, s. 307–310, December 2013. doi: 10.1109/WCSE.2013.56. ISBN 978-1-4799-2883-5.

[22] W3. *N-Triples*, 2001. Dostupné z: `http://www.w3.org/2001/sw/RDFCore/ntriples/`. W3C RDF Core WG Internal Working Draft.

[23] *SPARQL 1.1 Protocol*. W3C, 2013. Dostupné z: http://www.w3.org/TR/sparql11-protocol/. W3C Recommendation 21 March 2013.

[24] *OWL Web Ontology Language - Overview*. W3C, 2004. Dostupné z: http://www.w3.org/TR/owl-features/. W3C Recommendation 10 February 2004.

[25] W3C. *RDF - Semantic Web Standards* [online]. 2014. [cit. 12.6.2014]. Dostupné z: http://www.w3.org/RDF/.

[26] *RDF 1.1 Primer*. W3C, 2014. Dostupné z: http://www.w3.org/TR/rdf11-primer/. W3C Working Group Note 25 February 2014.

[27] *RDF Schema 1.1*. W3C, 2014. Dostupné z: http://www.w3.org/TR/rdf-schema/. W3C Recommendation 25 February 2014.

[28] *RDF 1.1 XML Syntax*. W3C, 2014. Dostupné z: http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/. W3C Recommendation 25 February 2014.

[29] *Resource Description Framework (RDF) Model and Syntax Specification*. W3C, 1999. Dostupné z: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. W3C Recommendation 22 February 1999.

[30] *SPARQL Query Language for RDF*. W3C, 2008. Dostupné z: http://www.w3.org/TR/rdf-sparql-query/. W3C Recommendation 15 January 2008.

[31] *SPARQL 1.1 Query Language*. W3C, 2013. Dostupné z: http://www.w3.org/TR/sparql11-query/. W3C Recommendation 21 March 2013.

[32] WALSH, N. *A Technical Introduction to XML* [online]. 1998. [cit. 24.6.2014]. Dostupné z: http://www.xml.com/pub/a/98/10/guide0.html.

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Příloha A

Uživatelská příručka

A.1 Požadavky

Jediným požadavkem na systém, kde bude aplikace provozována, je přítomnost běhového prostředí Oracle **Java Runtime Environment ve verzi 8** nebo kompatibilního.

Pro práci s lokálním úložištěm může být navíc vyžadováno vyšší množství operační paměti. Doporučeny jsou alespoň **4 GiB RAM**.

A.2 Sestavení

Doporučeným způsobem je sestavení aplikace pomocí nástroje Apache Maven.[1] Do jeho repozitáře je v první řadě nutné doinstalovat knihovny:

- `ojdbc6-11.2.0.4.jar`
- `orai18n-11.2.0.4.jar`
- `sdordfclient-11.2.0.4.jar`

Ty se nachází v adresáři se zdrojovým kódem aplikace (tam, kde se nachází soubor `pom.xml`), resp. jeho podadresáři `lib`. Instalaci je nutné provést zadáním těchto příkazů (vždy jeden příkaz, pouze na více řádek):

```
mvn install:install-file -Dfile=ojdbc6-11.2.0.4.jar
    -DgroupId=com.oracle -DartifactId=ojdbc6
    -Dversion=11.2.0.4 -Dpackaging=jar
```

[1] `http://maven.apache.org/`

```
mvn install:install-file -Dfile=orai18n-11.2.0.4.jar
    -DgroupId=com.oracle -DartifactId=orai18n
    -Dversion=11.2.0.4 -Dpackaging=jar
```

```
mvn install:install-file -Dfile=sdordfclient-11.2.0.4.jar
    -DgroupId=com.oracle -DartifactId=sdordfclient
    -Dversion=12.1.0.1 -Dpackaging=jar
```

Aplikaci lze poté sestavit z adresáře se zdrojovými kódy zadáním:

```
mvn clean install
```

Tím bude vytvořen adresář `target` a v něm spustitelný soubor s koncovkou `jar`, např `sparkle-1.0-r201406231040.jar`. Kromě dalších podadresářů se zde nachází také adresář `lib` obsahující knihovny nutné ke spuštění aplikace.

A.3 Instalace a spuštění

Sestavenou aplikaci není třeba specifickým způsobem instalovat. V případě potřeby přesunout spustitelný soubor mimo původní adresář `target` (viz A.2), je třeba spolu s ním přesunout i adresář `lib`, bez kterého nepůjde aplikace spustit.

Aplikaci lze spustit z příkazové řádky příkazem:

```
java -jar sparkle.jar
```

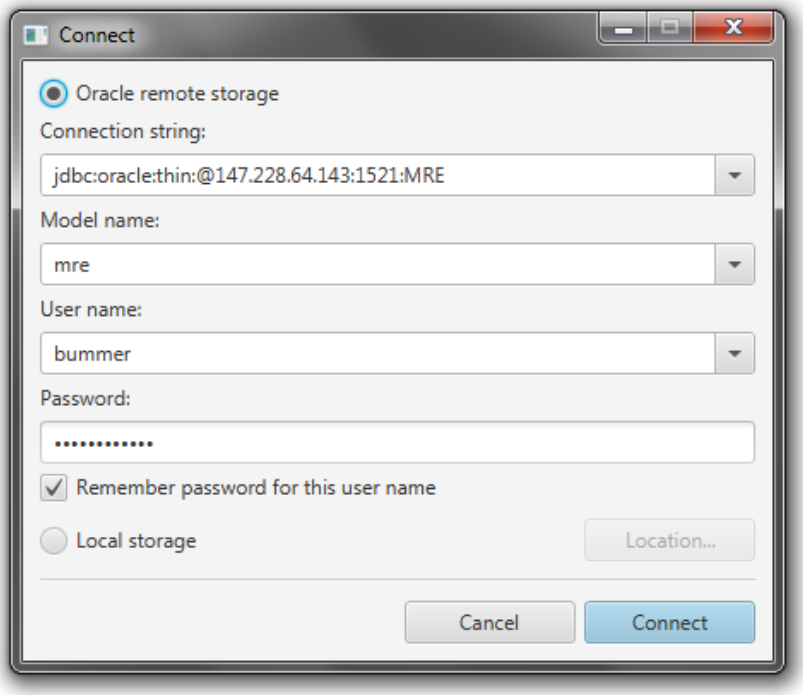
Za `sparkle.jar` je třeba dosadit skutečný název spustitelného souboru. V případě potřeby je možné doplnit parametr navyšující množství operační paměti, které může aplikace využívat:

```
java -jar sparkle.jar -Xmx4g
```

V tomto případě je maximální množství paměti stanoveno na 4 GiB tak, jak doporučuje kapitola A.1.

A.4 Obsluha aplikace

Po spuštění aplikace je jako první zobrazen dialog vyzývající k připojení k RDF úložišti (obr. A.1).



Obrázek A.1: Dialog pro připojení k úložišti.

A.4.1 Připojení k úložišti

Popis dialogu na obr. A.1:

- **Oracle remote storage** – Volba pro připojení k úložišti Oracle. K dokončení operace je nutné zadat několik údajů:
  - **Connection string** – Řetězec pro připojení k úložišti, podobný tomu na obr A.1.
  - **Model name** – Název modelu/grafu v úložišti, ke kterému se aplikace připojí.

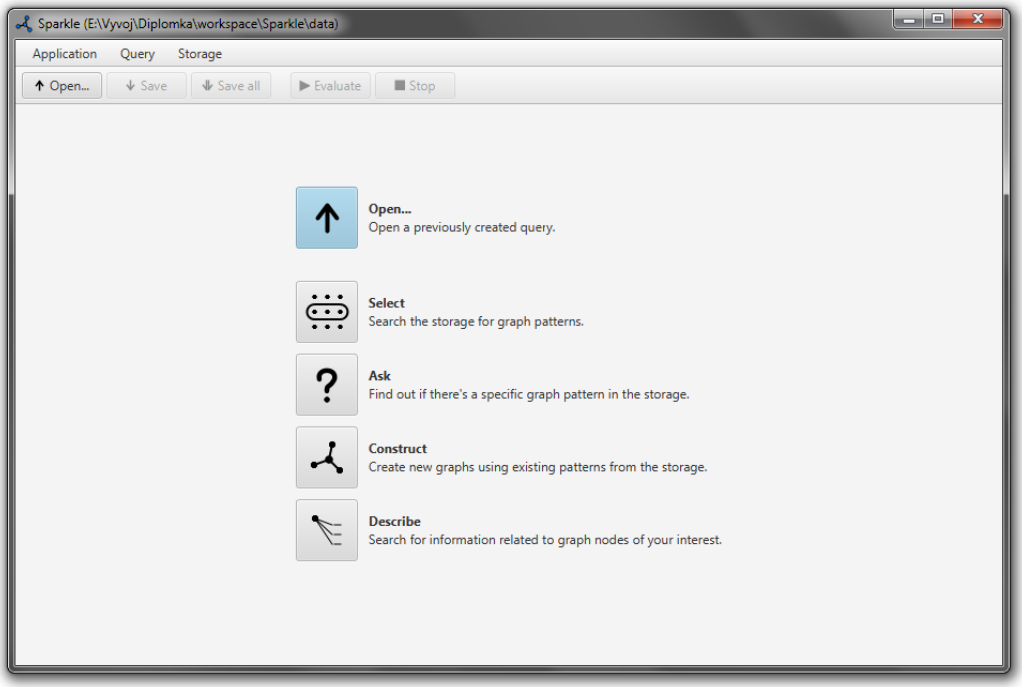
  - **User name** – Uživatelské jméno.
  - **Password** – Heslo. Zaškrtnutím „Remember password for this user name“ lze zajistit, aby si aplikace zadanou kombinaci uživatelského jména a hesla pamatovala a nebylo tak nutné jej při každém připojení zadávat. Heslo je ukládáno **v nešifrované podobě**.

- **Local storage** – Volba pro připojení k místnímu úložišti. Umístění úložiště si volí uživatel sám kliknutím na tlačítko „Location...“ a výběrem adresáře na svém zařízení.

Připojení k úložišti se provede stiskem tlačítka „Connect“.

A.4.2 Hlavní okno aplikace

Po připojení k úložišti se objeví hlavní okno aplikace tak, jak je na obrázku A.2.

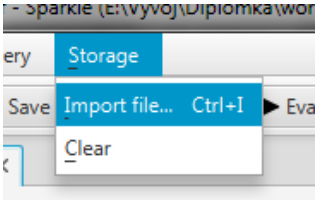


Obrázek A.2: Hlavní okno aplikace.

Uživateli je nabídnuta volba založení jednoho z druhů dotazů, případně otevření dotazu již existujícího. Nový dotaz lze vytvořit také prostřednictvím menu „Query→New“ umístěného v horní části okna aplikace.

A.4.3 Operace nad úložištěm

Do připojeného úložiště je možné nahrávat soubory ve formátech RDF/XML, Notation 3, Turtle a N-Triples. K importu jednoho nebo více souborů slouží volba „Storage→Import file...“ přístupná z menu hlavního okna aplikace (obr. A.3).



Obrázek A.3: Volba pro import souborů do úložiště.

A.4.4 Tvorba dotazu

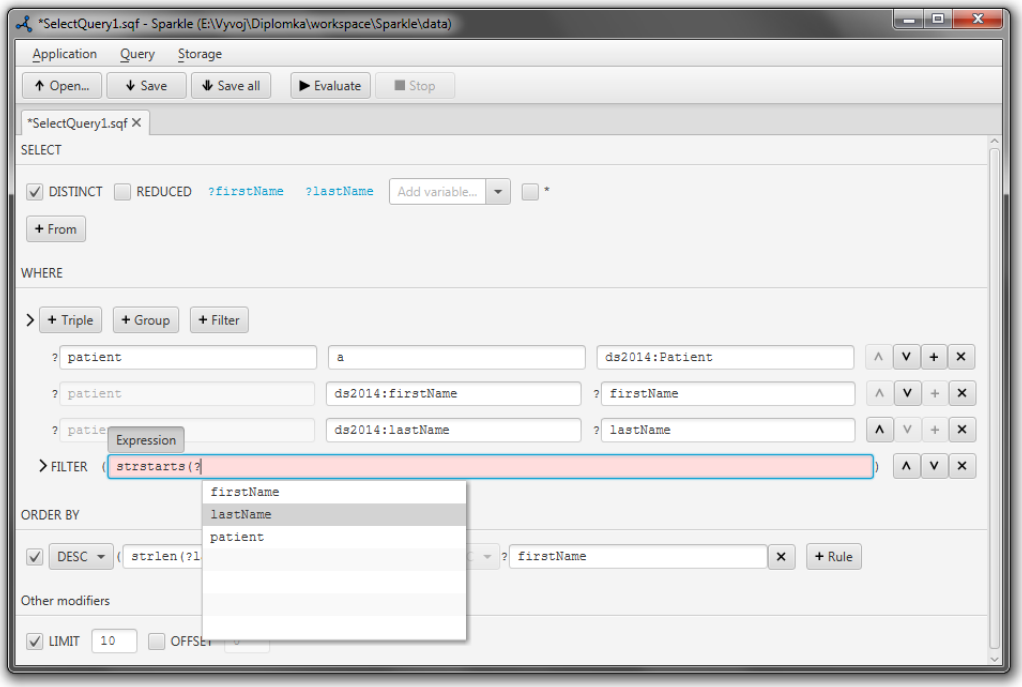
Obrázek A.4 ukazuje okno aplikace s rozpracovaným dotazem SELECT.[2] Následující text se pokusí stručně přiblížit uživateli základní úkony jeho tvorby.

Proměnné

Proměnnou lze do klauzule `SELECT` přidat napsáním jejího názvu (bez prefixu ‚`?`‘) do editačního pole na obr. A.5 a následným stisknutím klávesy Enter.

Proměnná se následně objeví vlevo od editačního pole. Pro její odstranění je možné na ni kliknout a v kontextovém menu vybrat položku „Remove“, případně „Remove all“ pro odstranění všech proměnných (viz obr. A.6).

[2]Pro popis smyslu jednotlivých jeho částí viz kapitola 4, případně specifikace jazyka SPARQL [30] a [31].



Obrázek A.4: Dotaz SELECT.

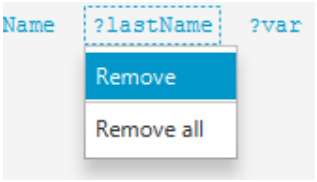
Obrázek A.5: Editační pole pro vložení proměnné.

Přidání trojice

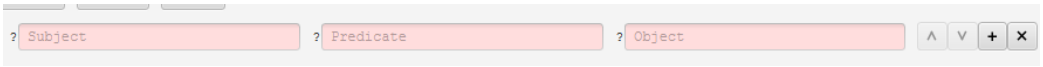
Trojici lze do klauzule `WHERE` či `CONSTRUCT` přidat stiskem tlačítka „+ Triple“. Nově vytvořenou trojici lze spatřit na obrázku A.7.

Červené pozadí polí znamená, že jsou nesprávně vyplněna. Před každým z nich lze spatřit znak ‚?‘ značící, že pole v tuto chvíli reprezentuje proměnnou (její název je třeba zadávat bez otazníku). Typ elementu, který bude pole reprezentovat, lze nastavit tlačítky, která se zobrazí po zaměření pole (třeba tím, že do něj uživatel klikne) – viz obr. A.8.

Kromě proměnné může pole v trojici zastupovat také plný identifikátor zdroje (typ „IRI“) nebo jeho zkrácenou verzi (typ „Prefixed name“). Je-li pole nastaveno na typ IRI, zobrazují se kolem něj špičaté závorky (IRI je třeba zadávat bez těchto závorek).



Obrázek A.6: Odebrání proměnné.



Obrázek A.7: Prázdná trojice.

Pole na pozici prefikátu lze nastavit také na typ „a“. V takovém případě bude pouze pro čtení a bude obsahovat pouze klíčové slovo `a` sloužící jako zkratka pro `rdf:type`.

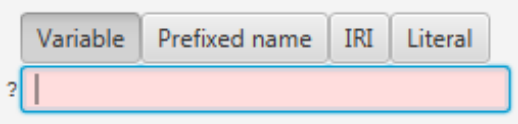
Posledním typem využitelným v trojici (na pozici objektu) je „Literal“. V takovém pole očekává prostý text bez uvozovek (jsou opět zobrazeny mimo pole). Literálu lze navíc nastavit datový typ nebo jazyk. Obrázek A.9 ukazuje tři pole pro zadání literálů. První shora má nastaveno datový typ `xsd:int` (naznačeno znaky `^^` stejně jako u textového zápisu SPARQL), u druhého je uvedeno, že je zadaný řetězec v anglickém jazyce (naznačeno znakem `@`) a v třetím případě jde o prostý řetězec.

Filtry

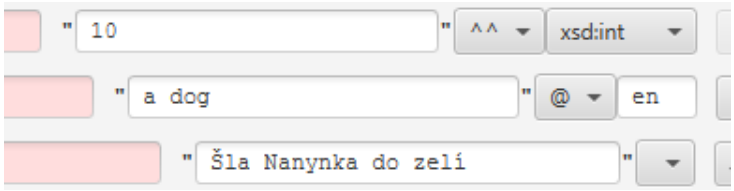
Klauzuli `FILTER` lze do `WHERE` přidat stiskem tlačítka `+ Filter`. Příslušné editační pole očekává výraz, např. jako na obr. A.10. Pole s výrazem je vždy uvozeno závorkami, není tedy nutné je zadávat ručně.

Skupiny

Skupinu trojic (viz kapitola 4.7) lze do do `WHERE` přidat stiskem tlačítka `+ Group`. Na obrázku A.11 je vidět trojice a pod ní dvě vnořené skupiny (každá s jednou trojicí uvnitř) s možností každé z nich přiřadit jeden ze skupiny modifikátorů.



Obrázek A.8: Tlačítka pro výběr typu pole.



Obrázek A.9: Pole pro zadání literálů.

Pravidla řazení výsledků

Ve spodní části dotazů SELECT, CONSTRUCT a DESCRIBE se nachází klauzule `ORDER BY`. Pravidlo pro řazení výsledků lze přidat kliknutím na tlačítko „+ Rule“ (viz obr. A.12). Řadit lze podle hodnot proměnných či podle hodnoty výrazu (toto lze opět specifikovat výběrem typu pole, stejně jako v A.4.4) a je možné specifikovat směr (`ASC` či `DESC`, viz kapitola 4.6).

Asistovaná editace

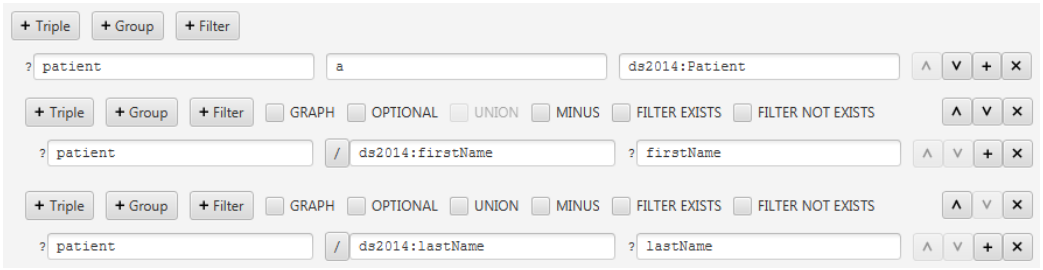
Aplikace se snaží uživateli při vyplňování polí napovídat na základě jejich typu, případně pozice v trojici a textu, který již uživatel napsal. Nápovědu pro zkrácený identifikátor ukazuje obr. A.13.

Seznam s návrhy je vyvolán buďto automaticky (zaměřením pole, pro které je nápověda k dispozici), jako reakce na napsání konkrétního znaku (‚`:`‘ v případě psaní zkáceného IRI nebo ‚`?`‘ při psaní jména proměnné) nebo klávesovou zkratkou (viz tabulka klávesových zkratek A.1).

Aplikace umí uživateli navrhovat:

- Názvy existujících proměnných
- Prefixy pro psaní zkrácených IRI

Obrázek A.10: Pole pro specifikaci filtru.



Obrázek A.11: Skupiny trojic.

- Pojmy ze známých jmenných prostorů při psaní zkrácených IRI (vč. datových typů)
- Funkce (při psaní výrazů)

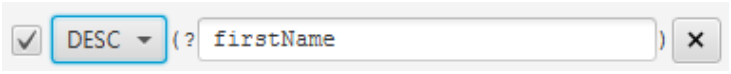
Některé prefixy, pojmy a funkce má aplikace vestavěné. Uživatel má ale možnost doplnit další – viz kapitola A.4.7.

A.4.5 Vyhodnocení dotazu

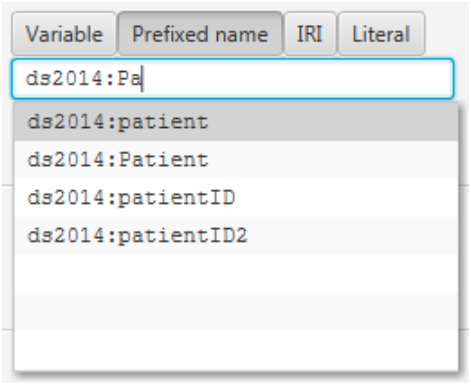
Vyhodnocování hotového dotazu lze spustit tlačítkem „Evaluate“ v horní části okna aplikace. Dojde k otevření a výběru nové záložky sloužící speciálně k tomuto účelu. Že na této záložce probíhá vyhodnocování, lze poznat podle toho, že má jako ikonu animovaný ukazatel průběhu. Zastavit vyhodnocování lze buďto zavřením této záložky nebo stiskem tlačítka „Stop“.

Pro dotaz typu SELECT má výsledek formu tabulky. Tabulka se zobrazí ve chvíli, kdy aplikace obdrží první řádku hodnot (viz obr. A.14). Zobrazují se pouze neprázdné sloupce.

Po obdržení posledního výsledku se aktivuje možnost uložit výsledky do formátu CSV. Volbu uživatel nalezne v kontextovém menu, které lze obvykle zobrazit kliknutím pravým tlačítkem myši na tabulku – viz obr. A.15. Na výběr je ze dvou norem.



Obrázek A.12: Pravidlo pro řazení.



Obrázek A.13: Seznam s návrhy na dokončení.

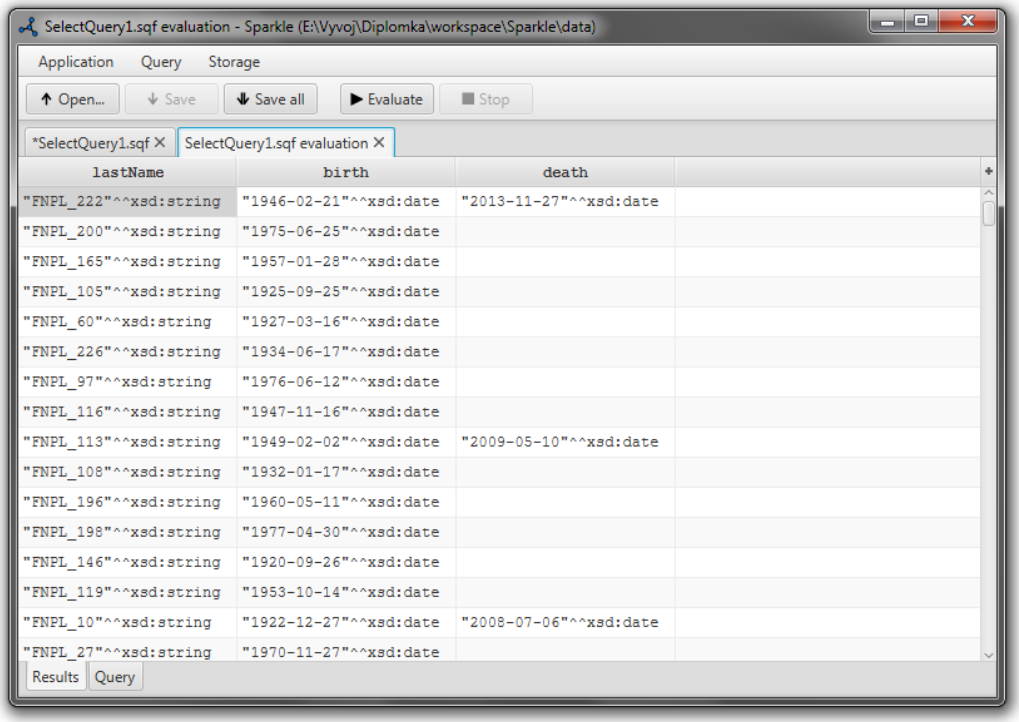
Obsah každé buňky lze zobrazit v samostatném okně poklikáním myší a následně uložit.

Dotaz typu ASK vrací pouze hodnotu ano/ne – viz obr. A.16.

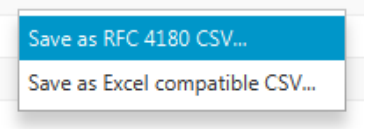
Dotazy typu CONSTRUCT a DESCRIBE vrací výsledek ve formě grafu, který aplikace zobrazuje v jednom z dostupných formátů – RDF/XML, Notation 3, Turtle a N-Triples. V tomto formátu lze výsledek také uložit kliknutím na tlačítko „Save results as...“. Aplikaci zobrazující tento druh výsledku ukazuje obrázek A.17.

Ruční úprava dotazu

Dotaz je možné také ručně upravit a nechat jej vyhodnotit znovu i v případě, že aplikace konstrukce v něm použité nepodporuje. Okno s vygenerovaným a vyhodnocovaným dotazem se nachází na záložce „Query“ u spodního okraje okna (viz obr. A.18).



Obrázek A.14: Výsledek dotazu SELECT.



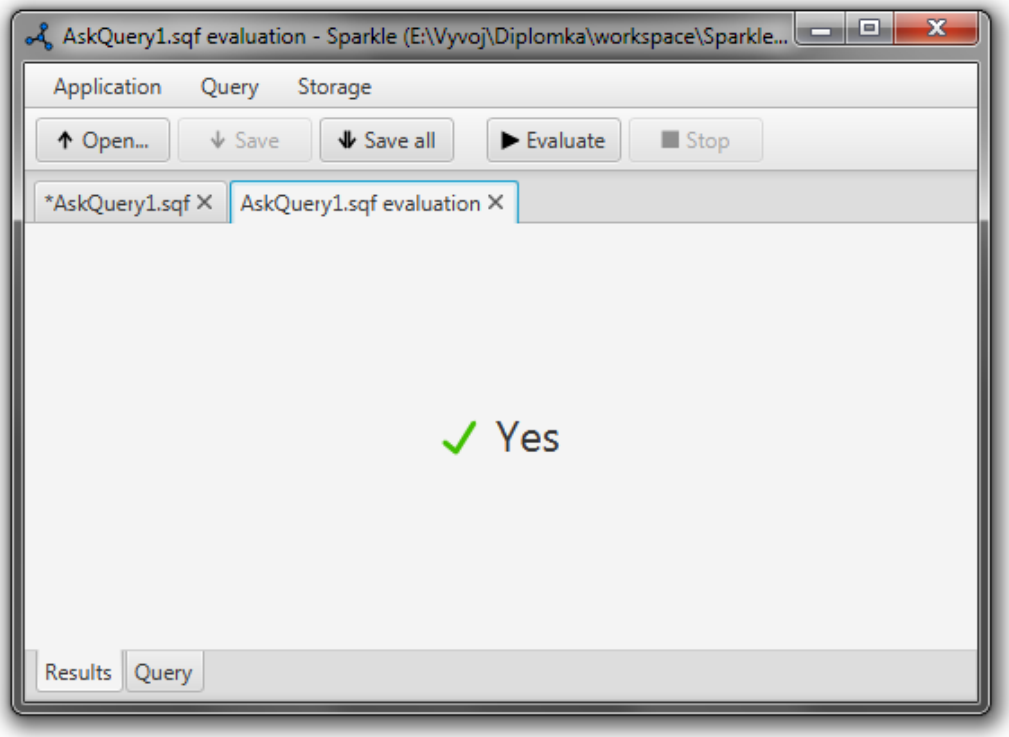
Obrázek A.15: Kontextové menu tabulky s výsledkem SELECT.

A.4.6 Uložení, vyvolání a export

K uložení dotazu slouží tlačítko „Save“ (případně „Save all“ pro uložení všech otevřených souborů), resp. stejně pojmenované volby v menu „Query“. Dotaz je ukládán do souborů s koncovkou `sqf` v proprietárním textovém formátu založeném na XML.

Otevřít takto uložený dotaz lze skrze tlačítko „Open...“ nebo stejně pojmenovanou volbu v menu „Query“.

Otevřený dotaz je možno exportovat ve formátu SPARQL skrze volbu „Query→Export...“. Výsledné soubory mají koncovku `rq`.

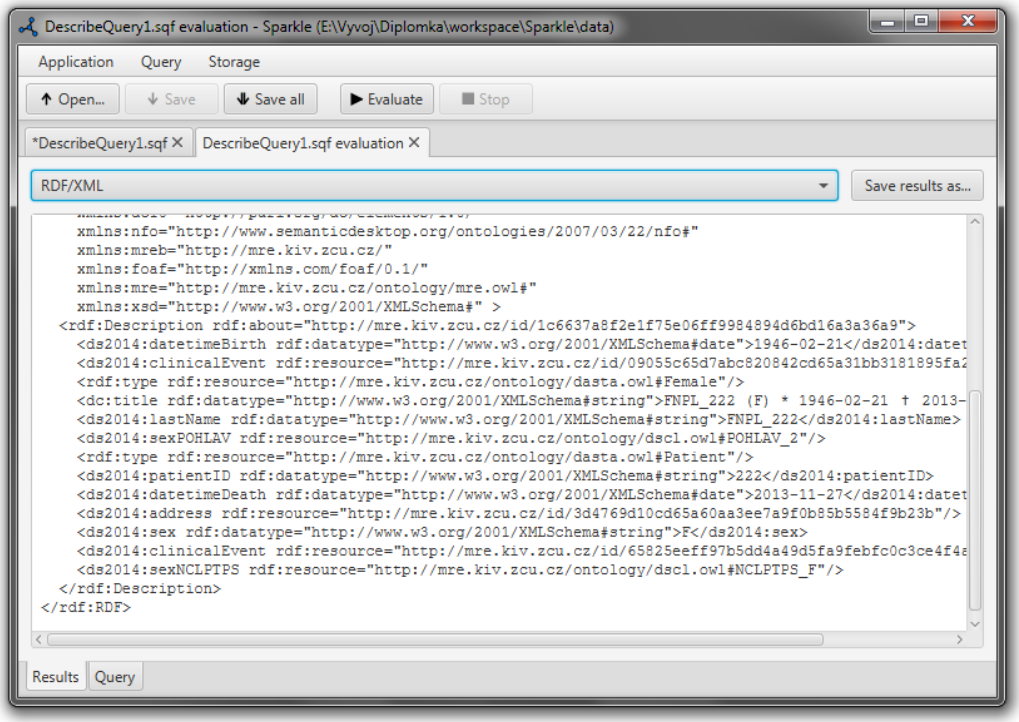


Obrázek A.16: Výsledek dotazu ASK.

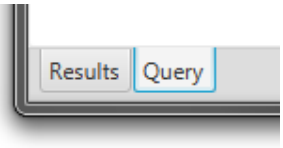
A.4.7 Správa prefixů a pojmů

Jeden ze způsobů, jak aplikace usnadňuje uživateli psaní dotazů, je ten, že se za něj postará o mapování jmenných prostorů na prefixy. Kromě tohoto mapování udržuje ještě také seznamy datových typů, funkcí a ostatních pojmů sloužící mimo jiné jako podpora asistované editace (viz kapitola A.4.4).

Tyto seznamy mapování a pojmů mají dvojí kontext – aplikace a dotaz. Seznamy v kontextu aplikace (označované jako *lokální*) tvoří základ pro seznamy v kontextu dotazu. Vytvoří-li uživatel nový dotaz, vytvoří se pro něj kopie lokálních seznamů a dotaz od této chvíle manipuluje jen s nimi. Různé dotazy tak mohou například používat stejné prefixy pro různé jmenné prostory.



Obrázek A.17: Výsledek dotazu DESCRIBE.



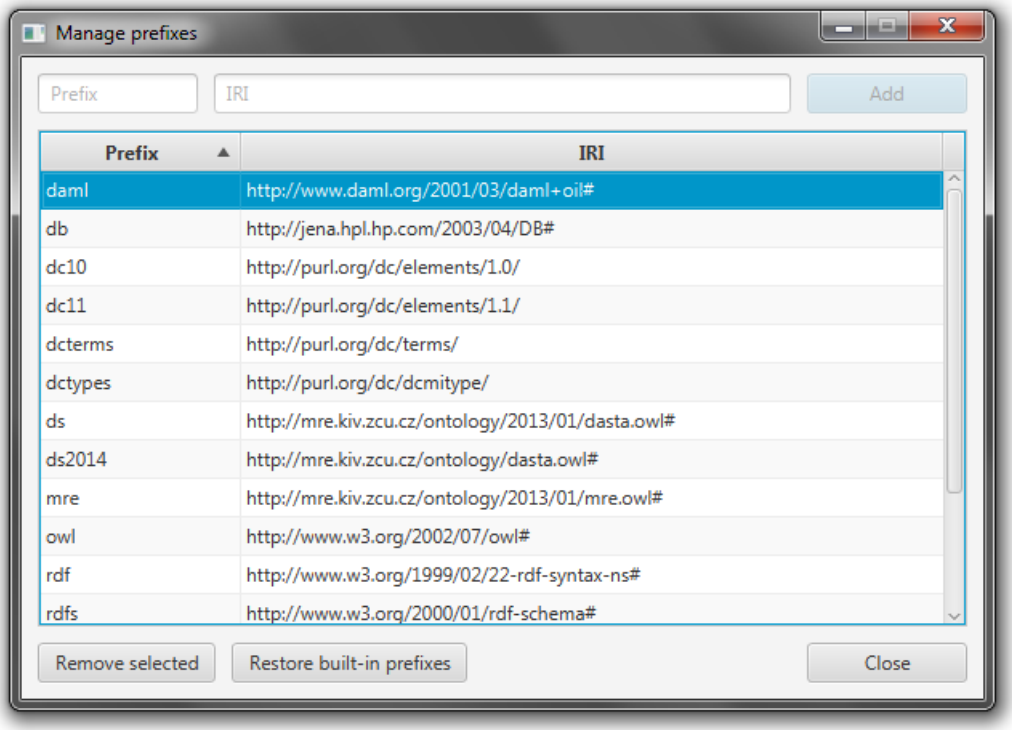
Obrázek A.18: Záložka Query.

Manipulace s prefixy

Okno pro správu prefixů lze vyvolat skrz menu „Application→Manage local prefixes...“, resp. „Query→Manage query prefixes...“. Jak vypadá, je vidět na obr. A.19.

Přidání mapování lze provést vyplněním polí „Prefix“ a „IRI“ a kliknutím na tlačítko „Add“. k odebírání vybraných mapování slouží tlačítko „Remove selected“. Kteroukoliv hodnotu v tabulce lze upravit poklikáním.

Tlačítko „Restore built-in prefixes“ obnoví vestavěná mapování.



Obrázek A.19: Okno pro správu prefixů.

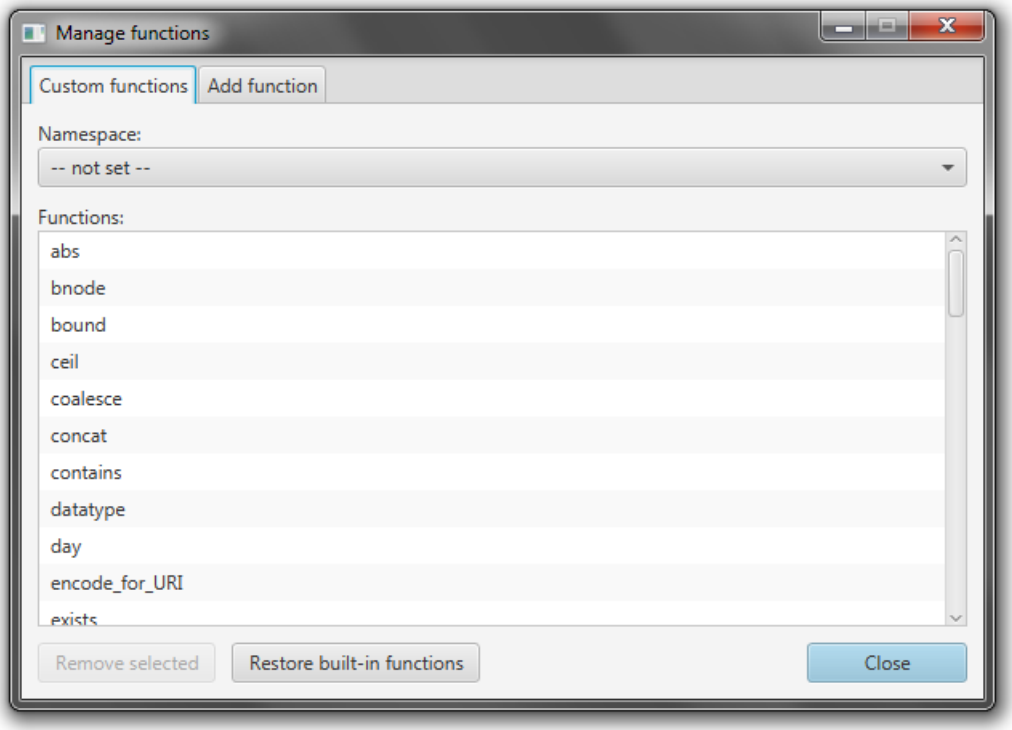
Manipulace s funkcemi a datovými typy

Okno pro správu funkcí lze vyvolat skrz menu „Application→Manage local functions...“, resp. „Query→Manage query functions...“. Jak vypadá, je vidět na obr. A.20. Dialog pro správu datových typů vypadá totožně, liší se pouze popisky. Vyvolat lze skrz menu „Application→Manage local data types...“, resp. „Query→Manage query data types...“.

Roletkové menu „Namespace“ slouží k výběru jmenného prostoru, ze kterého se zobrazují funkce v seznamu pod ním. Vestavěné funkce SPARQL jmenný prostor nemají a nachází se pod položkou „– not set –“.

Vybrané funkce lze ze seznamu odstranit stiskem tlačítka „Remove selected“. K obnovení seznamu vestavěných funkcí slouží tlačítko „Restore built-in functions“.

Přidávat vlastní funkce je možné na záložce „Add function“, jak ukazuje obr. A.21.



Obrázek A.20: Okno pro správu funkcí.

K přidání funkce je třeba znát její název a jmenný prostor, kam spadá. Pokud již aplikace příslušný jmenný prostor zná, lze jej vybrat v roletkovém menu „Namespace“. Pokud se zde nenachází, lze jeho IRI zadat do pole „Custom namespace IRI“ pod ním.

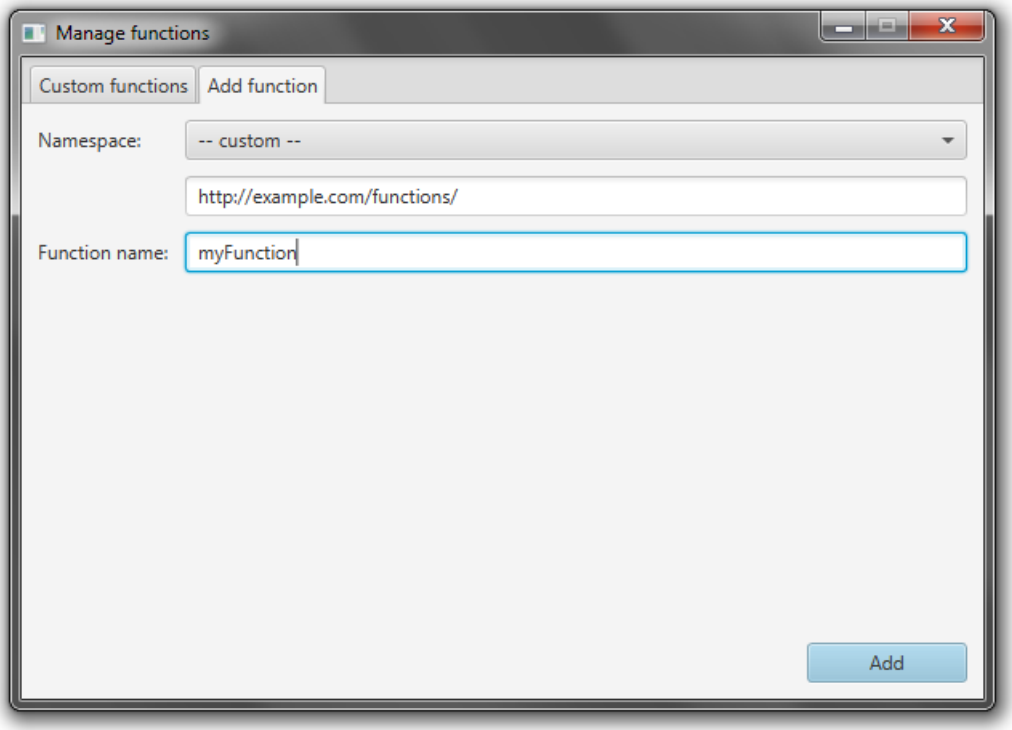
Poslední pole slouží k vyplnění názvu funkce, kterou lze posléze přidat kliknutím na tlačítko „Add“.

Datové typy lze přidávat či odebírat naprosto totožným způsobem.

Manipulace s ostatními pojmy

Okno pro správu ostatních pojmů lze vyvolat volbou „Application→Manage local resources...“, resp. „Query→Manage query resources...“. Jak vypadá, je vidět na obr. A.22.

Roletkové menu „Namespace“ slouží k výběru jmenného prostoru, ze kte-



Obrázek A.21: Formulář pro přidání nové funkce.

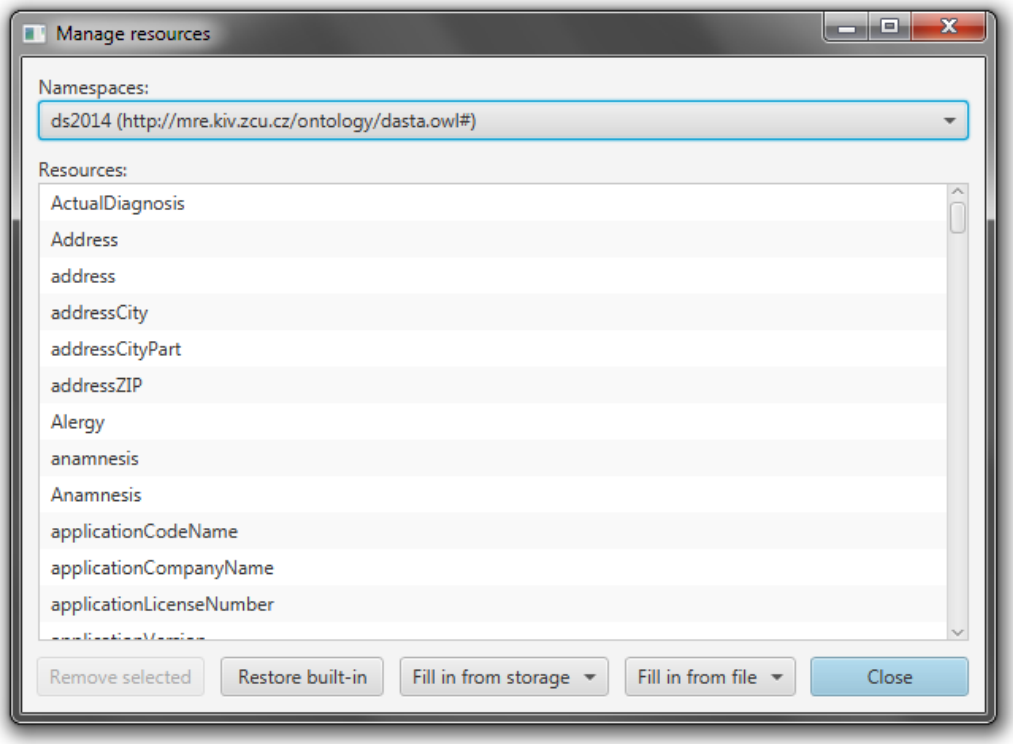
rého se zobrazují pojmy v seznamu pod ním.

Pojmy lze odebírat prostřednictvím tlačítka „Remove selected“. Tlačítko „Restore built-in“ vrátí na seznam vestavěné pojmy, které byly dříve smazány.

Pojmy nelze přidávat ručně. Namísto toho je nutné je importovat z RDF souborů v jednom z podporovaných formátů (RDF/XML, Notation 3, Turtle a N-Triples) nebo z právě připojeného úložiště. Importují se vždy pouze pojmy spadající do právě vybraného jmenného prostoru. Pokud se požadovaný jmenný prostor v seznamu nenachází, je to proto, že jej aplikace dosud nezná a je třeba jej přidat prostřednictvím dialogu pro správu mapování prefixů.

Pro import slouží rozbalovací menu „Fill in from file“, resp. „Fill in from storage“. Menu obsahuje dvě položky: „Include individuals“ a „Exclude individuals“. První zmíněná zahrne do importu všechny pojmy z vybraného jmenného prostoru, zatímco druhá filtruje pojmy třídy `owl:NamedIndividual`.

Import pojmů je náročná operace a obzvlášť import z úložiště může trvat



Obrázek A.22: Okno pro správu ostatních pojmů.

velice dlouho.

Klávesová zkratka	Funkce
Ctrl+T	Vložení trojice
Ctrl+G	Vložení skupiny
Ctrl+F	Vložení filtru
Ctrl+A	Vložení proměnné z právě editovaného pole do klauzule `SELECT` či `DESCRIBE`
Enter	Posun z právě editovaného pole trojice do dalšího pole v pořadí
Alt+←	Změna typu právě editovaného pole (přepnutí na typ vlevo)
Alt+→	Změna typu právě editovaného pole (přepnutí na typ vpravo)
Ctrl+Mezerník	Vyvolání nabídky asistované editace

Tabulka A.1: Zkratky při editaci dotazu.

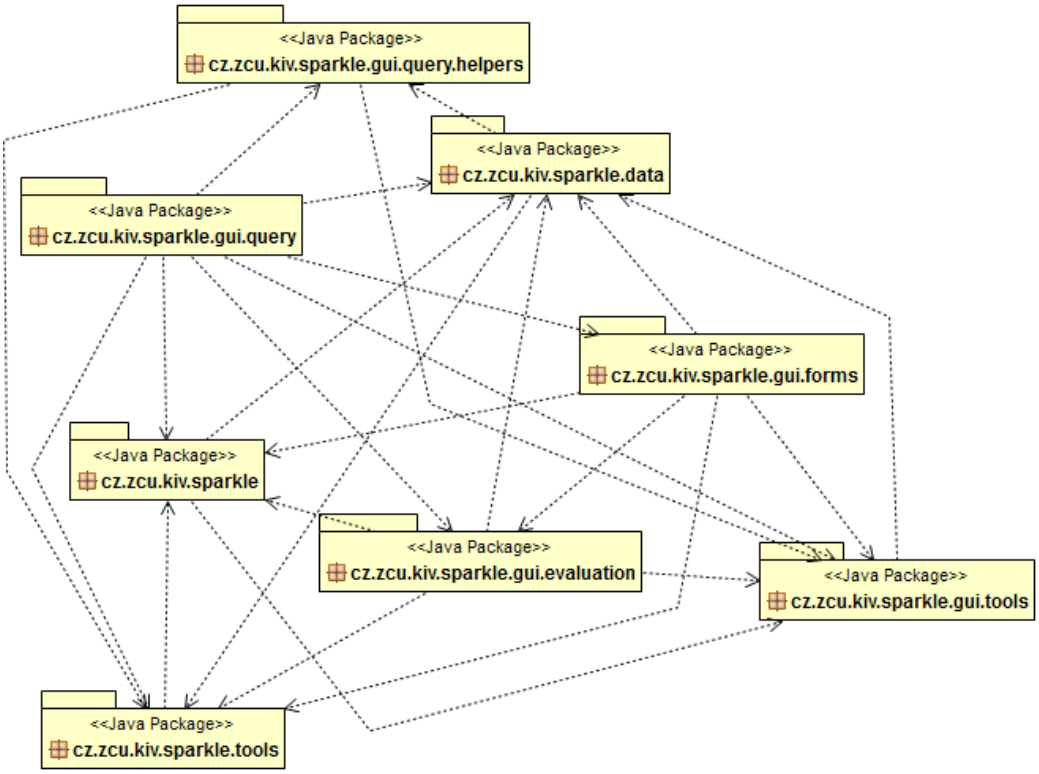
Klávesová zkratka	Funkce
Ctrl+O	Otevření souboru
Ctrl+S	Uložení právě editovaného souboru
Ctrl+Shift+S	Uložení všech otevřených souborů
Ctrl+X	Export editovaného dotazu
Alt+Enter	Vyhodnocení editovaného dotazu
Ctrl+I	Import souboru do úložiště
Ctrl+Alt+P	Správa mapování prefixů v kontextu aplikace
Ctrl+Alt+Shift+P	Správa mapování prefixů v kontextu dotazu
Ctrl+Alt+R	Správa seznamu pojmů v kontextu aplikace
Ctrl+Alt+Shift+R	Správa seznamu pojmů v kontextu dotazu
Ctrl+Alt+F	Správa seznamu funkcí v kontextu aplikace
Ctrl+Alt+Shift+F	Správa seznamu funkcí v kontextu dotazu
Ctrl+Alt+D	Správa seznamu datových typů v kontextu aplikace
Ctrl+Alt+Shift+D	Správa seznamu datových typů v kontextu dotazu

Tabulka A.2: Obecné zkratky.

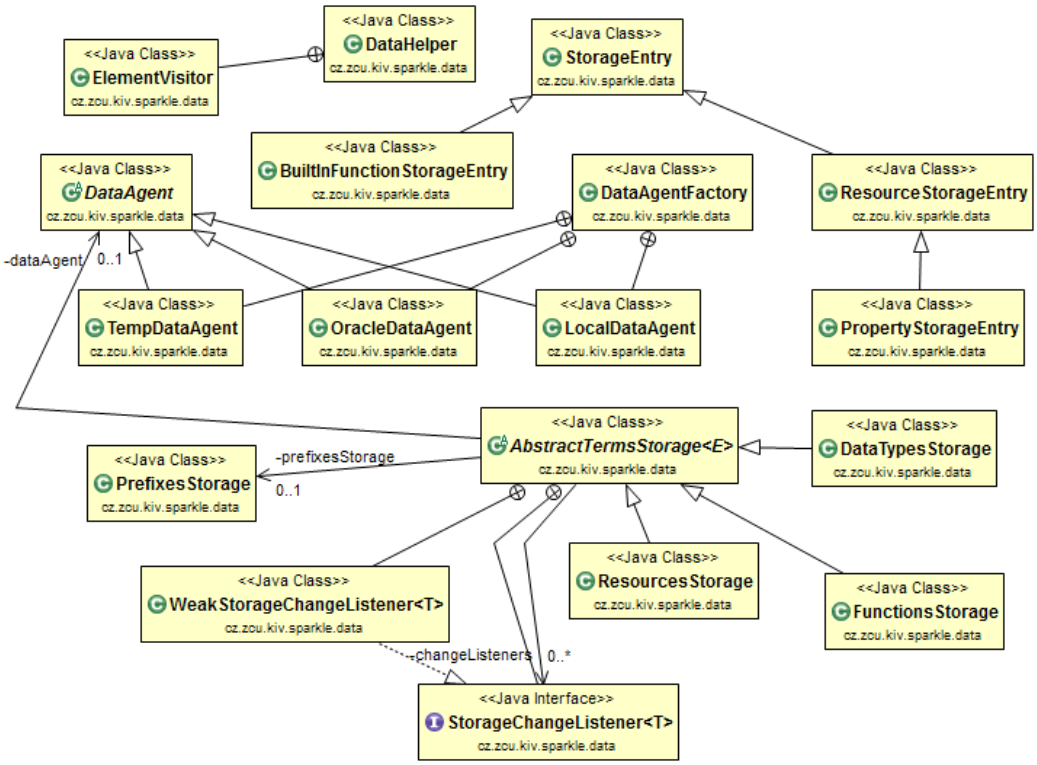
Příloha B

UML diagramy

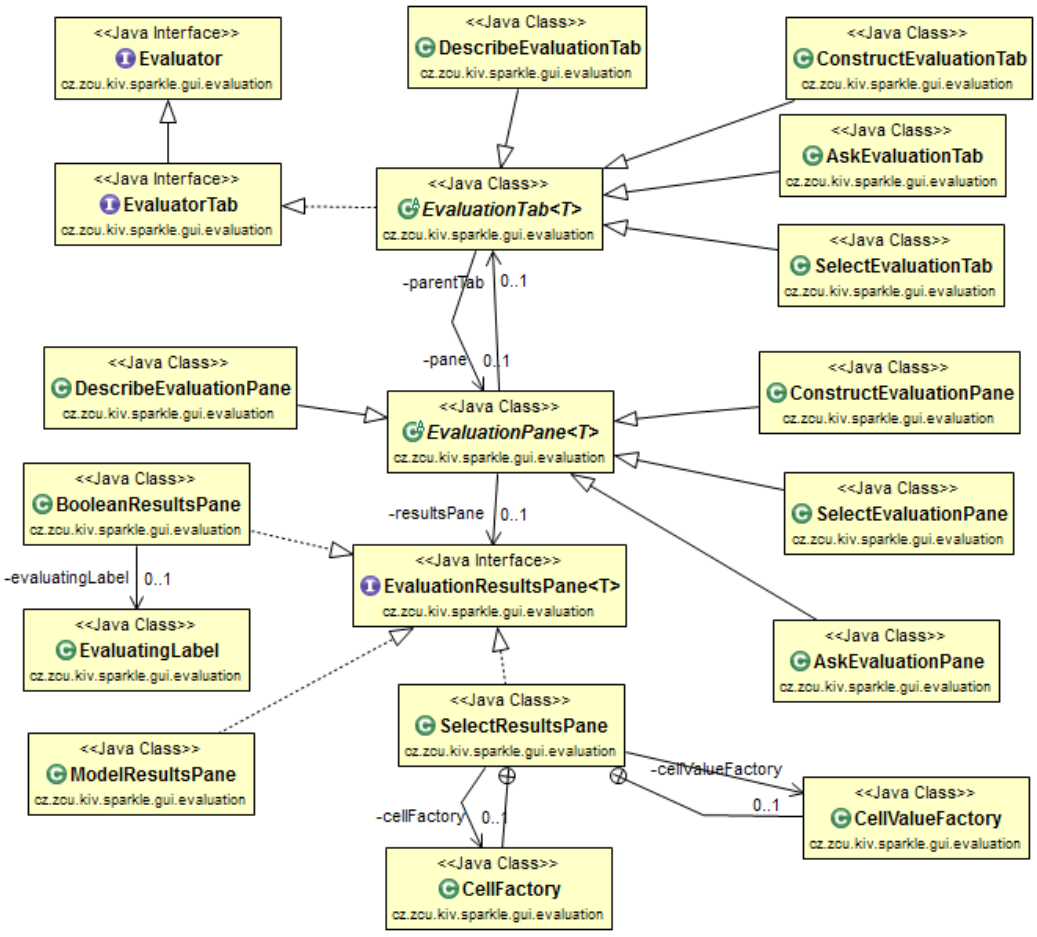
Pro úplný diagram tříd viz přiložené CD.



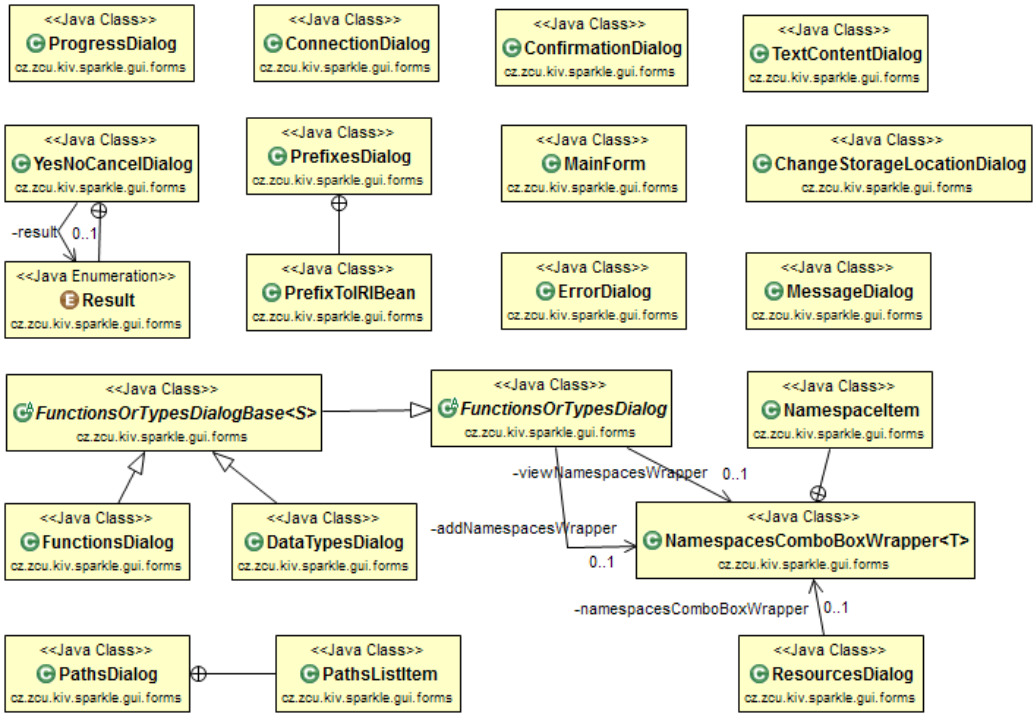
Obrázek B.1: UML diagram balíků.



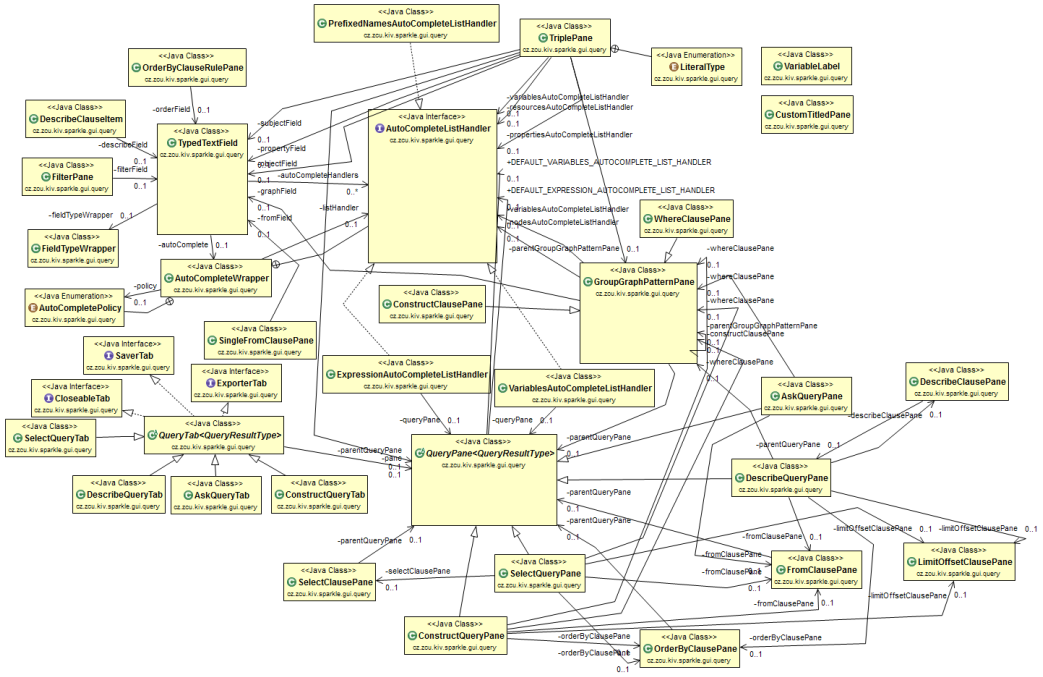
Obrázek B.2: UML diagram balíku „data“.



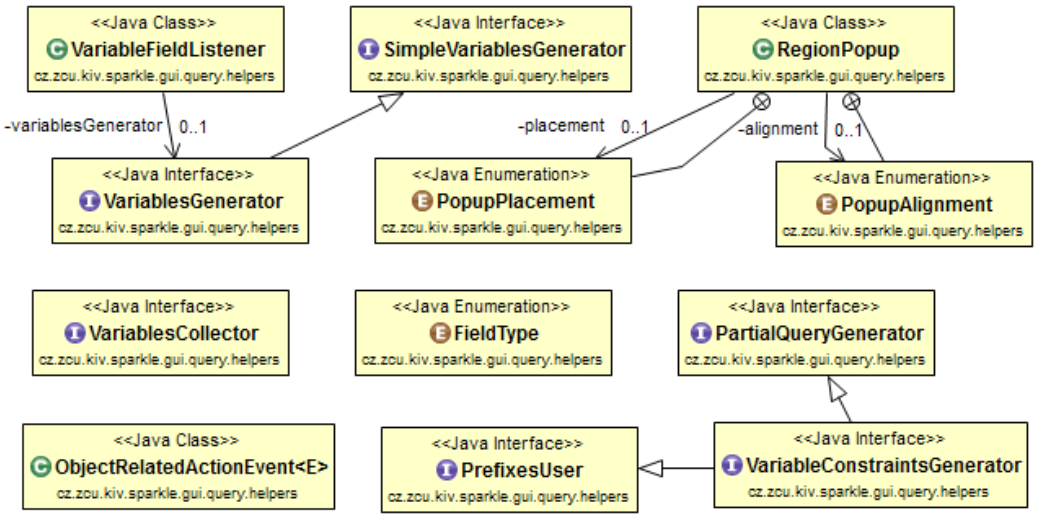
Obrázek B.3: UML diagram balíku „gui.evaluation“.



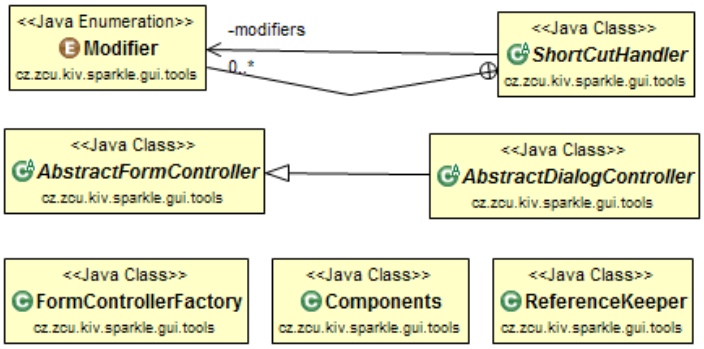
Obrázek B.4: UML diagram balíku „gui.forms“.



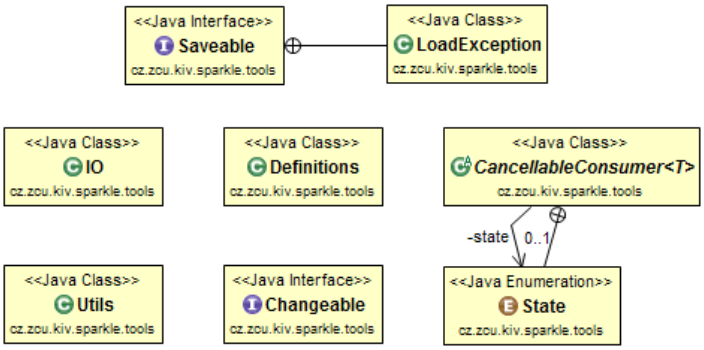
Obrázek B.5: UML diagram balíku „gui.query“.



Obrázek B.6: UML diagram balíku „gui.query.helpers“.



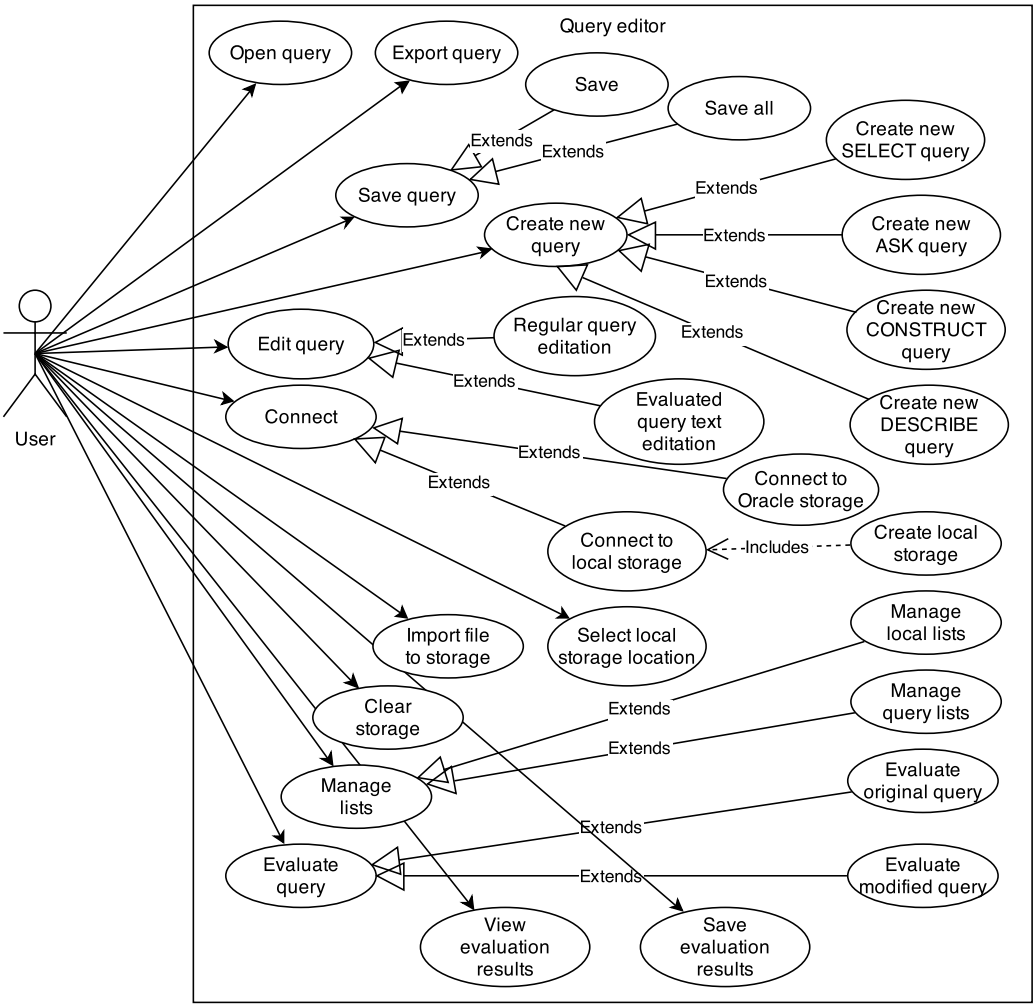
Obrázek B.7: UML diagram balíku „gui.tools“.



Obrázek B.8: UML diagram balíku „tools“.

Příloha C

Diagram užití



Obrázek C.1: Diagram užití aplikace.