VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VIZUALIZACE RDF DAT VE WEBOVÝCH PROHLÍŽEČÍCH

RDF DATA VIZUALIZATON IN WEB BROWSERS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **KRISTIÁN ŠKROBÁNEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Škrobánek Kristián, Bc.**
Program: Informační technologie
Obor: Informační systémy
Název: **Vizualizace RDF dat ve webových prohlížečích**
RDF Data Visualization in Web Browsers
Kategorie: Web
Zadání:

1. Seznamte se s problematikou grafové reprezentace dat v databázích se zaměřením na datový model RDF.
2. Prostudujte jazyk SVG a jeho podporu v hlavních webových prohlížečích. Zmapujte existující knihovny pro vykreslování grafových dat založené na SVG.
3. Po konzultaci s vedoucím navrhněte nástroj pro vykreslení a interaktivní procházení grafových dat na webu.
4. Implementujte navržený nástroj v jazyce JavaScript s využitím vhodných knihoven a rámcových řešení.
5. Proveďte testování navrženého řešení na rozsáhlých datech. Ověřte možnosti webových prohlížečů ohledně maximálního použitelného množství prvků v grafu.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Scalable Vector Graphics, The Wold Wide Web Consortium, http://www.w3.org/Graphics/SVG/
- Lasila, I., Swick, R. R.: Resource Description Framework (RDF) Model and Syntax Specification, W3C Recommendation 22 February 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 22. října 2020

Abstrakt

Tato diplomová práce se zabývá zobrazením dat grafové databáze, kdy data jsou uložena ve formátu RDF. Standardní zobrazení RDF dat pomocí tabulek nenabízí dostatečně uživatelsky užitečný pohled. Jedním cílem práce je zobrazit RDF data v interaktivním grafu, jež je ideální formou prohlížení dat z hlediska přehlednosti a výpovědní hodnoty. Takovýto graf dává přehled nejen o datech samotných, ale i vztazích mezi daty. Dalším cílem práce je otestování schopnosti prohlížečů zobrazovat velké množství dat.

Abstract

This diploma thesis focuses on graph database data visualization, where data is stored in RDF format. Standard visualisation of RDF data in tables does not offer sufficiently usable user view. One of the goals of this work is to show RDF data in interactive graph, which is ideal form of viewing data considering lucidity and information value. The graph gives good view of not only the data itself but also relationships between the data. Another goal is to test ability of browsers to visualize large amounts of data.

Klíčová slova

Grafové databáze, Vizualizace grafů, Webová aplikace, Resource Description Framework, Scalable Vector Graphics, Zobrazení, Interaktivita, Animace, Javascript, Julia, RDF4J

Keywords

Graph databases, Graph visualisation, Web application, Resource Description Framework, Scalable Vector Graphics, View, Interactivity, Animation, Javascript, Julia, RDF4J

Citace

ŠKROBÁNEK, Kristián. *Vizualizace RDF dat ve webových prohlížečích.* Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Vizualizace RDF dat ve webových prohlížečích

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing.Radka Burgeta,Ph.D.

Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Kristián Škrobánek
30. dubna 2021

Poděkování

Chtěl bych poděkovat Ing. Radku Burgetovi, Ph.D. za pomoc, rady a skvělé vedení při vytváření aplikace a psaní této práce.

Obsah

Kapitola 1

Úvod

S vývojem grafových databází reprezentujících data v grafové podobě roste potřeba vhodné vizualizace takto uložených dat v grafové formě. Současné řešení poskytují převážně zobrazení dat v tabulkách, které nejsou adekvátní a dostatečně uživatelsky přívětivé. Vhodným stylem zobrazení grafových databází je právě podoba grafu, která odpovídá jejich konceptu. Framework RDF (Resource description framework) je jedním z možných způsobů reprezentace grafových dat. Tento formát je spojený s konceptem sémantického webu. SVG (Scalable vector graphics) umožňuje zobrazovat vektorovu grafiku na webu, a je proto vhodným kandidátem na zobrazení takových dat. Díky této technologii lze v kombinaci s použitím Javascriptu a technologiemi podporujícími formát RDF vytvořit aplikaci zobrazující data grafových databází v grafické formě.

Množství dat v grafových databázích může být značné a na jejich zobrazení jsou kladeny výkonnostní nároky. Při datových objemech o velikosti statisíců trojic je třeba brát v potaz možnosti prohlížečů a přizpůsobit formu, případně kvantitu zobrazení, dle jejich hranic. Kromě kvantity dat, která jsou vyobrazena pomocí SVG, je třeba dobře uvážit i algoritmy pro následné rozložení dat do prostoru. Tyto algoritmy ovlivňují náročnost zpracování dat Javascriptovou částí.

Tato práce se tedy zabývá vizualizací dat zmíněného formátu RDF, za pomocí webové vizualizační technologie SVG, v interaktivní grafické podobě. Otázkami vizualizace je výkonost, omezení daná prohlížeči, omezení daná technolgií SVG, formát zobrazení, vstup dat k zobrazení a uživatelské požadavky na zobrazení. Kromě návrhu a implementace aplikace je nutné provést průzkum existujících technologií a přístupů k daným otázkám. V závěru práce je pak provedeno nutné testování a analýza výsledků dosažených pomocí zvolených přístupů. Na základě těchto poznatků jsou pak vyvozeny závěry a úpravy pro finální aplikaci pro vizualizaci RDF dat v podobě grafu ve webovém prohlížeči.

V druhé kapitole jsou probrány grafové databáze a sémantický web 2. Následně je probrán nástroj SVG pro zobrazení dat v grafové podobě v prohlížeči 3. Ve čtvrté kapitole je popsán průzkum existujících řešení pro zobrazování RDF dat na webu 4. V páté části se pojednává o technologiích použitých pro vytvoření aplikace 5. Kapitola šest popisuje návrh řešení 6. V kapitole sedm je probrána implementace aplikace 7. V osmé části práce je rozebráno testování aplikace a komentovány výsledky testů, z nichž jsou také vyvozeny poznatky práce 8. V kapitole devět se nachází závěr práce 9.

Kapitola 2

Grafové databáze a sémantický web

V této kapitole je popsán charakter dat, která jsou cílem zobrazování. Nedříve je probrán obecný princip grafových databází, následně jedna z jejich implementací, kterou aplikace využívá. **RDF datový model a sémantický web.**

2.1 Grafové databáze

Grafové databáze jsou databáze, které přistupují ke vztahům dat se stejnou důležitostí jako k datům samotným [11] [7] [19] [13] . Datový model grafových databází má všechny výhody relačního modelu. Při rozvoji inteligence zabudované do databáze samotné umožňující elasticitu pro absorbování příchozích změn. Grafové databáze využívají expandující sémantický model, který mnohem snadněji zakomponovává příchozí požadavky. Místo obav o schéma a strukturu při změnách, lze vytvořit unifikovaný model, který odpovídá jak starým, tak novým požadavkům. Jinak řečeno, grafy nepotřebují schéma a je tedy snadné je rozšiřovat o další typy dat. Grafové databáze lépe pracují se vztahy a propojením dat. Toto v některých případech umožňuje efektivnější a snadnější dotazování, které vede k lepšímu způsobu získávání informací z databází než u relačního modelu. Porozumnění vztahům mezi daty, snížená složitost a rozsah dotazů produkují pozitivní efekt na výkonnost grafových databází. Jedním z těchto znaků je zvýšena rychlost zpracování dotazů a množství dat, jež je zpracováno za časový okamžik. Další z výhod provázejících grafové databáze je akcelerace datového modelingu a transformace dat. Tyto výhody dělají z grafových databází vhodnou alternativu, která může libovolné společnosti využívající databáze pomoci zvýšit efektivitu.

Grafové databáze organizují data v podobě grafu založeném na principu matematické grafové teorie. Graf může být považován za kolekci uzlů reprezentujících entity a hran představujících vztahy mezi entitami. Uzly tedy drží data popisující entity a hrany drží data popisující vztahy.

Mezi dva hlavní zástupce grafových databází patří resource description framework RDF pod dohledem konsorcia W3C a labeled property graph, neboli LPG používaný skupinou Neo4j. Předmětem této práce je model RDF, který je probrán detailněji v další části. LPG je model, ve kterém jsou unikátně identifikovány jak uzly, tak hrany. Hrany tedy mají ID, typ a množinu dvojic klíč-hodnota. To umožňuje LPG definovat některé skutečnosti, které lze v modelu RDF definovat s větší obtížností. Kupříkladu jednoduchý fakt, že pes snědl tři kosti vypadá v modelu LPG o poznání jednodušeji. V LPG v tomto případě můžeme

třikrát zadefinovat vztah *snědl*, jednou entitu *pes* a jednou entitu *kost*. Ve výsledku máme graf o dvou uzlech a třech hranách, u kterého se snadno dotážeme na počet vztahů *snědl* a zjistíme výsledek. RDF oproti tomu neumí unikátně identifikovat vztah stejného typu. Podobný pokus by vyústil v graf s pouze jednou hranou a špatnou kvantifikací vztahu. Dalším problémem s tím spojeným je nemožnost RDF kvalifikovat vztahy. V LPG můžeme jednoduše vyjádřit vlastnost vztahu tím, že mu dáme atributy. V RDF musíme nějakou vlastnost vztahu vyjádřit pomocí modelu, čímž vznikne složitější graf. Tyto rozdíly se v RDF řeší pomocí přemodelování nebo reifikace. Je důležité říci, že každý LPG graf lze převézt pomocí těchto technik na RDF a zase naopak. Vyjadřovací schopnost je stejná. V RDF však, jak již bylo naznačeno, mohou některé skutečnosti vést k mnohem složitějšímu grafu.

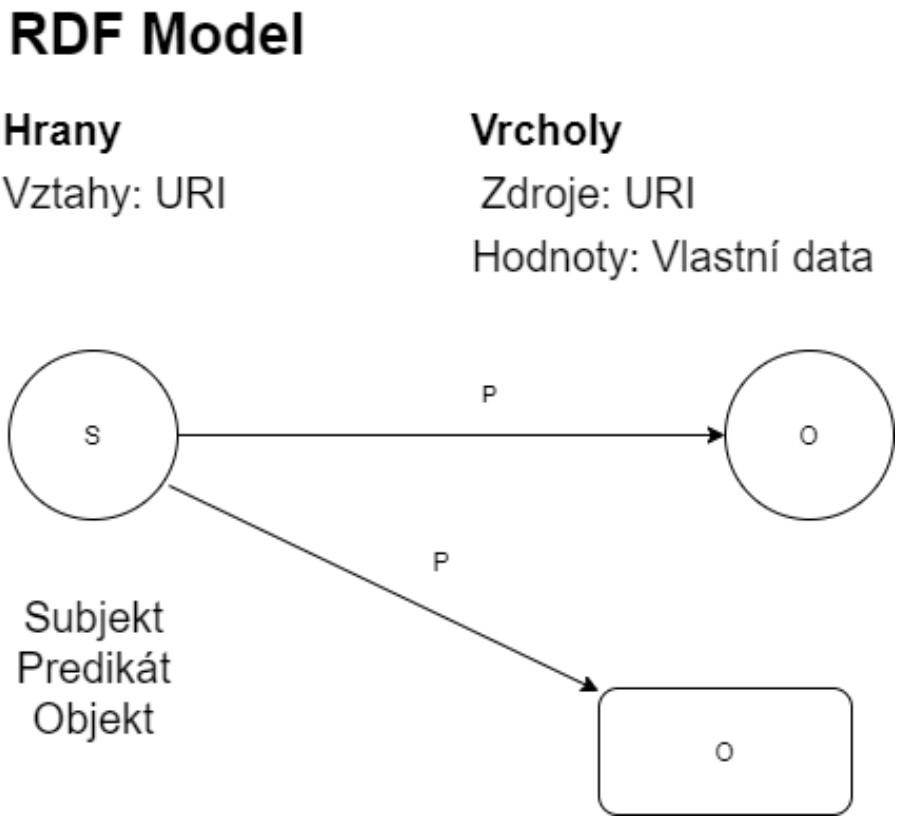
LPG nabízí oproti RDF redukovanou strukturu. RDF naopak rozebírá graf do detailní podoby. Výhodou formátu RDF je také snadné uložení a přesun vztahů neboli RDF je vhodný formát k výměně dat.

2.2 Sémantický web

Ideou sémantického webu je rozšíření existujícího WWW, které dodá softwarovým aplikacím strojově interpretovatelná metadata k již existujícím datům [15]. Toto poskytne možnost pro počítače provádět více smysluplné interpretace dat. Sémantický web indikuje schopnost stroje řešit dobře definovaný problém pomocí provádění dobře definovaných operací na dobře definovaných datech.

2.3 RDF

RDF neboli Resource Description Framework je framework pro reprezentaci informace a její výměnu na webu [21] [24]. RDF usnadňuje spojování dat, a to i v případě, že se liší jejich schéma a podporuje časový rozvoj schémat. Použití RDF modelu umožňuje směšování a šíření strukturovaných a polostrukturovaných dat mezi různými aplikacemi. Struktura formuje orientovaný graf, kde hrany reprezentují pojmenované spojení mezi dvěma zdroji, které jsou zastupovány grafovými uzly. RDF grafy jsou množiny trojic subjekt, predikát, objekt, kde elementy mohou být IRI (Internationalized Resource Identifier), prázdné uzly nebo literály určitého datového typu. Jsou použity k vyjádření popisů zdrojů.



Obrázek 2.1: RDF data model

IRI nebo literál představuje objekt ze skutečného světa. Tyto objekty jsou označovány jako zdroje (resource). Zdroj představován IRI je nazýván referent a zdroj představován literálem je konkrétní hodnota. Literály mají datový typ definující množinu možných hodnot. Určování referentu, na který IRI odkazuje, se řídí následujícími doporučeními. IRI mají globální rozsah, tedy dva různé výskyty IRI odkazují na stejného referenta. Narušení tohoto principu vede k IRI kolizi, případ kdy stejná IRI používaná rozdílnými subjekty odkazuje jiné refrenty. Kupříkladu pro společnost A IRI odkazuje na stránku google, pro společnost B IRI odkazuje prohlížeč googlu. Je konvencí, že majitel IRI říká, k jakému objektu IRI patří. Může k tomu použít dostupný dokument nebo specifikaci. Nejpodstatnější vlastností IRI je jejich dereference a můžou tak posloužit jako počáteční bod pro komunikaci se vzdáleným serverem, který poskytuje RDF data.

RDF slovník je kolekcí IRI určenou k použití v grafech. IRI v takovém slovníku začíná s podřetězcem známým jako namespace IRI. Některé namespace IRI jsou pak spojeny s namespace prefixem neboli zkratkou pro dané IRI. V některých serializačních formátech je běžné nahradit IRI pomocí namespace prefixu ke zlepšení čitelnosti. Nutno říci, že takový zápis není validní IRI, a nelze ho pak použít tam, kde je očekáváno IRI. Namespace IRI a namespace prefix nejsou formální součástí RDF modelu, jedná se pouze o zkratky pro zlepšení čitelnosti.

Namespace prefix	Namespace IRI	RDF vocabulary
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	The RDF built-in vocabulary [RDF11-SCHEMA]
rdfs	http://www.w3.org/2000/01/rdf-schema#	The RDF Schema vocabulary [RDF11-SCHEMA]
xsd	http://www.w3.org/2001/XMLSchema#	The RDF-compatible XSD types

Obrázek 2.2: RDF Namespace prefix a Namespace IRI

RDF datový model není temporální. Poskytuje statické záznamy informace. I přesto mohou RDF grafy poskytovat informaci o událostech a temporálních aspektech entit. Za **rdf** zdroj považujeme persistentní, ale měnitelný zdroj nebo kontejner rdf grafů. RDF zdroj může během času změnit obsah. Snapshot stavu může být znovu zaznamenán jako RDF graf. Shrneme-li tyto informace, víme, že vztah mezi dvěma zdroji může platit v jednom čase a v dalším nemusí. Zdroje mohou změnit svůj obsah, to může vést k jiným **rdf** grafům v jiném čase. Některé rdf zdroje však mohou být neměnitelné snapshoty jiného rdf zdroje a tak zaznamenávat jeho stav v čase [24] (1.5 RDF and Change over Time).

RDF grafy mohou být snadno kombinovány. Často může být požadavek na práci s více grafy při zachování rozdělení jejich obsahů. Tento požadavek splňují RDF datasety. RDF dataset je kolekcí RDF grafů, kdy každý z těchto grafů má v datasetu přiřazeno IRI nebo prázdný uzel. Takové grafy se pak nazývají pojmenované grafy.

RDF graf je množina svých trojic. Logickým významem se jedná o konjunkci svých výroků. Mezi dvěma grafy může být Entailment Equivalence a nad samotným grafem může být Inconsistency. Entailment pro graf A a B znamená, že při jakékoliv konfiguraci reálného světa, která znamená, že platí A, pak platí B. Equivalence znamená, že A i B popisují stejné tvrzení o reálném světě. Equivalence <=> A entails B and B entails A. Incosistency znamená, že graf obsahuje vnitřní kontradikci. Nemůže nastat takový stav světa, který by znamenal, že graf platí.

2.4 RDF syntaxe

RDF dokument je takový dokument, který kóduje RDF graf nebo RDF dataset v konkrétní RDF syntaxi. V následující části jsou popsány hlavní používané syntaxe.

Turtle Turtle neboli terse RDF triple language je konkrétní syntax pro RDF [8]. Dokument v Turtle je zápis RDF grafu v textové podobě. Nejjednodušší zápis je pomocí trojice subjekt, predikát, objekt. Turtle umožňuje využití prefixů, kolekcí, objektových listů a predikátových listů.

```
#Simple triple
<http://example.org/#monkey>
<http://www.perceive.net/schemas/relationship/eats>
<http://example.org/#banana> .


#Predicate list
<http://example.org/#monkey>
<http://www.perceive.net/schemas/relationship/eats>
<http://example.org/#banana>;
<http://xmlns.com/foaf/0.1/livesIn> "Jungle" .
```

```
#Object list
<http://example.org/#monkey> <http://xmlns.com/foaf/0.1/eats> "Banana",
"Orange".

#Prefix
@prefix somePrefix: <http://www.perceive.net/schemas/relationship/> .

<http://example.org/#monkey> somePrefix:eats <http://example.org/#banana> .
```

Výpis 2.1: Turtle příklady

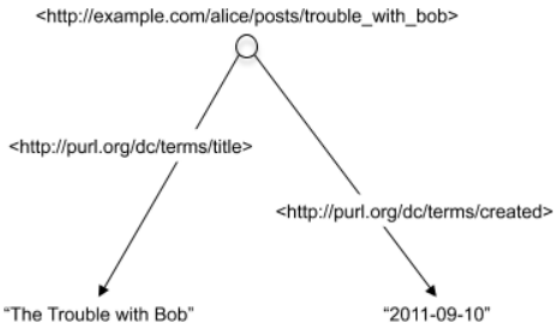
Na ukázce 2.1 je vidět využití predikátového listu při vypisování více vztahů jednoho subjektu. Dále objektového listu při vypisování více objektů na stejném predikátu. A použití prefixu pro zkrácení zápisu.

RDFa 1.1 Primer - Third Edition Technologie přidávající strukturované data přímo do html dokumentů [12]. Umožňuje zpracování html dat počítači a přímo tak podporuje koncept sémantického webu. Pomocí atributů lze klasické html elementy obohatit o dodatečnou informaci, která je čitelná pro počítačové zpracování.

```
<html>
    <head>
      ...
    </head>
    <body>
      ...
      <h2 property="http://purl.org/dc/terms/title">The Trouble with Bob</h2>
      <p>Date:
        <span property="http://purl.org/dc/terms/created">2011-09-10</span>
      </p>
      ...
    </body>
</html>
```

Výpis 2.2: RDFa příklad

Na ukázce 2.2 lze vidět, že přidáním atributu property s hodnotou predikátu uděláme z obyčejného html dokumentu rdf graf. Vizualizaci tohoto grafu je možné vidět na obrázku 2.3.



Obrázek 2.3: RDF graf z html dokumentu za použití RDFa [12]

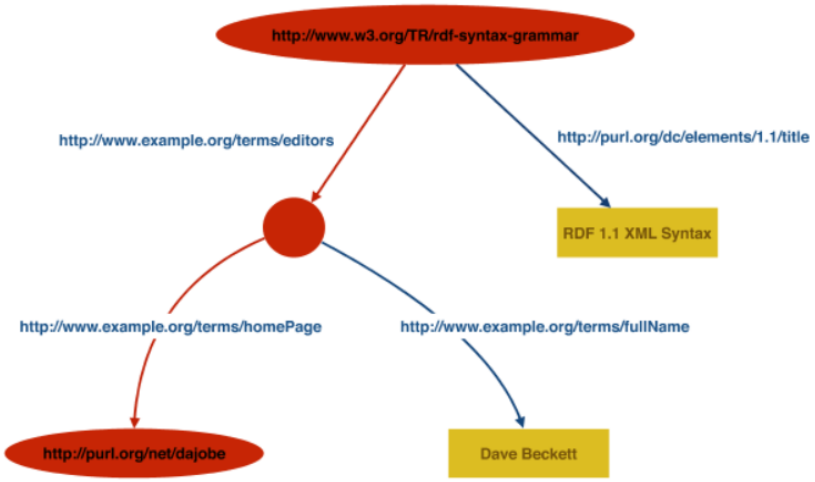
RDF-XML XML syntaxe pro RDF [10].

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
  <ex:editor>
    <rdf:Description>
      <ex:homePage>
        <rdf:Description rdf:about="http://purl.org/net/dajobe/">
        </rdf:Description>
      </ex:homePage>
    </rdf:Description>
  </ex:editor>
</rdf:Description>

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

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

-----------------------------------------------------------------------

<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc="http://purl.org/dc/elements/1.1/"
           xmlns:ex="http://example.org/stuff/1.0/">

  <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
            dc:title="RDF1.1 XML Syntax">
    <ex:editor>
```

```
      <rdf:Description ex:fullName="Dave Beckett">
        <ex:homePage rdf:resource="http://purl.org/net/dajobe/" />
      </rdf:Description>
    </ex:editor>
  </rdf:Description>

</rdf:RDF>
```

Výpis 2.3: RDF-XML příklady

Na ukázce 2.3 lze vidět v první části základní ukázku zápisu RDF grafu pomocí RDF-XML. V druhé části pak vidíme kompletní rdf-xml dokument s využitím vícerých zkrácení zápisu. Vizualizaci tohoto grafu lze vidět na obrázku 2.4.



Obrázek 2.4: RDF graf z rdf-xml dokumentu [10]

JSON-LD JSON-LD (JavaScript Object Notation for Linked Data) je konkrétní RDF syntaxe sloužící k serializaci a přenosu dat přes www [26] [18]. Výhodou JSON-LD je, že je plně kompatibilní s formátem JSON. Mohou tedy být využity již existující knihovny, nástroje a aplikace pro práci s daty. JSON-LD nabízí různé možnosti jako například univerzální identifikační mechanismus pro JSON objekty díky použití IRI. Možnost rozeznání klíčů sdílených mezi různými JSON dokumenty. **Mechanismus, ve kterém hodnota v JSON objektu může referovat objekt na jiné stránce webu.** Asociace datových typů s hodnotami jako je datum a čas. JSON-LD je designován, aby mohl být použit jak bez znalosti RDF, tak také v kombinaci s RDF technologiemi jako například SPARQL.

```
{
  "@context": {
    "modified": {
      "@id": "http://purl.org/dc/terms/modified",
      "@type": "http://www.w3.org/2001/XMLSchema#dateTime"
    }
```

```
  },
  ...
  "@id": "http://example.com/docs/1",
  "modified": "2010-05-29T14:17:39+02:00",
  ...
}
```

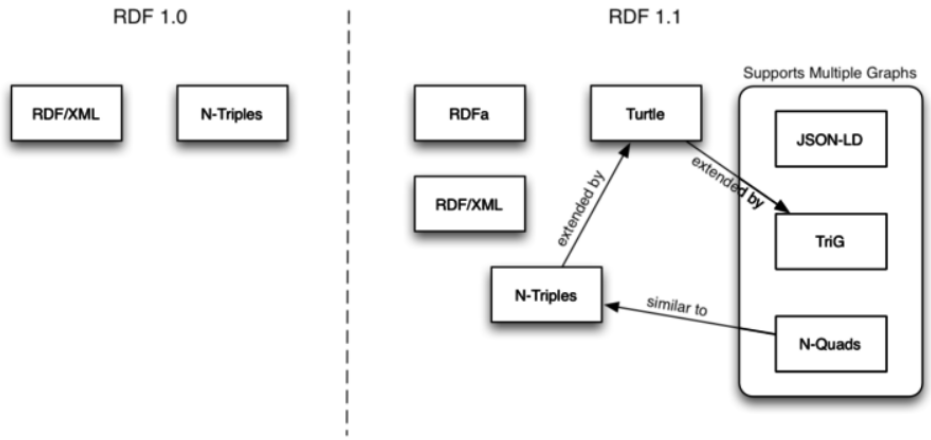
Výpis 2.4: JSON-LD příklady

Na ukázce 2.4 lze vidět reprezentaci uzlu pomocí atributu *@id* s hodnotou URI. Následně je zde atribut *modified*, který představuje predikát. V atributu *@context* je dále specifikováno jakou má predikát URI a na jaký typ dat ukazuje. Ve výsledku je JSON-LD z ukázky interpretováno dle tabulky 2.1.

Subjekt	Vlastnost	Hodnota	Typ hodnoty
*/docs/1	http://purl.org/dc/terms/modified	2010-05-29T14:17:39+02:00	xsd:dateTime

Tabulka 2.1: Interpretace JSON-LD příkladu

Ve výčtu RDF syntaxí lze vidět různé přístupy pro práci s RDF daty. Na obrázku 2.5 je vidět plný výčet RDF syntaxí. Mimo již zmíněné, jsou zde navíc syntaxe jako N-TRIPLES, která je podmnožinou TURTLE a syntaxe podporující více grafů jako například N-Quads, která je nadmnožinou TURTLE. Jak již bylo zmíněno JSON-LD poskytuje výhodu toho, že mnoho aplikací je již připraveno na práci s JSON dokumenty a je to tedy vhodný formát pro přenos RDF dat. Jeden z příkladů vhodného využití tohoto faktu je komunikace mezi klientem a serverem a toho je využito i v této práci.



Obrázek 2.5: RDF syntaxe [22]

2.5 SPARQL

SPARQL je dotazovací jazyk pro RDF [27]. Pomocí tohoto jazyku lze získávat a manipulovat data uložená v RDF databázi. Základním principem jazyku SPARQL je opět systém trojice subjekt, predikát, objekt. Jeho funkcionalita je ukázána na následujícím příkladu.

```
Data
<http://test.com/animals/monkey> <http://sub.com/content/eats> "Fruit"
<http://test.com/animals/whale> <http://sub.com/content/eats> "Plankton"
<http://test.com/animals/Bat> <http://sub.com/content/eats> "Insects"
<http://test.com/animals/Bat> <http://sub.com/content/isactive> "At night"

Dotaz
PREFIX pref: <http://sub.com/content/>
SELECT ?animal ?food
WHERE { ?animal pref:eats ?food }
```

Výpis 2.5: Ukázka SPARQL dotazu na datech

Subjekt	Objekt
<http://test.com/animals/monkey>	Fruit
<http://test.com/animals/whale>	Plankton
<http://test.com/animals/Bat>	Insects

Tabulka 2.2: Výsledek dotazu

V části prefix byla definována zkratka pro URI. V části SELECT bylo vybráno co nás bude zajímat, konkrétně tedy subjekt a objekt. V části WHERE byla definována trojice, která odpovídá námi hledánemu formátu. Hledáme všechny subjekty, které mají predikát *eats* a jaký objekt pomocí něj odkazují. Ve výsledku pak můžeme vidět všechny hledané subjekty - zvíře a objekty - potrava , které odpovídají vytvořenému dotazu.

Kapitola 3

Vektorová grafika na webu

SVG je platformou pro 2d grafiku, skládající se ze dvou částí [29]. První je formát na základě XML a druhou je aplikační rozhraní pro grafické aplikace. Mezi klíčové vlastnosti patří tvary, text a vestavěná rastrová grafika s různými styly kreslení. SVG podporuje skriptování jazyky jako ECMAScript a také obsahuje podporu pro animace. SVG je využíváno v mnoha bussiness oblastech, od webové grafiky po mobilní design.

SVG povoluje tři typy grafických objektů. Vektorové grafické tvary neboli cesty skládající se z rovných čar a křivek, obrázky a text. Grafické objekty mohou být stylovány, transformovány a komponovány do předešle vykreslených objektů. SVG kresby mohou být dynamické a interaktivní.

3.1 SVG DOM

DOM neboli document object model pro SVG, který zahrnuje plný XML DOM, dovoluje přímočarou a efektivní animaci vektorové grafiky přes skriptování. Grafickým objektům v SVG může být přidělena široká škála event handlerů, což umožňuje rošíření funkcionality SVG mimo pouhé zobrazování a dovoluje vytváření interkativních grafických aplikací.

SVG DOM podporuje všechna rozhraní a všechny druhy událostí definované v DOM Level 2 Events [9]. Jedná se o User Interface events: DOMFocusIn, DOMFocusOut, DOMActivate. Mouse events: click, mousedown, mouseup, mouseover, mousemove, mouseout. U mouse events pak clientX a clientY představují koordináty, na kterých se událost vyskytla relativně k DOM implementační klientské oblasti. Mutation events: DOMSubtreeModified, DOMNodeInserted, DOMNodeRemoved, DOMNodeRemovedFromDocument, DOMNodeInsertedInto, DocumentDOMAttrModified, DOMCharacterDataModified. SVG DOM dále definuje SVG specifické event rozhraní, SVGLoad, SVGUnload, SVGAbort, SVGError, SVGResize, SVGScroll. Dále vlastní event rozhraní SVGZoom. A event typy týkající se změn stavů při animaci beginEvent, endEvent, repeatEvent.

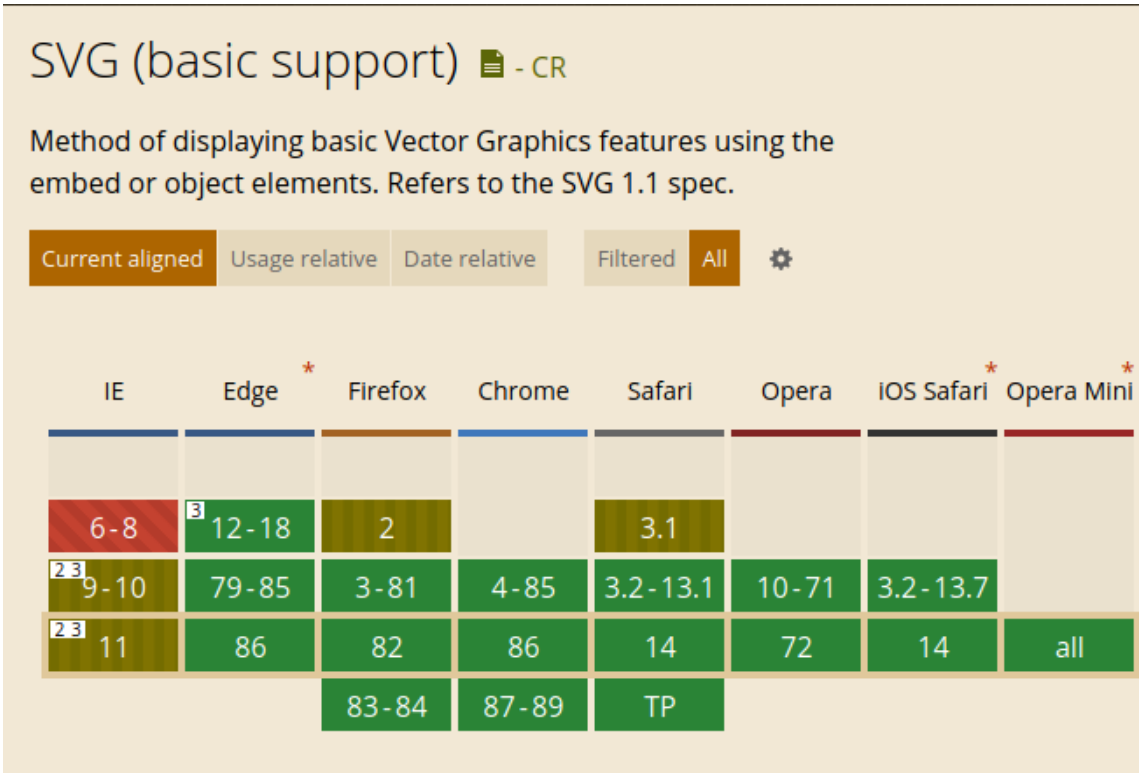
Dále SVG DOM podporuje užití CSS. Elementy lze stylovat pomocí CSS za použití určitých atributů. Některé atributy jsou sdíleny s CSS, jako například *font*, *cursor* a *color*, jiné jsou SVG specifické. SVG CSS atributy jsou takové atributy, které fungují pouze v SVG elementech. Jsou to například *aligment-baseline* pro textové atributy, *mask* pro maskující atributy a *stop-color* pro gradient atributy. Některé z těchto vlastností jsou pak specifické pouze pro některé svg elementy.

Při vykreslování svg elementů platí, že jsou vyobrazeny v takovém pořadí, v jakém jsou uloženy v DOM dokumentu. Tohoto je dále využito v práci.

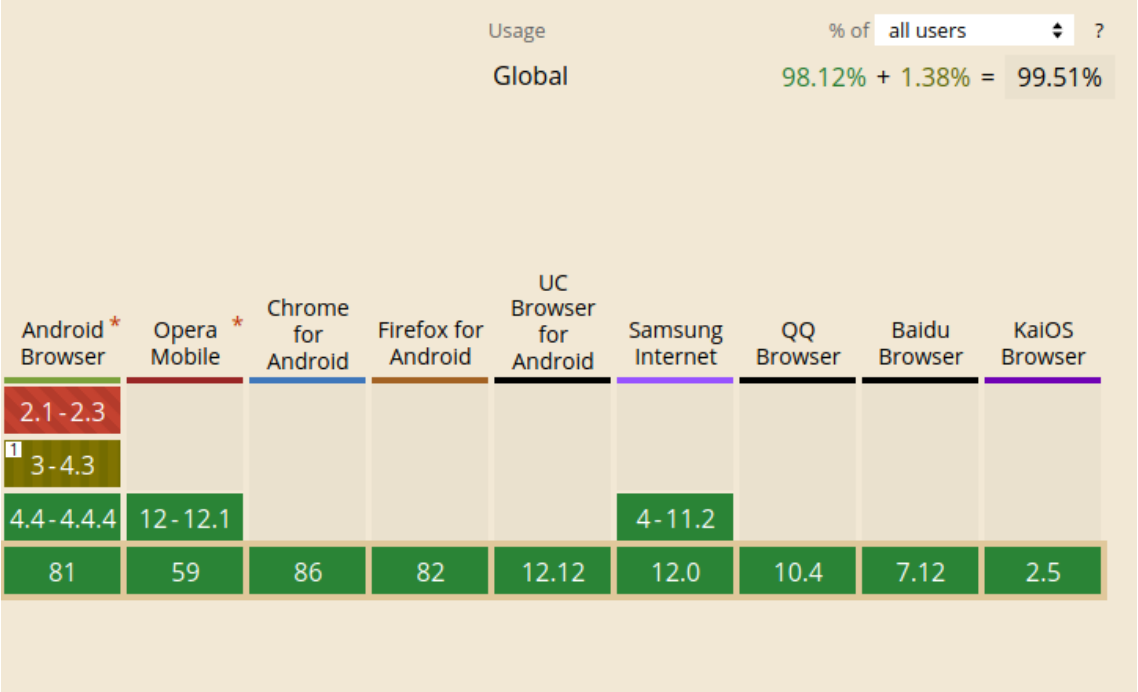
SVG nabízí praktické využití vlastností DOM dokumentu, které lze zužitečnit při budování grafické aplikace. Podpora SVG pro eventy a stylování je využita v této práci.

3.2 Podpora SVG

Podpora SVG v prohlížečích. Na základě údajů z běžně využívané webové služby caniuse, lze vidět následující data podpory SVG ve webových prohlížečích [5].



Obrázek 3.1: Podpora svg v prohlížečích k datu 25. 11. 2020 - první část



Obrázek 3.2: Podpora svg v prohlížečích k datu 25. 11. 2020 - druhá část

Je vidět, že svg je běžně podporováno ve všech důležitých a běžně používaných prohlížečích s výjimkou Internet Explorer, který poskytuje pouze částečnou podporu SVG.

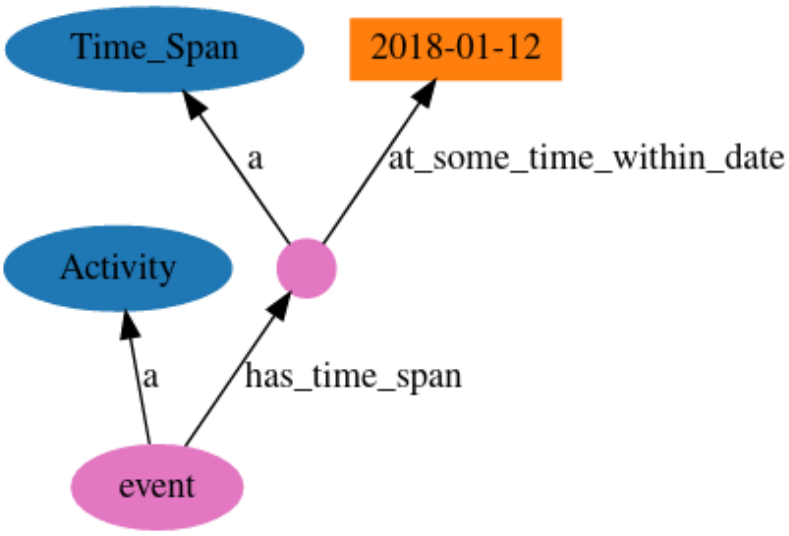
Kapitola 4

Existující řešení

V této kapitole jsou probrány existující řešení problematiky zobrazování RDF dat pomocí SVG. U každé nalezené technologie je krátce probráno, jak technologie funguje a co je její výstup. Z nalezených řešení jsou zde vybrána ta, která nejlépe odpovídají hledaným kritériím. V závěru kapitoly je provedeno zhodnocení průzkumu existujících řešení.

4.1 Ontology visualization

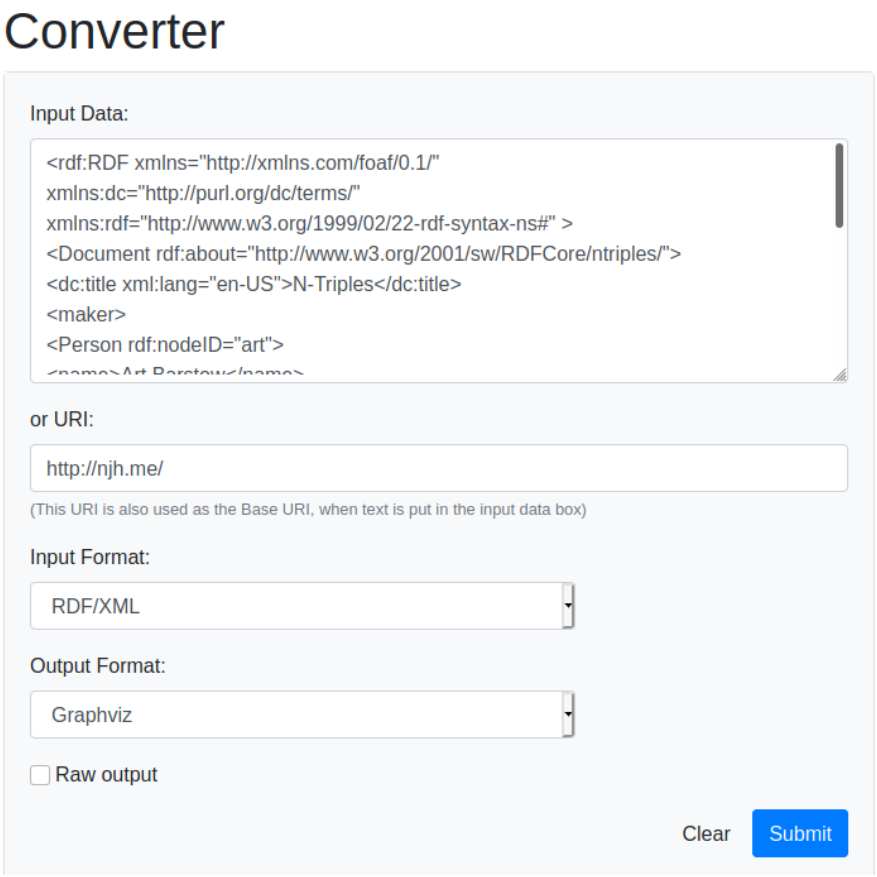
Python knihovna k vizualizaci RDF pomocí SVG [2].K samotné vizualizaci ve formátu SVG tato knihovna využívá další grafovou knihovnu graphviz. Realizuje RDF prvky pomocí elips, hran a obdélníků. Elipsa představuje subjekt nebo objekt, hrana predikát a obdélník hodnotu. Výsledek je pak bez možnosti animace nebo pohybu v zobrazení.



Obrázek 4.1: Ukázka ontology visualization

4.2 EasyRdf Converter

Online nástroj pro změnu formátu rdf dat [20]. Podpora více vstupních formátů i výstupních formátů. Místo vizualizace nabízí jako jeden z výstupů naformátovaný vstup pro software graphViz, který pak na základě tohoto vstupu může vytvořit SVG graf. Nejedná se tedy o vizualizační nástroj, ale v kombinaci s dalším softwarem má dostatečnou funkcionalitu k uskutečnění SVG zobrazení RDF dat. GraphViz je obecně software pro vizualizaci grafů libovolného formátu nebo typu.



Obrázek 4.2: Ukázka rdf converter input

Output

Number of triples parsed: 8

```
digraph {
  charset="utf-8";

  // Edges
  "Rhttp://www.w3.org/2001/sw/RDFCore/ntriples/" -> "Rhttp://xmlns.com/foaf/0.1/D
  "Rhttp://www.w3.org/2001/sw/RDFCore/ntriples/" -> "LN-Triples" [label="dc:title
  "Rhttp://www.w3.org/2001/sw/RDFCore/ntriples/" -> "B_:genid1" [label="foaf:make
  "Rhttp://www.w3.org/2001/sw/RDFCore/ntriples/" -> "B_:genid2" [label="foaf:make
  "B_:genid1" -> "Rhttp://xmlns.com/foaf/0.1/Person" [label="rdf:type"];
  "B_:genid1" -> "LArt Barstow" [label="foaf:name"];
  "B_:genid2" -> "Rhttp://xmlns.com/foaf/0.1/Person" [label="rdf:type"];
  "B_:genid2" -> "LDave Beckett" [label="foaf:name"];

  // Nodes
  "B_:genid1" [label="",shape=circle,color=green];
  "B_:genid2" [label="",shape=circle,color=green];
  "LArt Barstow" [label="Art Barstow",shape=record];
  "LDave Beckett" [label="Dave Beckett",shape=record];
  "LN-Triples" [label="N-Triples",shape=record];
  "Rhttp://www.w3.org/2001/sw/RDFCore/ntriples/" [URL="http://www.w3.org/2001/sw/
  "Rhttp://xmlns.com/foaf/0.1/Document" [URL="http://xmlns.com/foaf/0.1/Document"
  "Rhttp://xmlns.com/foaf/0.1/Person" [URL="http://xmlns.com/foaf/0.1/Person",lab
}
```

Obrázek 4.3: Ukázka rdf converter output

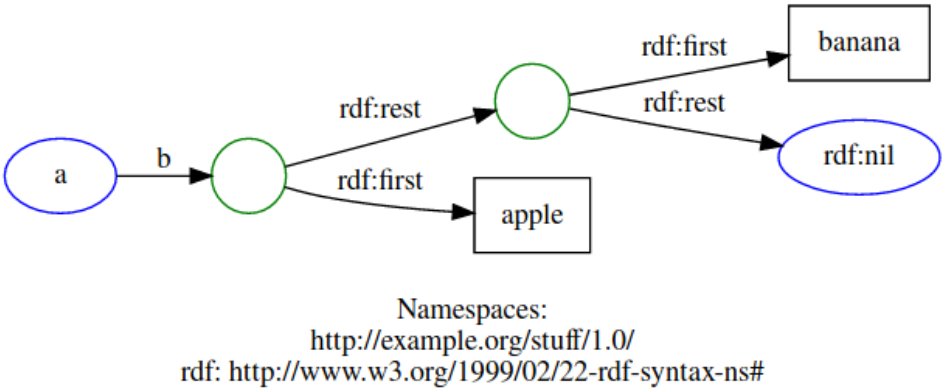
4.3 RDF2SVG

Java service RDF2SVG generuje svg zobrazení vstupních rdf dat [1]. Řešení není online, ale funguje jako spustitelná služba v java servlet container, například tedy apache tomcat. Vstup dat je možný ve třech formátech. Formáty jsou RDF/XML, N-TRIPLE, N3. Výstupem je pak vždy RDF graf v SVG. Bez animace, bez pohybu v zobrazení.

Obrázek 4.4: Ukázka rdf2svg

4.4 RDF grapher

RDF grapher je webová služba pro zobrazení rdf dat [14]. Služba je založena na knihovách graphviz a Redland Raptor. Grapher podporuje více vstupních formátů dat. Výstup služby může být opět ve více formátech, jedním z nich je RDF graf ve formátu PNG, který lze vidět na ukázce. Data mohou být zadána pomocí textu v poli služby nebo jako http odkaz pro velké datové sety. Služba nenabízí animace nebo kameru.



Obrázek 4.5: Ukázka RDF grapher

Při průzkumu technologií pro vizualizaci RDF pomocí SVG bylo zjištěno, že řešení poskytují často pouze výstup v obrazovém formátu bez možnosti interakce s grafem. Některá řešení pak poskytují pouze formátovaný výstup pro zpracování vizualizační knihovnou graphviz. Nejblíže interaktivní aplikaci je Java servlet RDF2SVG, který nabízí možnost vyobrazení dodatečných dat po kliknutí na uzly grafu. Možnost interaktivního procházení grafu nenabízí žádná ze zkoumaných možností.

Kapitola 5

Použité technologie

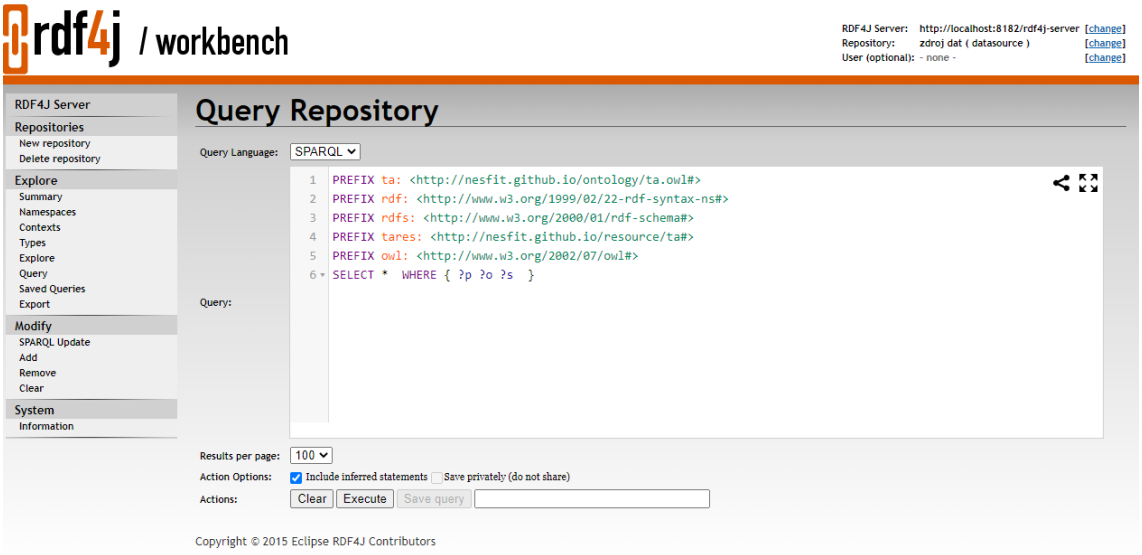
Aplikace pro zobrazování RDF dat je založena na webových technologiích. Je tedy použito HTML pro vytvoření webového dokumentu, CSS pro stylování elementů a jazyku Javascript pro logickou část aplikace na klientské straně. Dále je použita, jak již bylo zmíněno, technologie SVG pro vizualizaci grafů. Použití serverové strany je zdůvodněno požadavkem na uložiště dat, které pak klientská aplikace zobrazuje. Na straně serveru pak je využit Java framework RDF4J, který poskytuje funkcionalitu databáze a přístupového rozhraní pro RDF data. Dále je na straně serveru použit program pro výpočet matematicky náročného algoritmu pro rozmístění uzlů v jazyce Julia. Komunikace mezi serverem a klientskou aplikací je vedena pomocí REST API.

5.1 RDF4J

Eclipse RDF4J je open-source modulární framework pro práci s RDF daty [23]. Zahrnuje schopnost parsování, ukládání a dotazování nad RDF daty. Poskytuje API, které může být propojeno s mnohými řešeními ukládání RDF dat. Dovoluje propojení se s SPARQL endpointy. RDF4J podporuje všechny hlavní RDF formáty jako N-Triples, Turtle, RDF-XML, JSON-LD, TriG, N-Quads a TriX. Dále poskytuje nástroje pro práci s RDF daty jakou jsou RDF4J workbench a RDF4J server. Pro využití RDF4J je nutno použít vhodný Java servlet container. Pro účely této práce byl zvolen Apache Tomcat. Po instalaci Tomcatu je pouze nutné nasadit jednotlivé nástroje RDF4J.

RDF4J Server RDF4J server je databázová management aplikace. Poskytuje HTTP přístup do RDF4J repositářů. Přístup je vystaven jako SPARQL endpoint. RDF4J server slouží k užití ostatními aplikacemi. RDF4J server sám o sobě neposkytuje žádnou uživatelsky orientovanou funkcionalitu. K tomuto účelu slouží RDF4J Workbench.

RDF4J Workbench RDF4j workbench nabízí uživatelské grafické rozhraní. Mezi jeho funkcionalitu patří možnost procházení dat pomocí SPARQL query, nahrávání dat do RDF4J server repositářů a úpravu dat v repositářích.



Obrázek 5.1: Ukázka RDF4J workbench

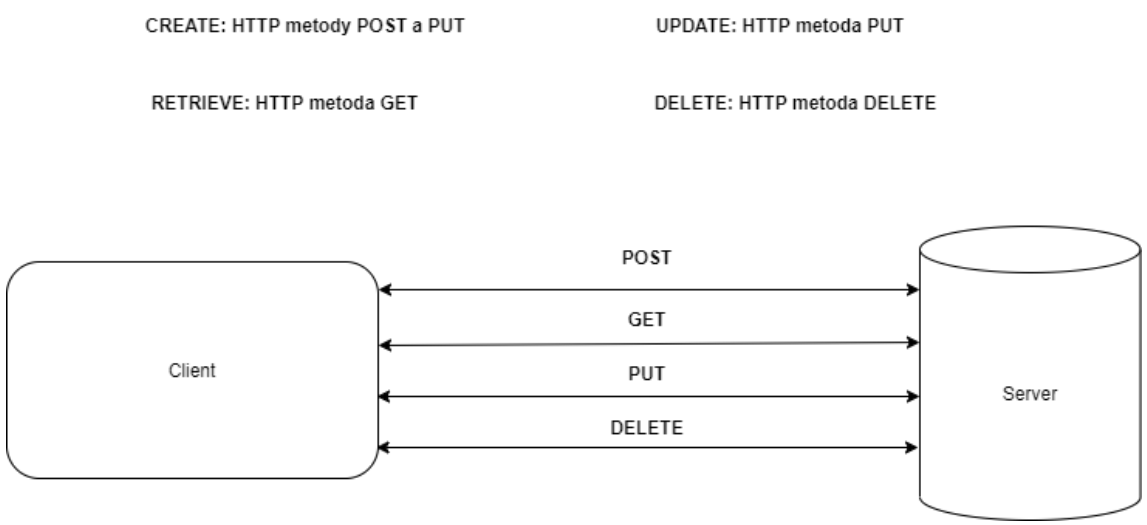
5.2 Programovací jazyk Julia

Julia je flexibilní dynamický jazyk vhodný pro vědecké a numerické výpočty s výkonem srovnatelným s klasickými staticky typovanými jazyky [4]. Julia používá odvození typů a Just In time kompilaci implementovanou pomocí LLVM. Jazyk tvoří více paradigmatů, kombinuje prvky imperativního, funkcionálního a objektově orientovaného programování. Julia nabízí lehkost a vysokou míru vyjádření pro vysokoúrovňové numerické výpočty, stejně tak jako například jazyky MATLAB a Python a přitom podporuje obecné programování. Julia používá Multiple Dispatch jako paradigma, což umožňuje snadno vyjádřit mnohé objektově orinetované a funkcionální vzory. Julia nabízí podporu pro interaktivní práci. Možnost síťové komunikace a vytvoření Julia serverů nebo volání Julia skriptů pomocí jiných programů nabízí možnosti využití v kombinaci s webovými technologiemi. Toto spolu s velkým výkonem, širokou podporou a přizpůsobením pro řešení složitých matematický problémů dělá z jazyku Julia vhodnou volbu pro tuto práci.

5.3 REST

Representational State Transfer (REST) je architektonický styl pro distribuované systémy [25]. Mezi základní principy patří client-server komunikace. Klient a server se mohou vyvíjet nezávisle na sobě, je zvýšena portabilita klientské části na různých platformách a zajištěna škálovatelnost serveru. Dalším klíčovým pojmem je bezstavovost. Zprávy mezi klientem a serverem musí obsahovat všechny potřebné informace k jejich zpracování, bez pamatování dodatečných instrukcí. Stav spojení(session) je tedy držen zcela na straně klienta. Tento princip zvyšuje viditelnost, spolehlivost a škálovatelnost. Díky jednoduchosti je snadnější zotavení z částečných chyb. Bezstavovost také umožňuje další zjednodušení serveru. Dalším důležitým pojmem je unifikované rozhraní, které RESTu umožňuje oddělení implementace a nabízených služeb, což vede k výhodě nezávislé evoluce. REST architektura využívá zdroje, které jsou definovány pomocí unikátních identidikátorů.

Webové služby, které dodržují zásady REST jsou nazývány RESTful [17]. Typickým příkladem RESTful komunikace s protokolem http je zvládnutí operací CRUD (create,read, update, delete) nad zdroji. Metody http implementující tyto operace lze vidět na ukázce.



Obrázek 5.2: Ukázka REST Api

5.4 Websocket Api

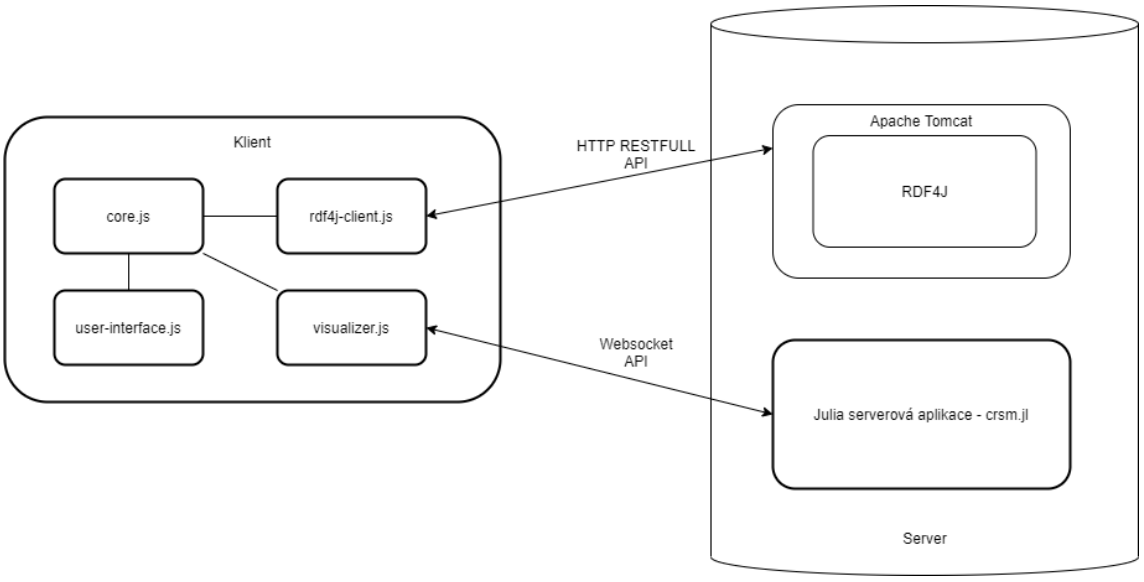
Websocket protokol umožňuje oboustranou komunikaci mezi clientem a serverem [6]. Protokol se skládá z počátečního handshaku následovaným základním rámcováním zpráv přes TCP. Cílem Websocket protokolu je poskytnout mechanismus pro browser založené aplikace, které potřebují oboustranou komunikaci se serverem, která nespoléhá na více http připojení jako například polling. Díky websocketům může klient komunikovat se serverem za účelem výměny dat. Výhodou je také například navázání spojení s serverovou aplikací v jednoduchých případech, kdy pak není nutno implementovat celé Api za pomoci rozsáhlých frameworků.

Kapitola 6

Návrh řešení

V této kapitole je probrán návrh aplikace. Konkrétně získání RDF dat k zobrazení ze strany aplikace, nahrání dat na server, funkcionalita aplikace a její využití. Klientská strana aplikace je rozdělena na několik částí. Komunikace s databází rdf4j-client.js. Zobrazení do DOMu ve visualizer.js, inicializace zobrazení, interakce s uživatelem a správa UI v core.js. Další doplňující funkce týkající se interaktivity a akcí uživatele jsou v modulu user-interface.js.

Na straně serveru je pak uložiště dat v podobě RDF4J, které poskytuje http endpoint pro komunikaci s klientem. Kromě toho na serveru běží program crsm.jl, který slouží k výpočtu pokročilého zobrazení grafů. Komunikace mezi klientem a crsm.jl probíhá přes websocket Api.



Obrázek 6.1: Návrh aplikace

6.1 Uložiště dat

Pro zvolení uložiště bylo nutné brát v potaz následující kritéria. Samotná podpora formátu RDF, podpora REST api na získání dat, možnosti formátů při nahrání dat a volná dostupnost samotného řešení. Tyto požadavky vhodně splňuje právě RDF4j. RDF4j nabízí ser-

verovou aplikaci RDF4J server a uživatelské rozhraní pro kontrolu dat RDF4J workbench. RDF4J je volně dostupný framework, který poskytuje endpoint pro http komunikaci, dále podporuje nejvíce rozšířený formát vstupních dat RDF/XML, čímž splňuje všechny požadavky. Z výkonnostního hlediska se předpokládá, že hlavním omezujícím faktorem je klienstká část, takže postačuje tato standardní databáze. Jako Java servlet container je použit Apache Tomcat.

Nahrání dat

Data jsou do databáze vkládána v jednom z mnoha formátu podporovaných RDF4j, jako například RDF-XML, TURTLE nebo JSON-LD. RDF soubory jsou nahrány na server přes RDF4J workbench. Uživatel vybere vstupní data a nahraje je do repositáře přes RDF4J workbench UI.

6.2 Funkcionalita klienta

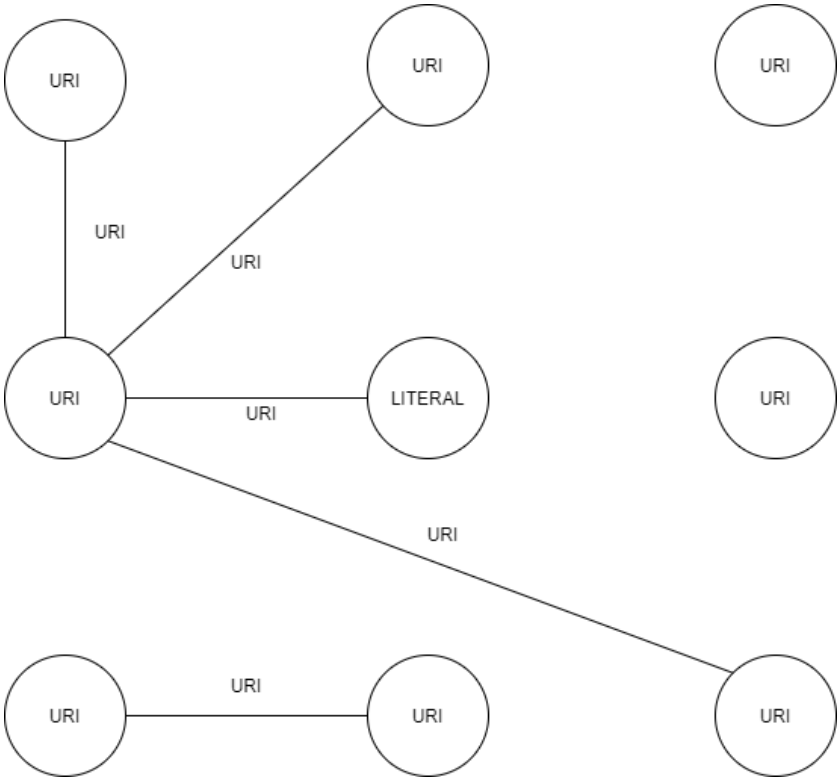
Aplikace zobrazuje grafickou reprezentaci RDF dat. Tedy uzly představující subjekty a objekty a hrany představující predikáty. Graf je, jak již bylo zmíněno, zobrazen pomocí SVG. Rozložení uzlů je tvořeno třemi způsoby.

1. První metodou je náhodné rozmístění do mřížky, poskytující největší výkon.
2. Následně je použita technika záměny uzlů s cílem minimalizace celkové délky hran.
3. Posledním zobrazením je metoda založená na minimalizaci překročení hran (Crossing reduction with Stress majorization).

Aplikace obsahuje možnost výběru počátečního uzlu na základě uživatelem zadané URI. Toto umožní zobrazovat určité okolí dat, které uživatele zajímá. Následně po kliknutí na uzel se zobrazí všechny uzly, které jsou jeho objekty. Postupně lze takhle zobrazovat vztahy, vycházející z počátku, který uživatele zajímá. Kromě konkrétního zobrazení na základě zadané URI, aplikace umožňuje zobrazení horně omezeného množství dat. Omezení je blíže dáno částí práce, která se zabývá testováním. Takovéto zobrazení je pak náhodné a slouží ke kvantitativnímu zobrazování dat. Aplikace umožňuje pohyb a přibližování v zobrazovací ploše SVG pro průzkum grafu a jeho dat. Dále je možné měnit polohu uzlů a uspořádání grafu, což dále zlepšuje praktickou stránku a uživatelskou zkušenost.

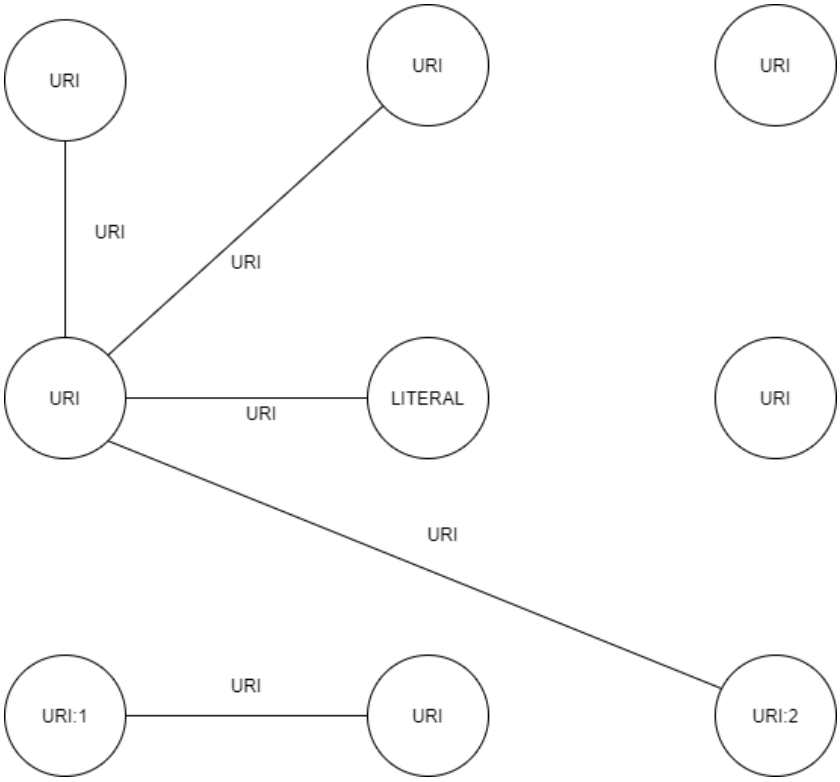
6.3 Zobrazení

Prvním typem je základní zobrazení uzlů do mřížky. Uzly jsou uspořádany v pořadí, v jakém přicházejí jako data ze serveru. Mezi uzly je zvolena základní vhodná vzdálenost. Toto zobrazení disponuje největším výkonem, k jeho uskutečnění je nutné pouze minimální množství výpočtů. Výměnou za výkonnost je malá intuitivita, v případě velkých grafů s velkým množstvím predikátů může působit graf velmi nepřehledně. Nepřehlednost částečně řeší možnost manuální reorganizace uzlů, která je ale v případě velkého množství uzlů velmi nepraktická. Ukázka základního rozložení 6.2.



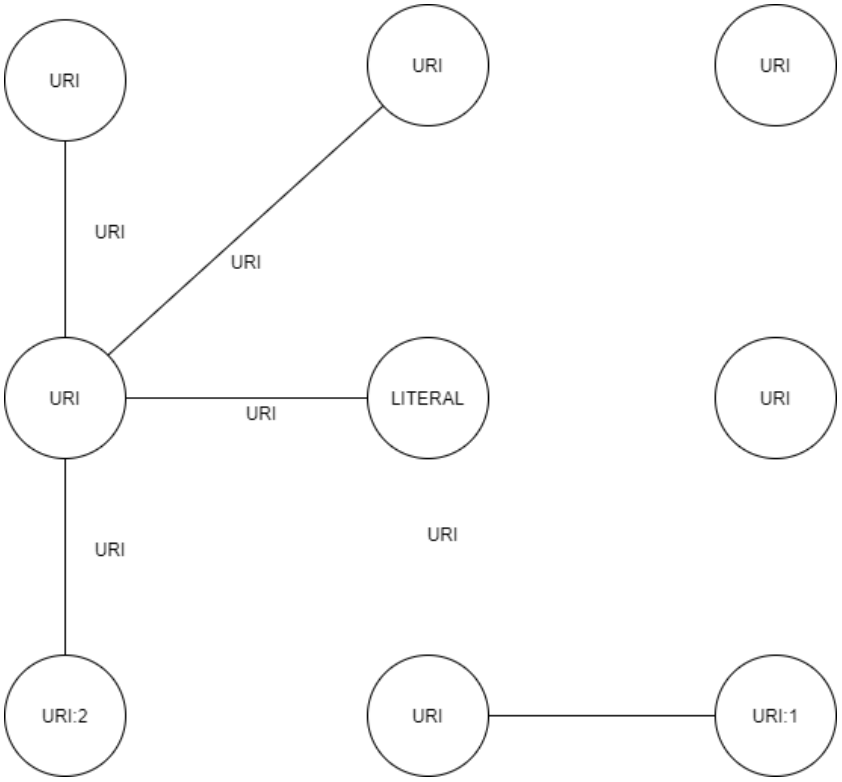
Obrázek 6.2: Základní rozložení uzlů do mřížky

Dalším typem zobrazení je opět organizace uzlů do mřížky s následným prohozením pozic uzlů tak, aby došlo k minimalizaci celkové délky hran. Toto zobrazení zachovává rozumný výpočetní výkon za zvětšení míry intuitivity. Algoritmus nejprve náhodně rozmístí uzly do mřížky a následně aplikuje prohazování uzlů dokud se zmenšuje celková hodnota délky hran. Při dokončení algoritmu se pak zobrazí mřížka se zvýšenou přehledností díky zmenšení překrytí hran. Nad grafem pak opět lze dodatečně provádět změny poloh uzlů. Na obrázku 6.3 lze vidět graf před použitím zobrazení.



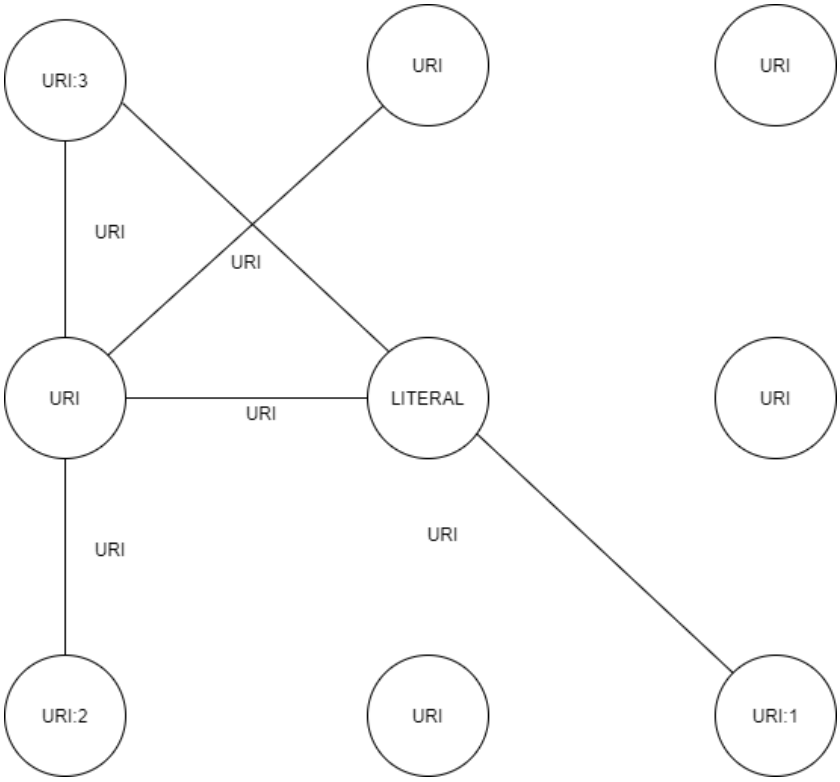
Obrázek 6.3: Základní rozložení uzlů do mřížky před algoritmem záměny uzlů

Na obrázku 6.4 pak je možné vidět stav grafu po použití minimalizace délky hran. Uzel s URI:1 byl zaměněn s uzlem s URI:2, čímž došlo k minimalizaci celkové délky hran. Po tomto kroku došlo ke zvýšení přehlednosti a čitelnosti grafu za užití přiměřeně náročného výpočtu. Jedná se o více náročný typ zobrazení než typ základní, ale stále zachovává přijatelně vysokou výkonnost díky jednoduchosti výpočtu.



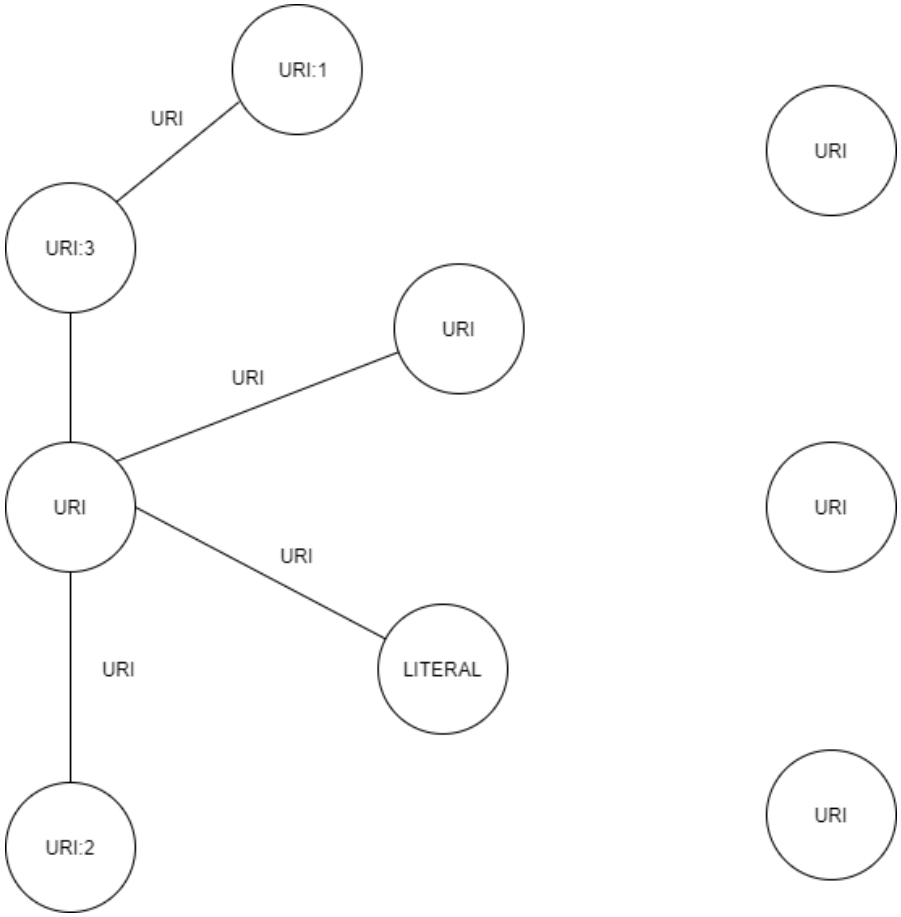
Obrázek 6.4: Základní rozložení uzlů do mřížky po algoritmu záměny uzlů

Posledním typem zobrazení je metoda CRSM (Crossing Reduction with Stress Majorization) inspirovaná dokumentem o problematice redukce překrytí hran v grafech [16]. Zobrazení využívá pokročilých matematických metod k takovému rozložení uzlů, ve kterém dochází k redukci překrytí hran. Uzly pak nejsou zobrazeny v mřížce jako v předešlých zobrazeních, ale jejich polohy jsou dány výstupem algoritmu, který vytváří rozložení s omezeným překrytím. Princip je blíže vysvětlen v části 6. Na obrázku 6.5 pak lze vidět situaci, kdy dochází k překrytí hran.



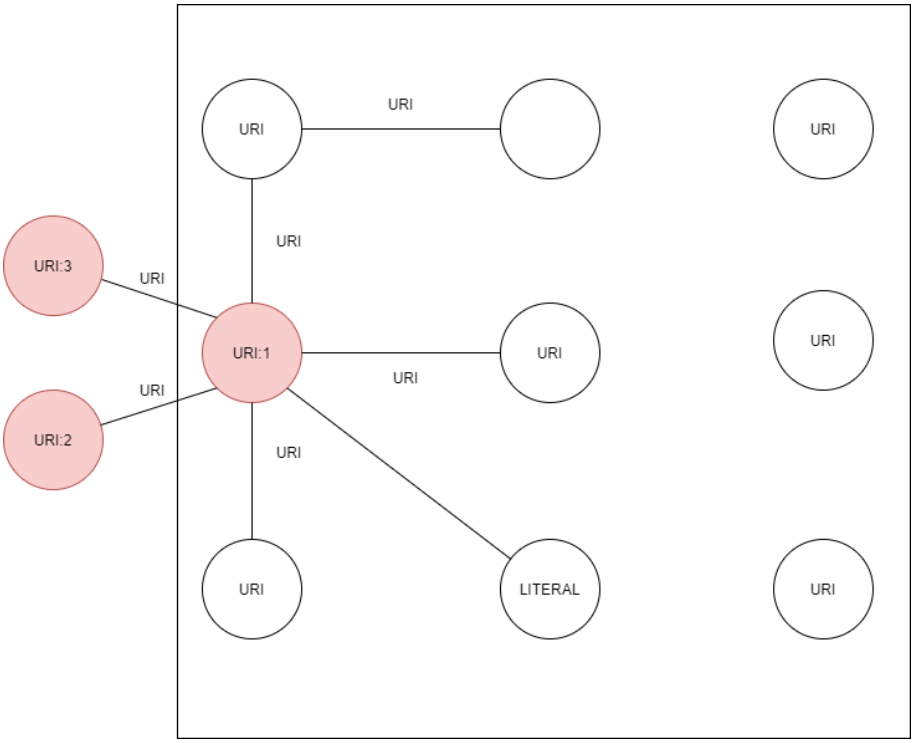
Obrázek 6.5: Základní rozložení uzlů do mřížky s překrytím hran

Po aplikaci algoritmu CRSM je vyobrazen graf bez překrytí. Uzly nadále nejsou rozloženy v mřížce, ale polohy jsou určeny úpravami provedenými algoritmem. Metoda CSRM nabízí nejintuitivnější zobrazení grafu za cenu nejvyšší náročnosti na výkon. Blíže k tomuto tématu v kapitole testování.



Obrázek 6.6: Rozložení grafu po CSRM zobrazení

Přidávání dalších uzlů po iniciálním zobrazení je vyřešeno mechanismem, kdy se nové uzly zobrazí mimo hranice současného obalujícího čtverce grafu. Tímto je umožněno přidání nových uzlů bez nutnosti přepočítat celý graf, což by vedlo k poklesu výkonnosti a pozitivní uživatelské zkušenosti. Aplikace drží informaci o svých momentálních hranicích v prostoru a při kliknutí na uzel s cílem zobrazit všechny objekty, na které ukazuje svými predikáty, se určí uzlu nejbližšší hranice, za ní se vyobrazí nové uzly. Následně se posune hranice ve vyobrazeném směru. Hranice se rovněž posouvají při přesunech uzlů uživatelem. Tato technika se ukazuje jako vhodné a uživatelsky přívětivé řešení přidávání dalších uzlů. Ukázka mechanismu je vidět na obrázku 6.7.



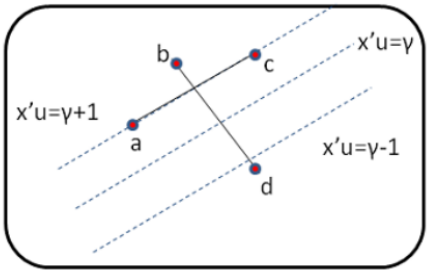
Obrázek 6.7: Přidání uzlů

Po kliknutí na uzel URI:1 se zjistí jeho nejbližší hranice z obalovacího čtverce a za ní se vyobrazí uzly, na které odkazuje pomocí svých predikátů.

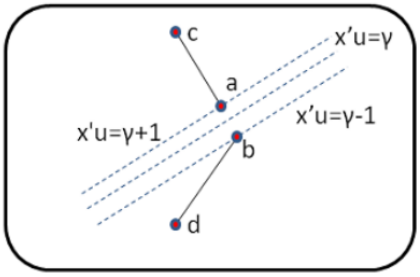
6.3.1 Teorie metody Crossings Reduction with Stress Majorization

Použitý algoritmus je založen na dokumentu popisující využití metody CRSM k redukci překročení hran v grafu [16]. Vstupem algoritmu jsou pozice uzlů a pozice hran.
Samotný algoritmus se skládá ze dvou částí:

V první části algoritmu se odehrává detekce hran, které se protínají. Problém protnutí dvou hran lze popsat za pomoci systému lineárních rovnic. Jelikož každá hrana je konvexní polyhedron, tak platí, že $\exists u \neq 0$ a γ takové, že pro dvě hrany, které se neprotínají je $xu - \gamma$ nekladné pro všechny body $x \in X^2$ ležící na první hraně a nezáporné pro všechny body na hraně druhé. Jinak řečeno existuje dělící přímka, která rozděluje prostor na část s jednou hranou a část s hranou druhou jak lze vidět na obrázcích 6.8a a 6.8b.



(a) Hyperplochy při překročení hran



(b) Hyperplochy při nepřekročení hran

Obrázek 6.8: Rozdělovací plochy

Každý bod hrany může být popsán konvexní kombinací extrémních bodů dané hrany. Mějme hranu k s koncovými body $a = [a_x\ a_y]$ a $b = [b_x\ b_y]$ a hranu l s koncovými body $c = [c_x\ c_y]$ a $d = [d_x\ d_y]$. Matice A a B představují extrémní body hran k a l. Jakýkoliv bod protnutí hran k a l může být zapsán jako konvexní kombinace extrémních bodů hrany A a hrany B. Lze ukázat, že se dvě hrany neprotínají, právě tehdy když následující systém rovnic nemá řešení.

$$\exists\ \delta_k, \delta_l \in R^2 \text{ takové, že } A\delta_k = \delta_l B \quad e^{'}\delta_k = 1 \quad e^{'}\delta_l = 1 \quad \delta_k \geq 0 \quad \delta_l \geq 0$$

Kde e je dvourozměrný jedničkový vektor a A=$\begin{pmatrix} a_x & a_y \\ b_x & b_y \end{pmatrix}$, B=$\begin{pmatrix} c_x & c_y \\ d_x & d_y \end{pmatrix}$

Daný systém rovnic lze převést na optimalizační úlohu 6.1, která nabývá hodnoty 0 tehdy a pouze tehdy, pokud se hrany k a l neprotínají. To znamená, že $\exists u \in R^2,\ \exists \gamma \in R$ takové, že:

$$0 \ = \ min_{u,\gamma} f(A, B, u, \gamma) \ = min_{u,\gamma} \left\| (-Au + (\gamma + 1)e)_+ \right\|_2^2 \ + \ \left\| (Bu - (\gamma - 1)e)_+ \right\|_2^2 \quad (6.1)$$

V této implementaci se používá klasická euklidovská norma. Díky této normě se jedná o kvadratické programování. Výsledkem optimalizace je pak zjištění, zda došlo k protnutí, či nikoliv. Pokud ano, tak jsou napočítané i koeficienty rozdělovacího prostoru u a γ.

V druhé částí algoritmu je opět využita optimalizace. Kombinují se zde dva penalizační prvky. Prvním z nich je samotné protnutí hran, jež je popsáno výše a druhým z nich je penalizace za libovolnou diferenci vzdálenosti od původního rozložení. Tato penalizační funkce se nazývá stresová funkce.

Samotná stresová funkce vypadá takto:

$$stress(X) = \sum_{i<j} w_{ij} (\left\| X_i - X_j \right\|_2 - d_{ij})^2 \quad (6.2)$$

Člen X_i představuje novou pozici uzlu pomocí jeho x a y souřadnic. d_{ij} je vzdálenost uzlů i a j ve starém zobrazení a w_{ij} je normalizační konstanta, která odpovídá rovnici $w_{ij} = d_{ij}^{-\alpha}$, kdy $\alpha = 2$. Funkční hodnota stresové funkce roste, pokud se vzdálenost mezi dvěma body zvětší nebo změnší oproti původnímu rozložení.

Dále lze vidět, že se jedná o problém nelineárního programování. Aby byla zaručená optimálnost a řešitelnost, využívá se její kvadratická aproximace 6.3.

$$Fstress(X,Z) = \sum_{i<j} w_{ij} d_{ij}^2 + Tr(X' L^w X) - 2Tr(X' L^z Z) \tag{6.3}$$

V této aproximaci je zapotřebí podpůrná matice bodů Z. Tato matice je dosazena z minulé iterace matice bodů X. Při spojení penalizace za protnutí a funkce Fstress dostáváme finální optimalizační problém 6.4.

$$\begin{aligned} min_{X,U,r} p(X,Z,U,r) = min_{X,U,r}\ & Fstress(X,Z) \\ & + \sum_{i=1}^{m} \frac{\rho_i}{2} [\left\| (-A^i X U_i + (r_i+1)e)_+ \right\|_2^2 \left\| (B^i X U_i - (r_i-1)e)_+ \right\|_2^2] \end{aligned} \tag{6.4}$$

Vstupem optimalizačního problému je hodnota Z, neboli minulá hodnota rozložení X a množiny U a r, které obsahují jednotlivé hodnoty koeficientů rozdělovacích ploch vypočítaných při detekci překročení hran.

Algoritmus v první části detekuje překročení pomocí optimalizačního problému 6.1. Pro každý pár hran je vypočítáno, zda došlo k překročení, a následně jsou přiřazeny koeficienty u a γ rozdělujícího prostoru. V druhé části algoritmus řeší minimalizační problém 6.4, kde se snaží najít takové rozložení X, které se co nejvíce přibližuje původnímu rozložení z hlediska vzdáleností mezi body (funkce Fstress) a zároveň je hledání řešení ovlivněno penalizační funkcí, která penalizuje výsledné rozložení pokud dochází k protnutí hran. Penalizace je ovlivňována penalizační konstantou ρ, která se iterativně zvyšuje pro každý cyklus, kdy se dvě hrany protnou. Algoritmus pokračuje v hledání nových rozložení X, dokud není zaručeno, že zkonvergoval k řešení.

Algorithm 1 CR-SM algoritmus

$C \leftarrow \emptyset$
$U* \leftarrow []$
$\gamma* \leftarrow []$
repeat
  repeat
    for each edge pair A^i and B^i, i = 1...,m **do**
      $(u^*, \gamma^*) \leftarrow minf(A^i, B^i, U_i, \gamma_i)$
      if $f(A^i, B^i, U_i, \gamma_i) \geq \tau$ **and** $C \neq \emptyset$ **then**
        $C \leftarrow C \cup i$
        $\rho_i \leftarrow \rho_{min}$
        $U_i \leftarrow u^*$
        $\gamma_i* \leftarrow \gamma^*$
      end if
    end for
    $Z \leftarrow X^j \cup X$
    $L^Z \leftarrow L^{X^j}$
    $X^{j+1} \leftarrow minp(X, Z, U^*, \gamma^*)$
    $j \leftarrow j + 1$
    $\rho_i \leftarrow min(\rho_{inc} \times \rho_i, \rho_{max})$
  until $\left\| X_j - X_{j-1} \right\| \geq \epsilon$ **and** $C \neq \emptyset$
until $\left\| X_j - X_{j-1} \right\| \geq \epsilon$ **and** $\frac{\rho_i}{2} p(X_j, X_{j-1}, U*, r*) \geq \epsilon$ **and** $C \neq \emptyset$

Kapitola 7

Implementace

Tato kapitola popisuje klíčové části implementace aplikace se zaměřením na důležité části kódu. Implementování aplikace je popsáno postupně od části získání dat, až po jejich vizualizaci. Na závěr implementace jsou popsány doplňkové funkcionality a informace o přístupu k implementování.

7.1 Získání dat

První částí funkcionality aplikace je získání dat. To probíhá pomocí technologie AJAX, kdy jsou odesílány dotazy na server, jejichž obsahem je SPARQL query. RDF4J server pak odpovídá zprávou ve formátu JSON, jež obsahuje výsledek query z dotazu, které bylo na serveru vykonáno. Dotazy jsou odesílány v javascriptovém modulu rdf4j-client.js, kde jsou formovány query pro jednotlivé účely. Příklad komunikace je vidět na ukázce 7.1, kde je vidět obsah query zaslaného na server a datovou odpověď serveru ve formátu JSON 7.2. Je zde konkrétně funkce *RDFClient.prototype.getLimitedNodes*, která plní funkcionalitu zobrazení obecného vzorku z databáze.

```
RDFClient.prototype.getLimitedNodes = function(limit) {
        var client = this;
        var query = this.getPrefixes() + 'SELECT * WHERE { ?s ?p ?o} LIMIT '
        + limit;
        return new Promise(function(resolve, reject) {
                var p = client.sendQuery(query);
            p.then(function (data) {

                        resolve(data);
                }).catch(function(reason) {
                        reject(reason);
                });
        });
};
```

Výpis 7.1: Funkce vytvářející querry pro obecné data

```
{
  "head" : {
    "vars" : [
```

```
      "s",
      "p",
      "o"
    ]
  },
  "results" : {
    "bindings" : [
      {
        "p" : {
          "type" : "uri",
          "value" : "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
        },
        "s" : {
          "type" : "uri",
          "value" : "http://nesfit.github.io/ontology/ta.owl#sourceId"
        },
        "o" : {
          "type" : "uri",
          "value" : "http://www.w3.org/2002/07/owl#DatatypeProperty"
        }
      },
      ...
    ]
  }
}
```

Výpis 7.2: Odpověď serveru ve formátu JSON

Data jsou po obdržení převedena do javascriptových objektů a podle typu akce jsou dále předána k určitému zobrazení.

7.2 Zpracování dat

Data jsou před zobrazením převedena na objekty obsahující atributy potřebné k zobrazení. Příklad objektu je vidět na ukázce 7.3. Algoritmicky jsou těmto objektům přiřazeny atributy, podle kterých je SVG rozmístí a spojí. Rozmístění je závislé na zvoleném typu zobrazení. Na zmíněné ukázce je vidět zobrazení základní, které je pak jádrem pro pokročilejší zobrazení, kdy se mění poziční parametry těchto objektů na základě zobrazovacího algoritmu.

```
nodeArray[dataArray[i].value] = {
   id: dataArray[i].value,
   type: dataArray[i].type,
   datatype:
   (dataArray[i].datatype === null || dataArray[i].datatype === undefined)
   ? null : dataArray[i].datatype,
   cx: 100 * xShift,
   cy: 100 + yShift,
   r: 50,
```

```
    stroke: "#11998e",
    strokewidth: 1,
    fillAt: "#11998e",
};
```

Výpis 7.3: Ukázka základních objektů pro zobrazení v SVG

7.3 Interaktivita

Interaktivita aplikace se skládá z několika prvků. Prvním z nich je možnost přiblížení a pohybu v zobrazovacím prostoru. Tuto funkcionalitu zajišťuje externí javascriptová knihovna pro práci s SVG PanAndZoom [1]. Ve funkci *panAndDrag* jsou nastaveny parametry funkcionality zoomu a zároveň vytvořeny funkce pro zpracování speciálních událostí, které se týkají dalšího prvku interaktivity. Tímto prvkem je přesun uzlů po zobrazovací ploše. Spolu s uzly se mění i poloha jejich spojovacích čar. Toto je obstaraváno ve funkcích, které mění polohy prvků při tahu myší. Jednou z těchto funkcí je *drag*, která mění koordináty uzlu při tahu myší 7.4. Na ukázce je vidět užití SVG DOMu, kdy lze pracovat přímo s konkrétním objektem místo nutnosti překreslovat celé zobrazení.

```
function drag(evt) {
      if (selectedElement) {
        evt.preventDefault();
        var coord = getMousePosition(evt);
        moveEdges(coord, offset,selectedElement.getAttributeNS(null, "id"));
        var textelement = document.getElementById("text-" +
        selectedElement.getAttributeNS(null, "id"));
        textelement.setAttributeNS(null, "x", coord.x - offset.x - 50);
        textelement.setAttributeNS(null, "y", coord.y - offset.y + 65);
        selectedElement.setAttributeNS(null, "cx", coord.x - offset.x);
        selectedElement.setAttributeNS(null, "cy", coord.y - offset.y);
      }
}
```

Výpis 7.4: Ukázka funkce pro přesun uzlů

Mezi další části interaktivity aplikace patří vyhledání uzlu v SVG ploše a zaměření pohledu na daný uzel. Aplikace vyhledává hledaný uzel, který je specifikován pomocí substringu zadaného uživatelem. Substring musí být obsažen v hledaném uzlu. Ve funkci *zoomAtSubstring* modulu user-interface.js probíhá prohledávání DOM stromu SVG elementů a po nalezení uzlu, jehož id odpovídá zadanému substringu, upraví pohled na plochu tak, aby byl zaměřen na požadovaný uzel 7.4. Na ukázce lze opět vidět využití funkcionality SVG DOMu při prohledávání uzlů na podřetězec zadaný uživatelem.

```
function zoomAtSubstring() {
        var g = document.getElementById('viewport');
        var nodes = g.children;
        var str = document.getElementById("search-uri").value;
        for (var i = 0; i < nodes.length; i++ ) {
```

[1] Knihovna PanAndZoom - https://github.com/ariutta/svg-pan-zoom

```
                var node = nodes[i];
                var subg = node.children;
                if (subg.length !== 0) {
                        var check = subg[0].getAttributeNS(null, "id");
                        if (check.includes(str)) {
                                svgZoom.zoom(1);
                                var xPos = 200;
                                var yPos = 200;
                                svgZoom.pan({x:
                                -Number(subg[0].getAttributeNS(null, "cx"))
                                + xPos,
                                y: -Number(subg[0].getAttributeNS(null, "cy"))
                                + yPos });
                                break;
                        }
                }
        }
}
```

Výpis 7.5: Funkce pro zaměření na uzel

7.4 Rozšiřování zobrazení

Rozšíření zobrazeného grafu probíhá pomocí funkce *getNeighbours*, která iniciuje zaslání SPARQL dotazu na objekty, které daný uzel odkazuje. Po získání dat se nová data vyobrazí jako uzly vázané na původní subjekt na nejbližší hraně obalovacího čtverce pomocí funkce *addNodesSquareMethod*. V této funkci se rozhodne nejbližší hrana a následně se pomocí finální funkce *addNodesToSvg* zobrazí nové uzly v patřičném prostoru. Po zobrazení se vypočítá nový obalovací čtverec pro zobrazení dalších dat. Při potřebě uživatele je možné zvolit jiný typ zobrazení, který se provede nad současně zobrazenými daty.

7.5 Implementace zobrazení

V této části je popsána implementace jednotlivých zobrazení. První dvě jsou implementována v hlavním modulu aplikace, kterým je core.js a zobrazení metodou CRSM je pak implementováno v programu crsm.jl běžícím na serveru.

7.5.1 Základní zobrazení

Realizace tohoto zobrazení je ve funkci *baseVisualization*. V první fázi dochází k vytvoření objektů pro uzly, jak již bylo zmíněno a ukázáno v části zpracování dat 7.2. V další části jsou přiřazeny vazby na základě predikátů. Na základě těchto vazeb jsou vytvořeny objekty pro hrany grafu. Tyto objekty jsou pak v modelu visualizer.js použity pro vytvoření SVG elemetů pro zobrazení. Příklad objektu je vidět na ukázce 7.6.

```
lineArray[sourceNode.id+destinationNode.id] = {
    text: nodeConnections[i].id,
    id: 'line' + nodeConnections[i].id,
```

```
    relationId: nodeConnections[i].id,
    x1: sourceNode.cx,
    y1: sourceNode.cy,
    x2: destinationNode.cx,
    y2: destinationNode.cy,
};
```

Výpis 7.6: Objekt pro hranu grafu

Zobrazovací algoritmus v základním zobrazení funguje na principu zobrazení do čtvercové mřížky. Na základě počtu datových uzlů se spočítá dostatečná plocha čtverce pro jejich zobrazení. Následně jsou v cyklu přiřazovány hodnoty souřadnic odpovídající vyplňování čtverce. Tato forma zobrazení je základem i pro pokročilá zobrazení, kdy s tímto inicálním rozložením uzlů pracují algoritmy vytvářející lepší zobrazení.

7.5.2 Zobrazení se záměnou uzlů

Realizace tohoto zobrazení je ve funkci *positionSwitchVisualization*. V první fázi dochází k vytvoření objektů pro uzly, jak již bylo zmíněno a ukázáno v části 7.2. Poté jsou opět přiřazeny vazby na základě predikátů.

Následuje práce algoritmu, který provádí záměny pozic uzlů a porovnává hodnotu TEL (total edge length) s předchozí iterací. V každé iteraci algoritmu se pro všechny spojení zjistí, zda prohození koncového uzlu spojení s některým z blízkých sousedů počátečního uzlu (jsou brány v úvahu uzly, které se zdrojovým uzlem přímo sousedí) nebude mít za následek zmenšení hodnoty TEL. Při tomto kroku je v algoritmu dbáno na to, že oba účastnící záměny se mohou účastnit více spojení a je tak třeba prověřit všechny změny délek, které záměna vyvolá. Menší hodnota TEL přímo odpovídá větší přehlednosti grafu, neboť se zkracuje délka hran a tedy i šance na překrytí hran. Pokud je hodnota TEL po prohození menší, jsou uzly skutečně prohozeny a vypočítá se nová hodnota TEL. Algoritmus pokračuje v činnosti dokud TEL nekonverguje. Následně je opět zobrazováno do SVG pomocí modulu visualizer.js.

7.5.3 CRSM zobrazení

Vstupním bodem pro algoritmus CSRM, jsou uzly a jejich spojení. Ve funkci *crsmVisualization* je opět nedjříve použito vytvoření objektů pro uzly a hrany se základním rozložením. Následně jsou tyto data odeslána ve formátu JSON přes websocket pro zpracování programem crsm.jl.

Po rozparsování dat se vstupuje do hlavní části algoritmu voláním funkce *CRwSM Maticove*. V této funkci dochází nejprve k inicializaci proměných potřebných k fungování algoritmu. Jednou z těchto proměnných je *DistPairwise* neboli matice vzdáleností mezi uzly, důležitá pro výpočet Fstress (viz. 6.3). Další důležitou proměnou je *IndexSet*, která obsahuje objekty představující hranové páry. Pro tyto hranové páry pak algoritmus kontroluje jejich překročení a mění rozložení uzlů. V hlavní části algoritmu se nacházejí tři klíčové funkce. První je funkce doOverlap, která zjistí zda vůbec může u hran dojít k překročení. Jedná se čistě o optimalizační prvek, jelikož samotné spouštění optimalizace je dosti drahé. Pokud nemůže dojít k překročení, pak algoritmus pokračuje dalším hranovým párem. Pokud ano, zjišťuje optimalizační funkce EdgeCrossPen (viz. 6.1), zda k překročení skutečně došlo a pokud ano, tak vrací koeficienty rozdělovacího prostoru u^*a γ^*, které jsou nutné v

třetí klíčové funkci. Tato funkce se jmenuje *SMACOFwithEdgeCrossing*. Ta řeší minimalizační problém 6.4, jehož výsledkem jsou nová rozložení uzlů, která odpovídají omezení ze strany funkce Fstress a omezení penalizační funkce. Funkci lze vidět na ukázce 7.7.

```
function SMACOFwithEdgeCrossing(Y::Array,DisOrigin::Array,WOrigin::Array,
U::Matrix,r::Vector,Edge_List::Vector,Vec::Vector,IndexSet::Vector,L::Array)
::Array
    m=length(IndexSet)
    p=Model(Ipopt.Optimizer)
    set_silent(p)
    n=length(Y[:,1])
    @variable(p, XX[1:n-1,1:2])
    @variable(p, Norms[1:m,1:4])
    XX=[transpose(Y[1,1:2]); XX]
    for i=1:m
        if Vec[i]==0
            continue
        end
        @constraint(p, Norms[i,1:4].>=zeros(4))
        A=(transpose([XX[Edge_List[IndexSet[i][1]][1],:]
        XX[Edge_List[IndexSet[i][1]][2],:]]))
        @constraint(p, Norms[i,1:2].>=(-A)*(U[i,:])+(r[i]+1)*(ones(2)))
        B=(transpose([XX[Edge_List[IndexSet[i][2]][1],:]
        XX[Edge_List[IndexSet[i][2]][2],:]]))
        @constraint(p, Norms[i,3:4].>=(B)*(U[i,:])-(r[i]-1)*(ones(2)))
    end
    @expression(p, Stress,(n*(n-1)/2)+2(XX,L)-(2)*(XX,Y,WOrigin,DisOrigin))
    @expression(p, Crossing,sum((Vec[i]/2)*(sum(Norms[i,j].^2 for j=1:4))
    for i=1:m if Vec[i]!=0))
    @objective(p, Min, Stress+Crossing)
    JuMP.optimize!(p)
    return JuMP.value.(XX)
end
```

Výpis 7.7: Stress majorization

Ve funkci, kromě její důležitosti pro algoritmus jako takový, lze vidět použití řešení minimalizačního problému pomocí programování, což je původní důvod volby jazyku Julia.

Algoritmus po získání nového rozložení kontroluje, jak moc se liší nové rozložení od starého. Pokud je výsledná odchylka dostatečně malá, pak algoritmus končí a prohlásí poslední hodnoty matice X za řešení, ke kterému zkonvergoval. Hodnoty matice X jsou následně převedeny do formátu JSON a poslány přes websocket do clientské části. Zde jsou ve funkci *vizCRSM* vytvořeny nové objekty dle získaného rozložení. Tyto objekty jsou pak jako v ostatních zobrazeních předány modulu visualizer.js k zobrazení do SVG.

Během implementace bylo otestováno mnoho verzí algoritmu s různými matematickými koeficienty, které výrazně měnily vlastnosti algoritmu. Nakonec je přikloněno k verzi, která poskytuje vyváženou hodnotu rychlosti a kvality zobrazení.

7.6 Využití asociativních polí

V implementaci algoritmů a struktur pro uchovávání dat jsou využívána asociativní pole. Díky tomuto přístupu se mnohonásobně zrychluje vyhledávání v datových strukturách, kdy rychlost vyhledání v asociativním poli je oproti cyklickému prohledávání konstantní. Při přidávání prvků do těchto polí se využívá unikátních klíčů jednoznačně identifikujících dané prvky. To umožňuje následovně snadné použití při hledání daného prvku. Rychlý přístup k prvkům je potřebný zejména při vytváření propojení, neboli vytváření hranových objektů mezi uzly a také při algoritmu záměny uzlů.

7.7 Využití SVG DOM

SVG DOM je využíván hlavně v části animací. Při konkrétní akci uživatele je vyhledán DOM element, kterého se akce týká, a následně s využitím dat tohoto elementu je akce provedena. Například se může jednat o přesun elementu v zobrazovací ploše tahem myši. Tento přístup umožňuje provádět změny, jako právě například animace elementů, bez toho, aby bylo nutné provádět operace nad celým zobrazením. Akce se provede pouze nad konkrétním elementem a prvky s ním přímo spojenými (například hrany). Toto umožňuje značné výkonnostní urychlení oproti případu, kdy by podobná akce vyžadovala překreslení celého zobrazení. SVG umožňuje řadu funkcí spojených s javascriptem, právě díky tomu, že se jedná o html elementy. Dalším důležitým využitím jsou OnClick akce vyvolávající přidání dalších uzlů, které jsou odkazovány kliknutým uzlem.

7.8 Funkcionalita uživatelského rozhraní

Uživatelské rozhraní nabízí několik doplňujících funkcí k zobrazení RDF dat. Je to například náhrada části URI v uzlech za prefix pro zpřehlednění grafu. Tato funkcionalita je implementována v modulu user-interface-functions.js za využití vlastnosti local storage. Namapování URI na prefix, zvolený uživatelem, je ukládáno do local storage a je tak zachován při opětovném spuštění aplikace.

Dalším prvkem je vyhledání počátečního uzlu a jeho sousedů pomocí funkce setStartPoint, která iniciuje dotaz na daný uzel a jeho okolí, a následně předá k základnímu zobrazení.

Kapitola 8

Testování

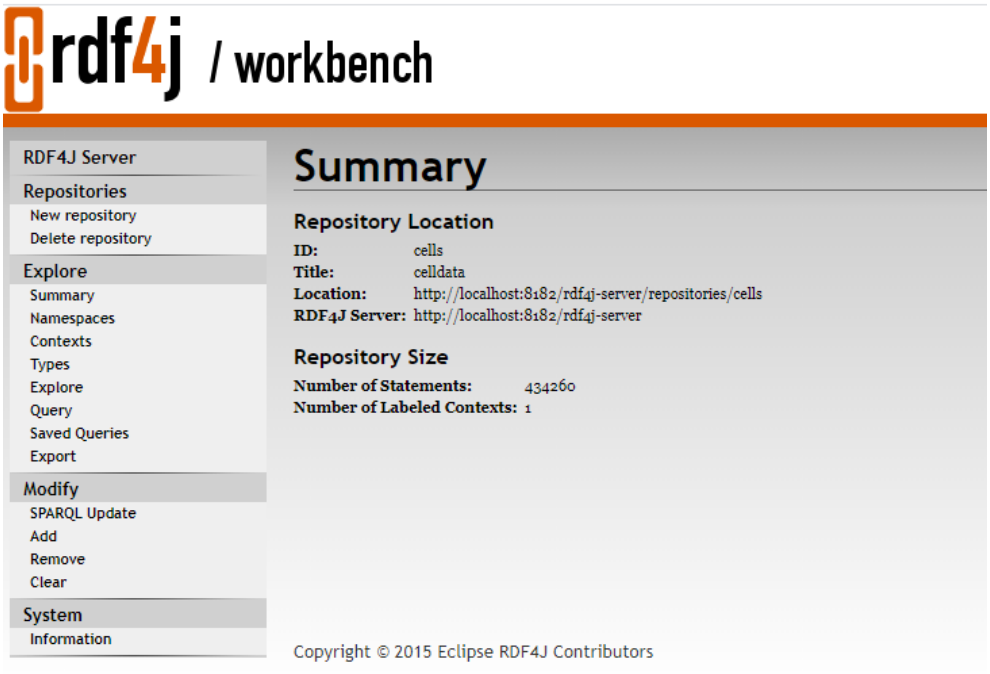
V této kapitole jsou popsány metody testování cílů práce. V první části je to výkonnost aplikace pro zobrazení velkých datových setů. Pak porovnání kvality jednotlivých zobrazení z hlediska uživatelské přívětivosti. Jedním z druhů testování je také ověření funkcionality aplikace.

8.1 Datové sady

Hlavním zdrojem dat pro testování jsou volně dostupné datové sady (datasety) ze stránek DBpedia [3]. Výhodou těchto datasetů je jejich velikost a možnost testování velkých objemů trojic. Datasety rovněž obsahují velké množství predikátů, tedy jejich hustota propojení je značná. Díky velké velikosti zdrojů je možné testovat libovolně velké, následně aplikačně omezené, počty dat. Příklad velkého datasetu nahraného na RDF4j server 8.1. Tento dataset obsahuje reálná data týkající se buněk a poskytuje jak dostatečnou velikost statisíců trojic, tak velkou hustotu propojení a poskytuje tak možnost testovat porovnání zobrazení s hustým propojením oproti slabému propojení. Z datasetů byl používán především zmíněný dataset buněk a set sociálních síti, protože dostatečně splňovaly všechny požadavky na testování.

Dataset	Velikost v KB
Biologická data o buňkách	19231
Sociální sítě	3760
geonames$_l inks_e n$	64118
homepages$_e n$	84863

Tabulka 8.1: Tabulka datasetů



Obrázek 8.1: Dataset v RDF4J

8.2 Výkonnost

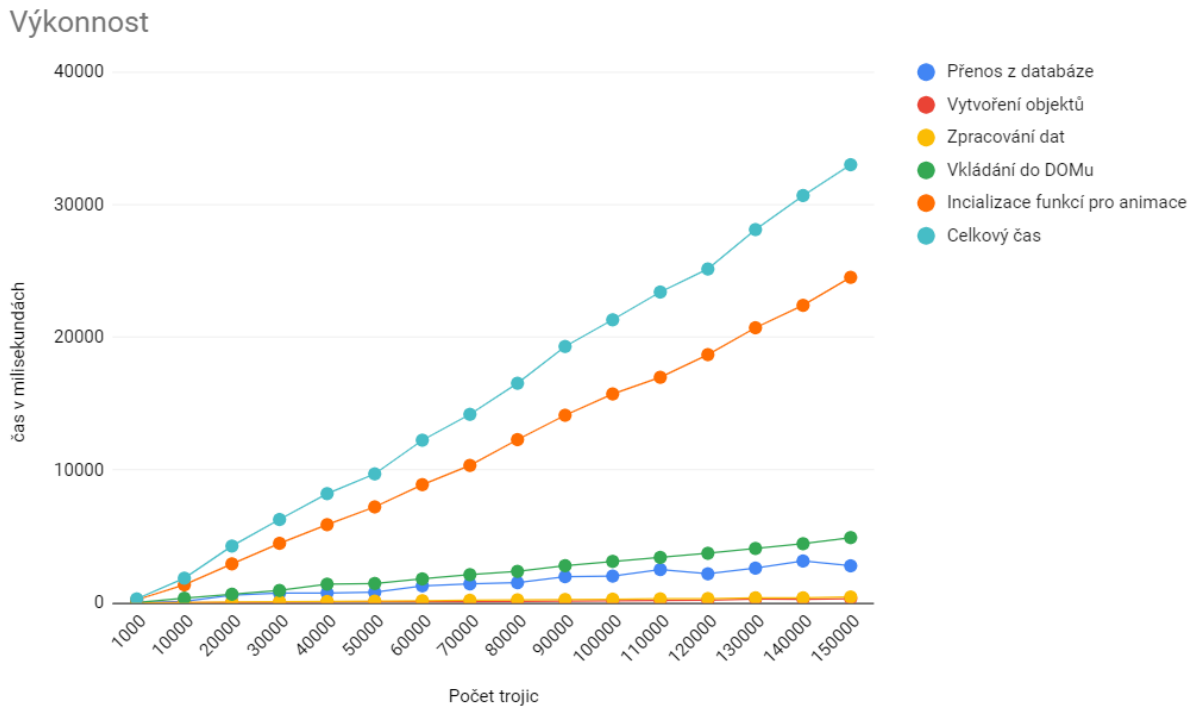
Testování výkonnosti je prováděno na datasetu o velikosti statisíců trojic. Kromě standardního zobrazení dat s plnou hustotou (vysokým počtem zobrazených predikátů), je testováno i zobrazení s nižší, aplikačně omezenou, hustotou. Toto umožní poskytnout pohled na rozdíl mezi schopností zobrazovat hustě propojená a řídce propojená data. V rámci tohoto testování jsou měřeny následující atributy. Čas přenosu dat z databáze, kdy tato informace potenciálně vypovídá o reálném maximálním oběmu dat, který má vůbec smysl zobrazovat. Jestliže je doba přenosu příliš velká, klesá celkový význam zkoumání dalšího zobrazení dat. Dalším atributem je čas vytvoření iniciálních datových objektů, které jsou dále použity při přípravě dat k zobrazení. Poté je měřen čas přípravy dat k zobrazení. Tato část je důležitá z hlediska toho, že je dobré vědět, jakou dobu zabere zpracování dat ještě předtím, než je aplikováno vkládání do SVG. Poté následuje měření tvorby a vkládání SVG elementů do SVG DOM. Tato část je jedna z nejdůležitějších, protože jedním z cílů práce je zjistit informace o použití SVG k účelu zobrazení RDF dat. Dalším atributem měření je doba inicializace a aplikování animačních funkcí nad SVG. Tyto funkce provedou první transformaci zobrazení tak, aby zobrazení obsahovalo všechny uzly. Při větším objemu dat, je téměř nemožné, získat z grafu jakoukoliv intuitivní informaci bez použití ZOOMu nebo pohybu v grafu. Následně je vypočítán celkový čas od zahájení zobrazování, tedy přenosu dat, po vykreslení dat na obrazovku. Důležitost tohoto času je značná, neboť vypovídá obecně o výkonnosti aplikace vzhledem k objemu dat. Tento čas je pak přidán jako poslední časový atribut. Kromě časových atributů je měřen celkový počet elementů vložených do DOM. Tento počet je důležitý z hlediska indikování zátěže na prohlížeč při vyšších objemech elementů v DOM. Posledním typem měření, který souvisí s částí interaktivity je zjištění FPS

při akcích prováděných nad grafem. Jak již bylo zmíněno, pokud se nelze v grafu aktivně pohybovat a přibližovat, je jeho intuitivita značně omezená.

Na první tabulce a grafu je vidět výsledek testování výkonnosti na silně propojených datech. Testování probíhalo až po 150000 trojic, kdy cílem bylo zjistit možnosti SVG na vysokých kvantitách dat. Jednotlivé časy výkonání zmíněných atributů v milisekundách jsou zobrazeny v závislosti na jednotlivých objemech.

Počet Trojic	Přenos z databáze	Vytvoření objektů	Zpracování dat	Vkládání do DOM	Inicializace funkcí pro animace	Celkový čas	Celkový počet SVG elementů
1000	31	1	4	29	236	300	3081
10000	105	17	23	356	1364	1865	28665
20000	587	35	66	655	2943	4286	59042
30000	737	40	82	937	4490	6286	89596
40000	751	67	107	1414	5897	8236	117458
50000	803	87	133	1465	7236	9724	145785
60000	1286	90	166	1809	8913	12264	176893
70000	1436	86	200	2128	10362	14212	209609
80000	1525	118	229	2374	12308	16554	243857
90000	1980	141	257	2805	14142	19326	277179
100000	2021	158	280	3130	15748	21336	310448
110000	2515	177	305	3429	17003	23429	337195
120000	2193	193	327	3740	18710	25164	371418
130000	2621	290	383	4103	20738	28135	407731
140000	3159	275	382	4460	22426	30702	443511
150000	2796	324	445	4922	24537	33024	481305

Tabulka 8.2: Tabulka výkonnosti na silně propojených datech



Obrázek 8.2: Graf výkonnosti na silně propojených datech

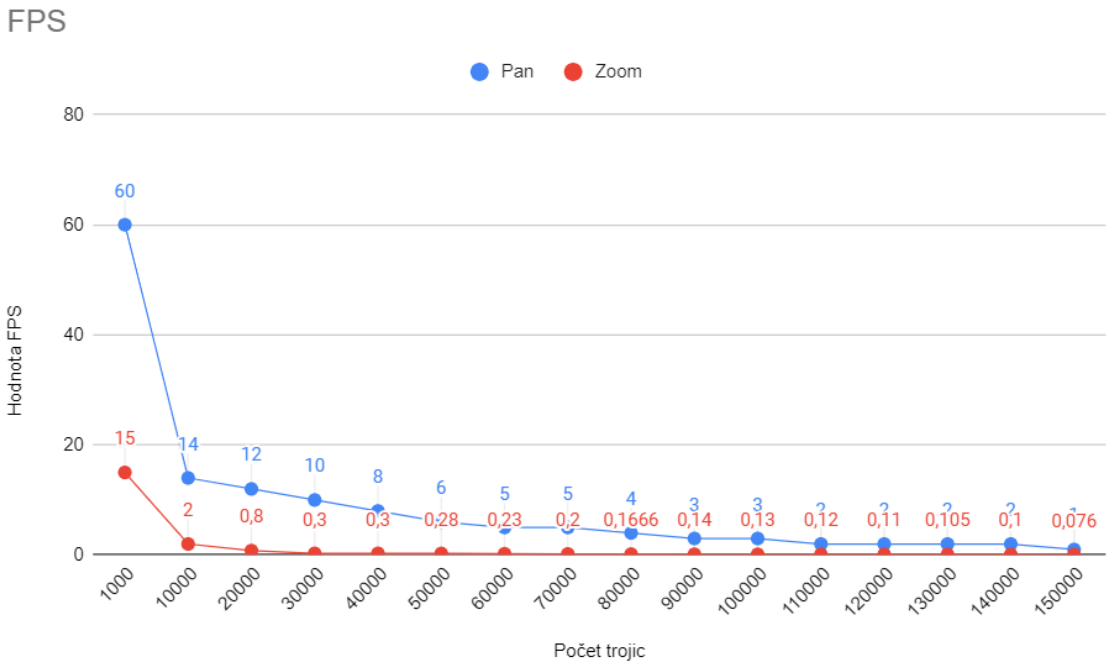
Na grafu lze vidět trend růstu délky průběhu jednotlivých etap běhu aplikace. V souladu s informacemi z tabulky je vidět, že zpracování dat zabírá zanedbatelnou část celkového času. Časy vkládání do SVG DOMu a přenosu z databáze rostou větším tempem nicméně zůstvájí v únosné míře. Atributem, který zásadně narůstá, je inicializace skupiny funkcí zajišťujících animace a provedení prvotní animace Fit a Center, která zajistí, že je pohled

zaměřen na celé zobrazení grafu. Poslední atribut, který má zásadní vliv na výkonnost, je počet SVG elementů v SVG DOMU po dokončení zobrazení. V prohlížeči při dosažení počtu elementů přesahujících 200 000 dochází k zásadnímu poklesu výkonnosti, což lze vidět na ukázce 8.3.

Dalším měřeným aspektem kromě atributů týkajících se iniciálního zobrazení je následná práce s aplikací a Frames Per Second (dále FPS) neboli počet zobrazených snímků za sekundu při akcích uživatele. Tyto hodnoty jsou naměřeny v následující tabulce. Jedná se o FPS při akci PAN neboli posouvání v SVG ploše a FPS při akci ZOOM neboli přiblížení v ploše.

Počet Trojic	Pan	Zoom
1000	60	15
10000	14	1.3
20000	12	0.6
30000	10	0.3
40000	8	0.3
50000	6	0.28
60000	5	0.23
70000	5	0.2
80000	4	0.17
90000	3	0.14
100000	3	0.13
110000	2	0.12
120000	2	0.11
130000	2	0.105
140000	2	0.1
150000	2	0.076

Tabulka 8.3: Tabulka FPS při velké hustotě propojení

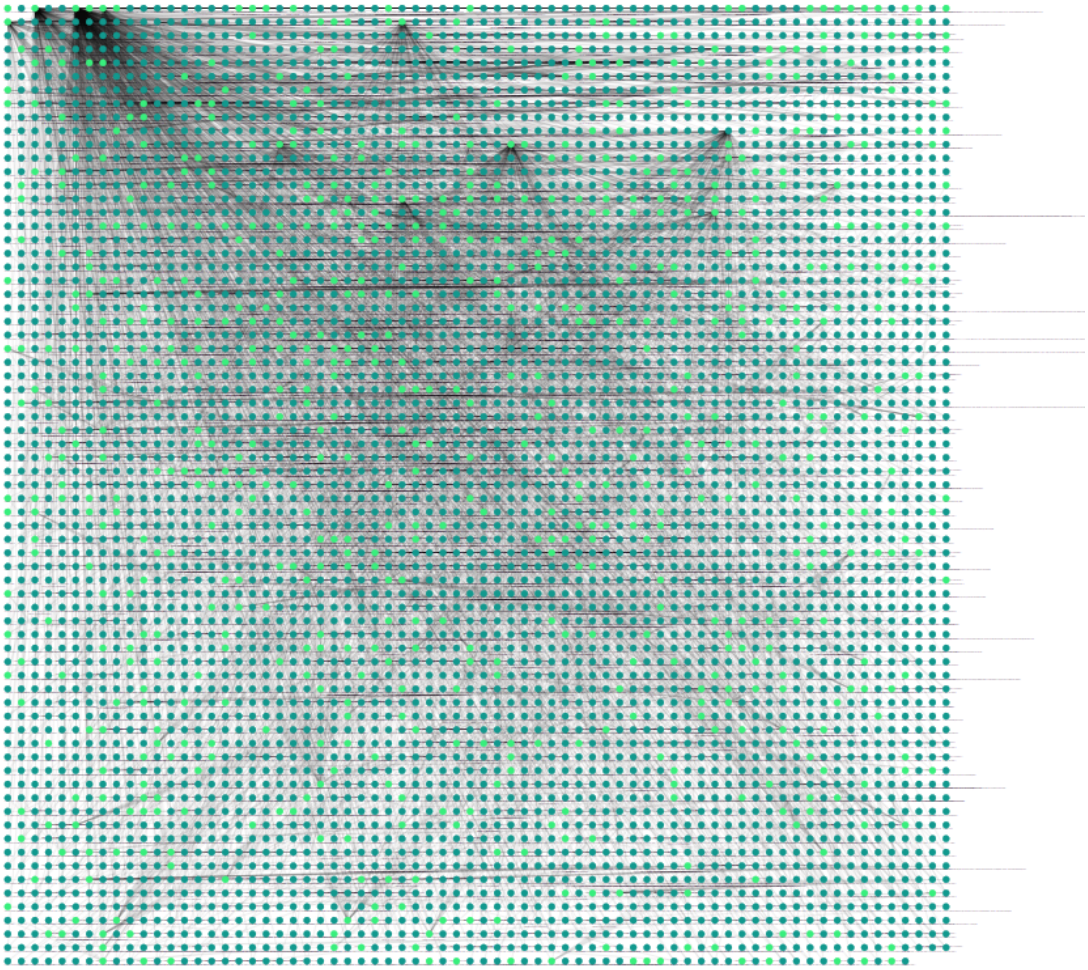


Obrázek 8.3: Graf FPS při velké hustotě propojení

Z výsledků vyplývá, že vysoké počty svg elementů mají velmi negativní vliv na FPS aplikace a použitelnost přesahující inicální zobrazení. Za hranicí 100000 trojic výkon přesahuje hranici menší než 1,5 průměrného FPS a aplikace se stává obtížně použitelnou za účelem detailnějšího procházení dat.

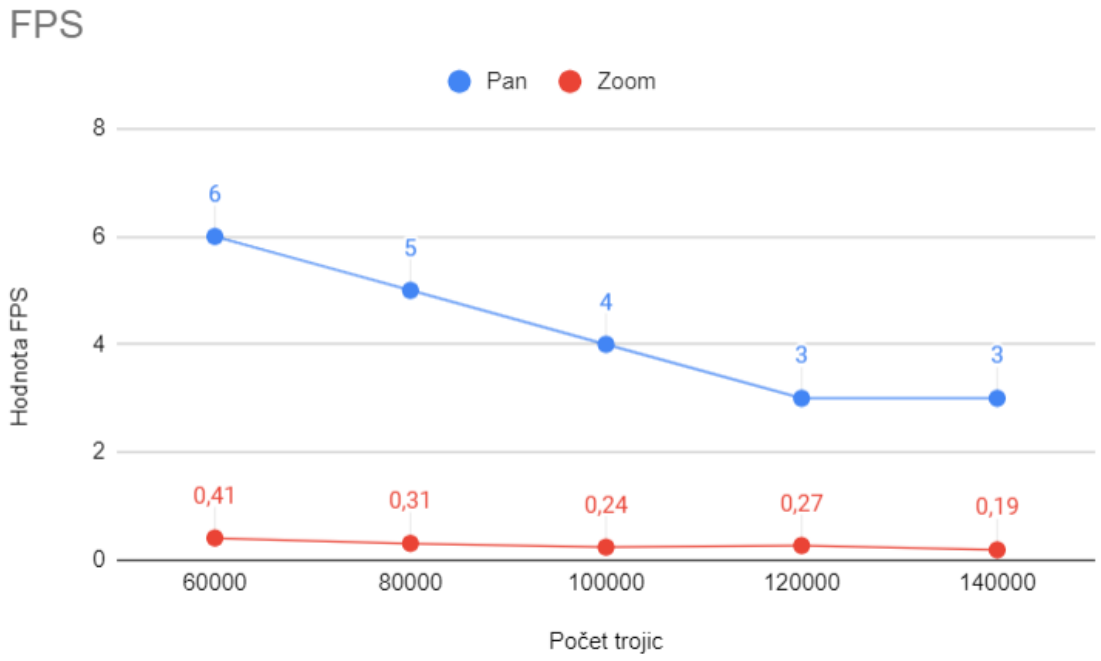
Při zkoumání výsledků je nutno brát v potaz fakt, že akce PAN dosahuje různých FPS při různých úrovních zoomu. Uvedené hodnoty jsou brány při akci PAN na inicálním zoomu. Při jiném přiblížení mohou být nižší. Hlavním ukazatelem FPS výkonnosti je tak akce ZOOM jako bod nejvyšší zátěže aplikace a nejnižšího výkonu.

Na ukázce 8.4 je vyobrazení grafu o velikosti 10000 trojic. Je vidět, že hustota propojení na běžných reálných RDF datech je značná. Aspekt propojení pak má zejména vliv na výkonnost animací.



Obrázek 8.4: Náhled grafu s vysokým počtem trojic

V následující tabulce a grafu je testování výkonnosti animací na slabě propojených datech. Tento test byl proveden na užším vzorku testovacích trojic, slouží pro porovnání výkonu animace se silně propojenými daty a ukázkou vlivu hustoty propojení na výkon FPS.



Obrázek 8.5: Graf FPS na slabě propojených datech

Celkově lze vidět, že nižší hustota propojení má důsledek na výkonnost aplikace. Křivky následují podobný trend jako u hustě propojených dat, ale celkové hodnoty FPS jsou vyšší. Výsledkem tohoto testu je informace, že velké množství hran je náročnější pro animace nad grafem než srovnatelné celkové množství trojic s převažujícím počtem uzlů.

8.2.1 Srovnání výkonnosti v prohlížečích

Předmětem testování bylo porovnání výkonnu mezi prohlížeči s rozdílným přístupem k zobrazování. Google chrome Verze 89.0.4389.90 a Mozilla Firefox verze 86.0.1. Nejdříve byly testovány časy provedení jednotlivých částí aplikace. Výsledky lze vidět v tabulce 8.4.

Počet Trojic	Přenos z databáze	Vytvoření objektů	Zpracování dat	Vkládání do DOM	Inicializace funkcí pro animace	Celkový čas	Celkový počet SVG elementů
10000	105	16	28	216	861	1226	28665
20000	203	28	67	447	1931	2676	59042
30000	264	59	93	698	2941	4055	89596
40000	349	69	141	898	3917	5374	117458
50000	424	78	145	1137	4933	6717	145785

Tabulka 8.4: Tabulka výkonnosti z prohlížeče Firefox

Na výsledcích z tabulky je možné vidět, že časy přenosů, vytvoření objektů, zpracování a vkládání dat jsou podobné s výsledky získanými z prohlížeče Chrome 8.2. Co se liší je čas incializace funkcí pro animace a prvotní zobrazení. Výsledek naznačuje, že Firefox je schopný rychleji zobrazit SVG a provést transformace nad zobrazením. Tento výsledek podporuje i následující tabulka testování FPS 8.5.

Z výsledků a porovnání s Chromem 8.3 vychází, že firefox dosahuje lepších FPS při provádění SVG transformací. Hodnoty akce ZOOM jsou znatelně rychlejší než u Chromu. Hodnoty akce PAN sice na první pohled vypadají vyšší než u Chromu, nicméně jak již bylo zmíněno v části testování FPS, tak akce PAN má jiné hodnoty při různých Zoomech a v

Počet Trojic	Pan	Zoom
10000	17	5
20000	7	3
30000	4.2	1.54
40000	3.12	1.34
50000	2.2	1.22

Tabulka 8.5: Tabulka FPS z prohlížeče Firefox

případě firefoxu byl PAN testován u náročnějších zoomů, takže i tato hodnota je reálně lepší než u chromu. Firefox dosahuje lepších výkonnostních výsledků při práci s SVG než Chrome.

8.2.2 Výkonnost alternativních zobrazení

Další testovanou částí je výkonnost dalších zobrazení a jejich porovnání. Vzhledem ke složitosti pokročilejších zobrazení probíhá toto testování na menším vzorku trojic. Výměnou za menší výkonnost je pak vyšší kvalita zobrazení. Předmětem testování je časová hodnota trvání algoritmů. Prvním testovaným algoritmem je záměna uzlů s účelem redukce celkové délky hran. Výsledek testování je popsán v následující tabulce 8.6.

Počet Trojic	Trvání algoritmu (ms)
300	397
600	451
900	1642
1200	5156
1500	8919
1800	8956
2100	13859
2400	20096

Tabulka 8.6: Tabulka výkonnosti zobrazení se záměnou uzlů

Z výsledků lze vidět, že algoritmus záměny uzlů stoupá v náročnosti poměrně rychle. Jeho náročnost je exponenciální, avšak pro zpřehlednění menších grafů trvá přijatelnou dobu. U algoritmu CR-SM je situace složitější. Složitost algoritmu se neodvíjí pouze od počtu uzlů, ale zejména úrovně jeho zauzlení.

Toto může být vidět na ukázce 8.11, kdy pro 20 trojic z hustě propojeného vzorku dat, trvá výsledek přibližně 2 sekundy. Ukázka z 50 trojic s větší hustotou překročení pak trvá už desítky sekund, ale jiná ukázka s hodnotou 40 trojic, ale menší četností zauzlení pak trvá zase přibližně 2 sekundy.

Pro méně překřížený graf s více trojicemi, tedy může být algoritmus rychlejší, než pro graf s menším počtem trojic, ale větším zauzlením. Měření jeho výkonnosti je tedy obtížné.

Na testu jsou vzorky s menší hodnotou trojic a vyšší hodnotou zauzlení. Obecně se dá říct, že je algoritmus CR-SM velmi výpočetně náročný a není příliš vhodný pro použití v interaktivní aplikaci pro zobrazování větších grafů za požadavku rychlého zobrazení. Na druhou stranu, pro případ užití, kdy čas nehraje zásadní roli, algoritmus poskytuje kvalitní způsob zobrazení s maximální redukcí zauzlení.

8.2.3 Závěr výkonnostního testování

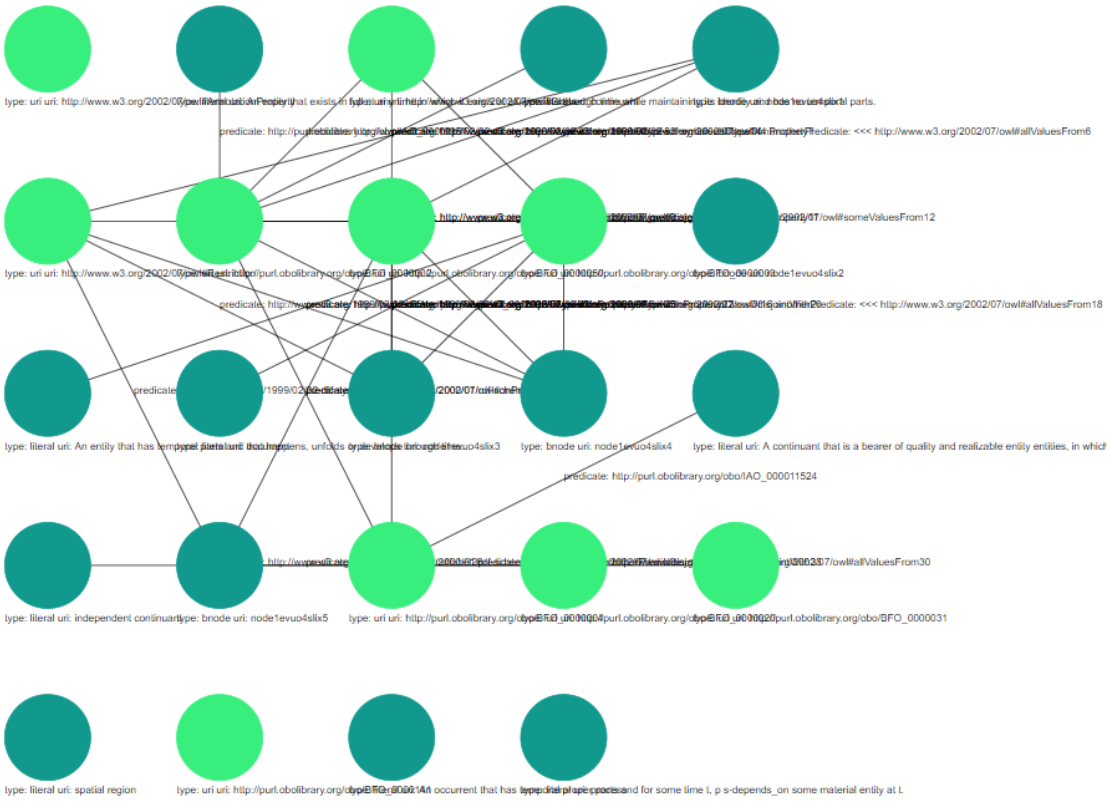
Výsledek výkonnostního testování podává několik informací. Jednou z nich je, že výkon aplikace klesá pod uživatelsky vhodnou hodnotu 60 fps poměrně rychle. Již po překročení 10000 uzlů, se v případě prohlížeče Goggle Chrome, aplikace dostává s akcí zoom na hodnotu 1.3 FPS. Akce Zoom je výkonnostně nejnáročnější a je tedy použita jako referenční hodnota, průměrné hodnoty FPS jsou však vyšší. Další infromací je, že při velkých objemech trojic překračujících hranici 100000 vzniká v DOM velké množství objektů, které klade nároky na prohlížeč i mimo akce spojené s aplikací a omezuje jeho FPS. Aplikace se ukazuje jako kvalitní, rychlý nástroj pro hodnoty trojic nižší než 10000. Překročení této hranice vyžaduje trpělivější používání aplikace. Zajímavou informací je, že Firefox dosahuje lepších výsledků pro SVG transformace a aplikace působí v tomto prohlížeči rychleji.

8.3 Kvalita zobrazení

V této části je řešeno porovnání a zhodnocení kvalit jednotlivých zobrazení. Je hodnocena především intuitivita, přehlednost a hodnota výpovědní informace grafu. Porovnávány jsou všechny tři zobrazení z kapitoly návrhu.

8.3.1 Základní zobrazení

Prvním zobrazením k porovnání je základní zobrazení do mřížky. Na ukázce je vidět toto zobrazení pro menší počet prvků. Hned zpočátku je zřejmé, že překročení hran způsobuje zhoršenou přehlednost grafu.

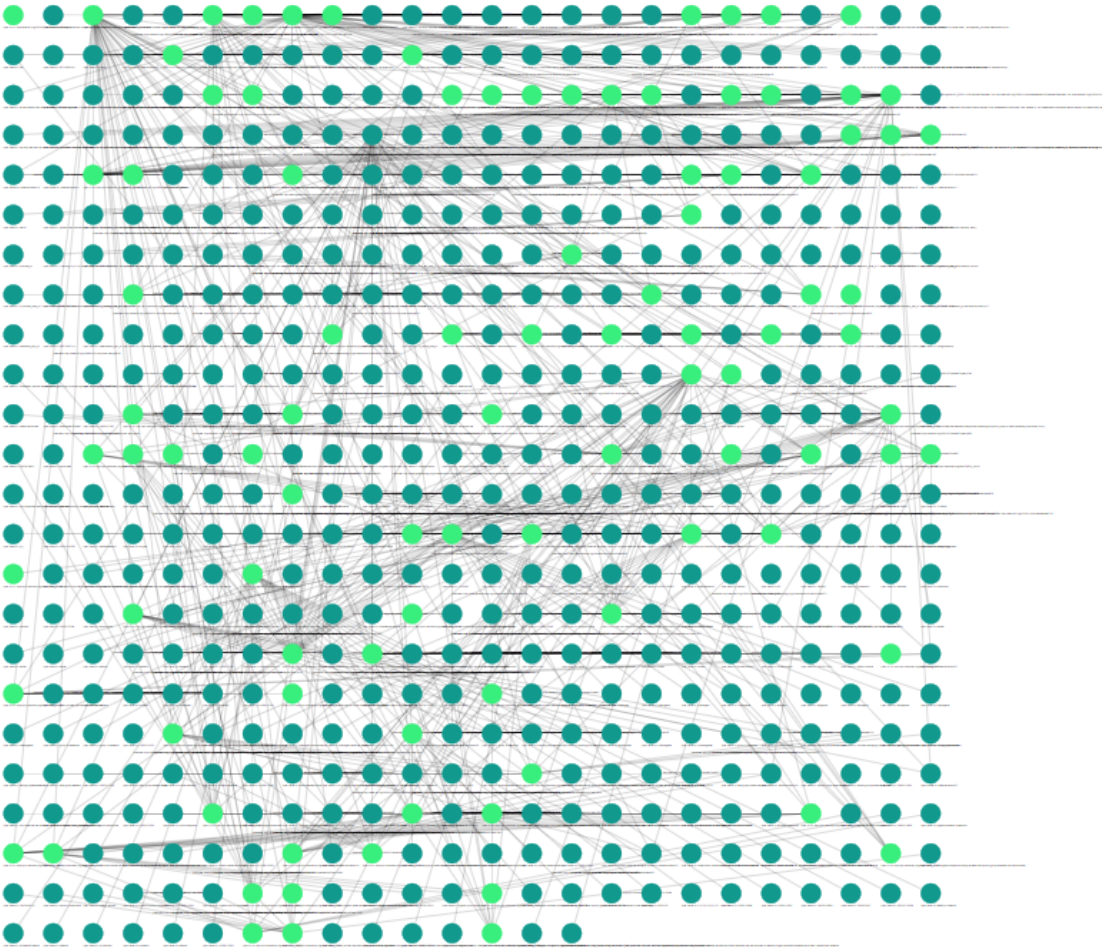


Obrázek 8.6: Ukázka základního zobrazení s malým počtem prvků

Problém zhoršené přehlednosti řeší možnost reorganizace grafu pomocí přesunu uzlů a hran. Aplikaci tohoto přístupu můžeme vidět v praxi na ukázce 8.7. Za použití akce Pan, Zoom a posunutí uzlu při držení a tahem myši došlo ke změně grafu a zvýšení jeho přehlednosti. Tento přístup dobře funguje na grafech menšího objemu avšak na grafech s větším počtem prvků by taková reorganizace byla velmi náročná.

Obrázek 8.7: Ukázka základního zobrazení s manuálním přesunem prvků

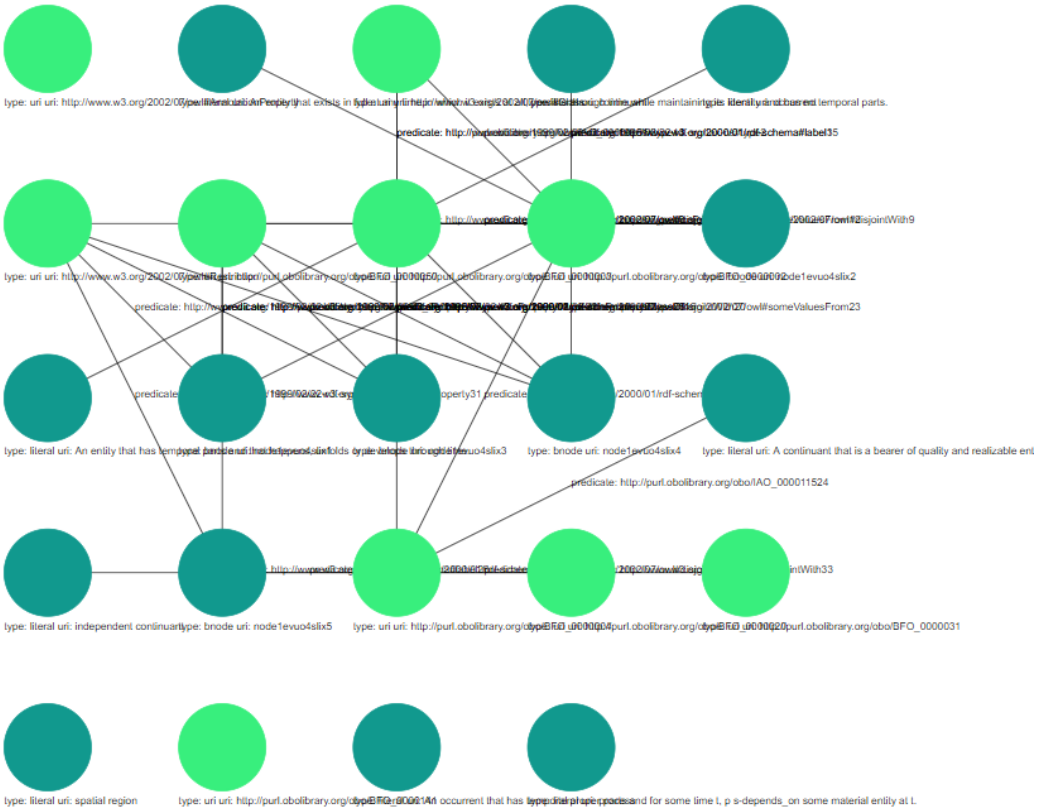
Na poslední ukázce lze vidět graf s větším počtem prvků. Na takovémto grafu, už je zvýšená obtížnost reorganizace a přehlednost klesá. Zatímco první zobrazení poskytuje největší výkon, jeho kvalita není značná a bez možnosti manuální reorganizace může graf při velkém počtu překročení hran působit nepřehledně. Celkově se jedná o nejrychlejší avšak nejméně přehledný typ grafu, kvůli vysoké míře překročení hran.



Obrázek 8.8: Ukázka základního zobrazení s velkým počtem prvků

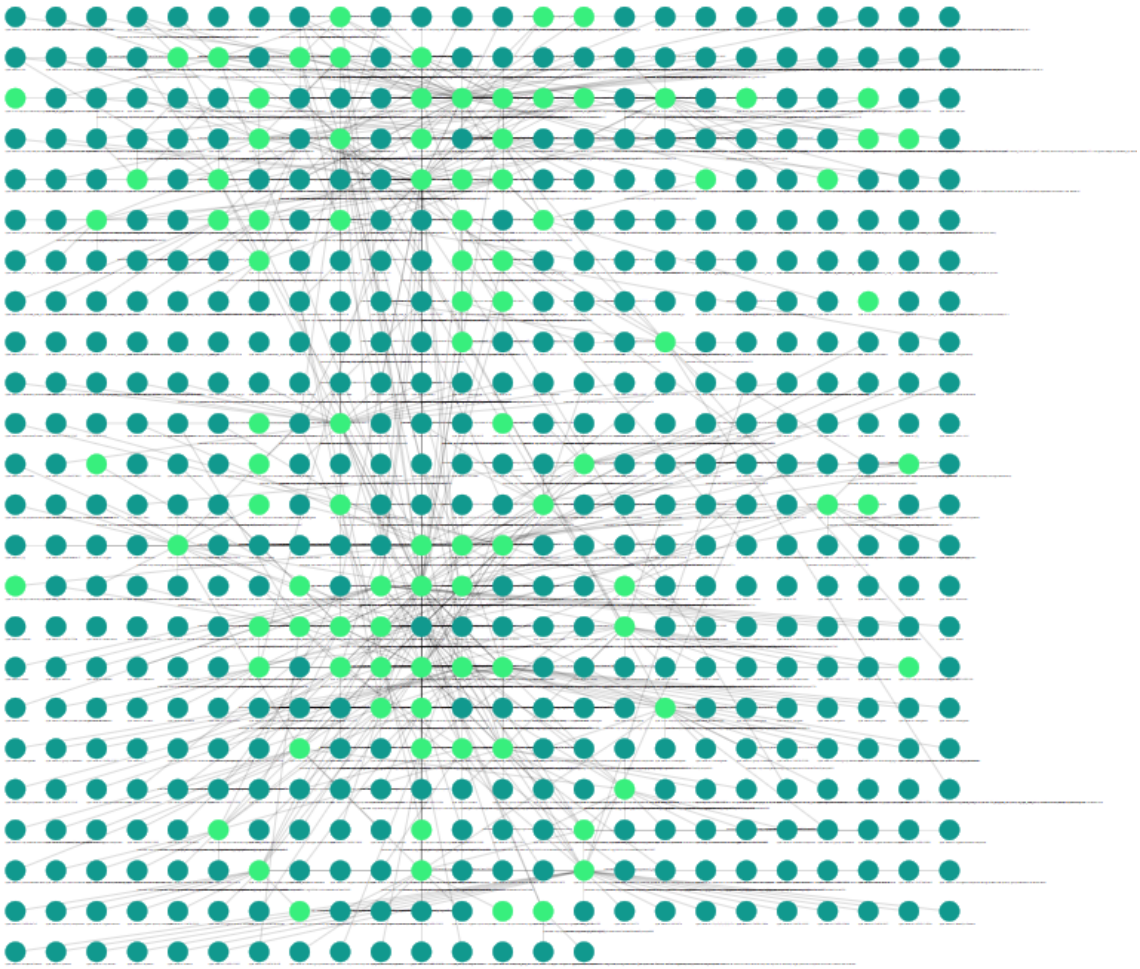
8.3.2 Zobrazení se záměnou uzlů

Zobrazení provádí záměnu uzlů s cílem snížit celkovou délku hran. Uzly jsou zaměňovány dokud nadále nedochází k další minimalizaci délky. Toto zobrazení vede ke zmírněnému počtu překročení hran, zmenšení délky hran a tedy k větší přehlednosti grafu. Na ukázce s malým počtem vzorků je vidět v porovnání se základním zobrazením zmírnění počtu překročení a redukce délky.



Obrázek 8.9: Ukázka zobrazení se záměnou uzlů s malým počtem prvků

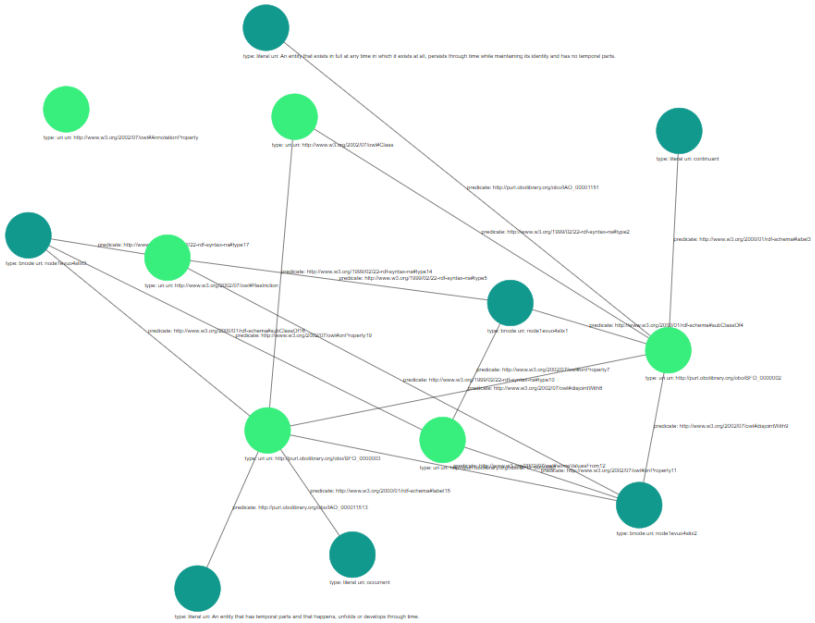
Podobně tomu je u vzorku s větším počtem prvků, který lze vidět na další ukázce. Zobrazení koncentruje uzly s velkým množstvím hran do určitých míst. Na ukázce je vidět centra s uzly vyššího propojení. V porovnání s ukázkou lze vidět, že graf se záměnou uzlů působí více uspořádaně a přehledně. Cenou za lepší zobrazení je zvýšení časové náročnosti provedení zobrazení.



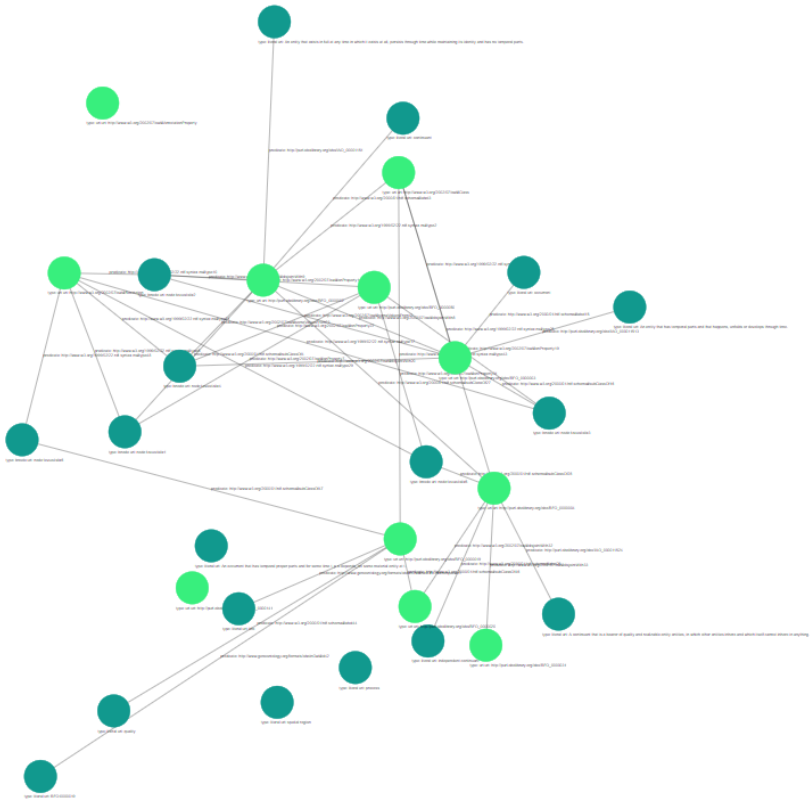
Obrázek 8.10: Ukázka zobrazení se záměnou uzlů s velkým počtem prvků

8.3.3 Zobrazení pomocí metody CR-SM

Tato metoda je ze všech tří výpočetně nejnáročnější, avšak nabízí nejkvalitnější druh zobrazení. Eliminací překročení dochází k vytvoření velmi přehledných grafů. V kombinaci s animacemi se tak jedná o nejhodnotnější typ zobrazení z hlediska intuitivity, praktičnosti a výpovědní hodnoty. Nicméně náročnost výpočtu je tak značná, že i malé datasety mohou zabrat desítky sekund před výsledným rozložením. I přes různé optimalizace algoritmus trvá delší dobu u zapletených datasetů. Obtížnost se neodvíjí pouze od počtu trojic, ale značně od míry překročení a zapletení grafu. Graf s mnoha překročeními, trvá delší dobu i u minimálního počtu trojic. Použití zobrazení CR-SM lze vidět na ukázkách malého a většího datasetu.



Obrázek 8.11: Ukázka zobrazení CR-SM s malým počtem prvků



Obrázek 8.12: Ukázka zobrazení CR-SM s velkým počtem prvků

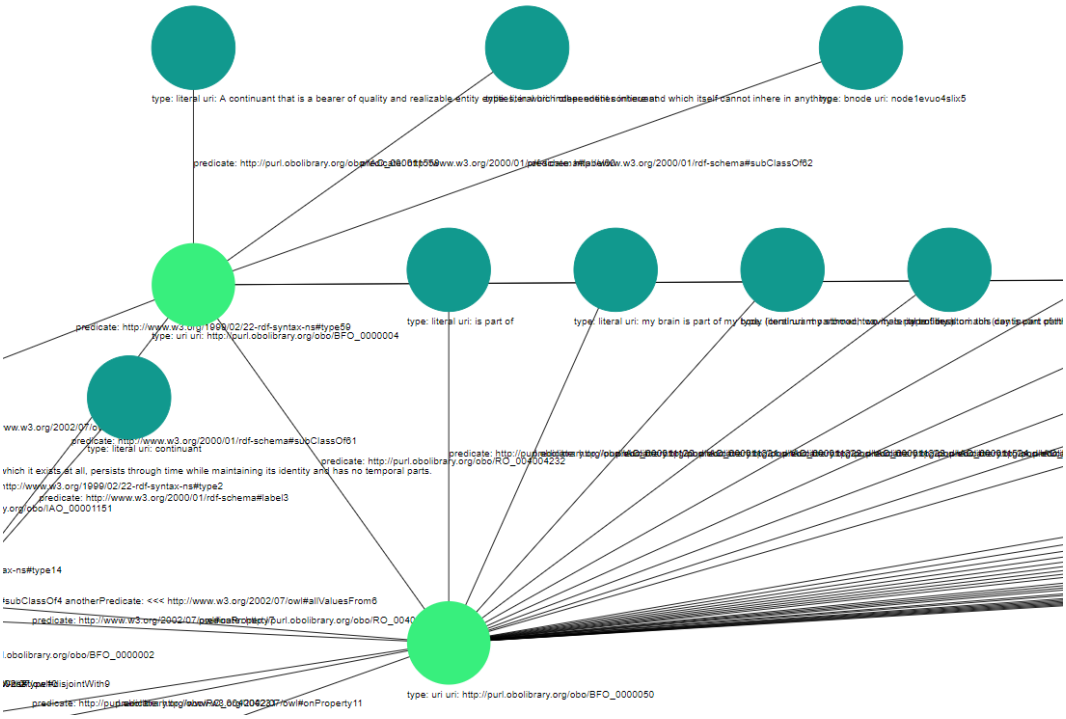
Výsledkem testování tohoto zobrazení je verdikt, že algoritmus CR-SM není příliš vhodný pro použití v interaktivní aplikaci, kde je očekáván rychlý výsledek. Na druhou stranu redukuje zauzlení sítě mnohem lépe než algoritmus záměny a poskytuje kvalitní výsledky. Je tedy vhodný pro použití, kde nejsou vysoké nároky na čas zpracování.

8.3.4 Zobrazení dalších uzlů po vytvoření iniciálního grafu

Jedná se o doplňkový typ zobrazení, ke každému ze zmíněných zástupců. Při kliknutí na uri libovolného uzlu se načtou a zobrazí všechny další uzly, které odkazuje pomocí predikátu. Tyto nové uzly se zobrazí za hranicí obalovacího čtverce grafu. Výsledek tohoto zobrazení je vidět na ukázce. Tento doplňkový typ zobrazení je vhodný k procházení menších vzorků dat s možností postupného rozšíření na základě potřeby. Pro menší datasety se jedná o nejlepší a nejkvalitnější způsob zobrazování grafu. Na ukázkách je vidět postupné zobrazení sousedů ve dvou krocích.

Obrázek 8.13: Ukázka zobrazení sousedních uzlů č.1

Obrázek 8.14: Ukázka zobrazení sousedních uzlů č.2



Obrázek 8.15: Ukázka zobrazení sousedních uzlů č.3

8.4 Porovnání SVG s Canvas

V této části je provedeno srovnání vyobrazení RDF dat pomocí SVG a Canvasu. Odkazuji se zde na práci kolegy, která byla zaměřena na vyobrazení grafových databází pomocí technologie Canvas [28]. Základním rozdílem mezi technologiemi je, že SVG vyobrazuje vektorovu grafiku pomocí DOM objektů, které jsou definované XML dokumentem a Canvas kreslí rastrovou grafiku. Z tohoto vyplývá, že změny prováděné nad zobrazenou grafikou pomocí technologie Canvas znamenají nové vykreslení obrázku. Zároveň je nutné akce prováděné nad grafikou, jako například přesun uzlu, detekovat pomocí určení pozice kliknutí a následného získání cíleného objektu z uložené struktury. Tento přístup je náročnější oproti SVG, které vykreslený objekt drží jako DOM element a je tak snadná jeho identifikace a následná změna paramterů bez nutnosti ovlivňovat všechny prvky v síti. Charakter SVG DOMu umožňuje snadnou spolupráci s javascriptem při zajišťování interaktivity aplikace.

Canvas na druhou stranu disponuje rychlejšími časy vykreslení a nezatěžuje prohlížeč vysokou kvantitou DOM elementů, která může při velkých grafech dosahovat počtu stovek tisíc. V takovémto případě SVG přístup značně omezuje výkonnost prohlížeče a FPS klesají na nízké hodnoty.

Co se týče animací, Canvas drží podobnou hodnotu FPS při jakékoliv animaci, neboť to znamená překreslení celého obrazu. SVG má (viz. část testování 8.3) jiné výsledky u různých animací. Při pohledu na srovnání 10000 trojic pro obě technologie, dosahuje SVG horších FPS při akci ZOOM pro prohlížeč Chrome. U prohlížeče Firefox se SVG zdá výkonnější, avšak celková zobrazovací výkonnost je v průměru u SVG v porovnání s Canvas nižší. Oblast kde SVG dosahuje vyšší výkonnosti jsou však animace na vysoké úrovni zoomu, kdy SVG nemusí překreslovat všechny prvky.

Při shrnutí výsledků je možné říct, že SVG vyniká v části interaktivity a možností využití výhod SVG DOM, jako jsou práce s javascriptem a lokální změny zobrazení. Canvas oproti tomu disponuje stabilnějším a místy výkonnějším zobrazením celkového grafu. Obecně se dá říci, že obě technologie ztrácí plynulost zobrazení při podobných hranicích. Na první přejaté tabulce z práce [28], pro srovnání výsledků s Canvasem, lze vidět výsledky z části, Tabulka 7.2: Časy potřebné k načtení dat ze serveru do klienta v milisekundách.

Počet trojic	Doba přenosu	Doba zpracování	Doba vkládání	Celkový čas
10000	293,2	312,6	45,1	650,9
20000	874,1	623,8	69,8	1567,7
30000	1701,1	925,3	93,8	2720,2
40000	2922,3	1241,6	125,4	4289,3
50000	4721,8	1550,0	134,0	6405,8
60000	6761,7	1861,3	138,1	8761,1
70000	9431,6	2177,7	163,2	11772,5
80000	12258,3	2490,5	190,3	14939,1
90000	16351,0	2806,5	215,1	19372,6
100000	20584,6	3125,9	234,5	23945,0

Tabulka 8.7: Časy potřebné k načtení dat ze serveru do klienta v milisekundách. [28]

Při srovnání s SVG hodnotami 8.2 je vidět, že jsou celkové časy SVG vyšší než u Canvas. Hlavním důvodem rozdílu je část aplikace s incializací funkcí pro animace a prvotním zobrazením, což trvá déle než u Canvas. Canvas poskytuje větší výkon pro samostatné

zobrazení. V další přejaté tabulce, lze vidět výsledky z části práce, Tabulka 7.5: Časy potřebné k provedení všech částí programové smyčky v milisekundách.

Počet trojic	Doba přenosu	Doba zpracování	Doba vkládání	Celkový čas
100	16,67	16,67	16,67	16,66
500	20,00	16,67	16,68	16,67
1000	24,65	19,64	16,67	16,67
2500	56,72	45,99	23,12	16,67
5000	118,51	96,42	43,47	22,07
7500	170,07	145,83	61,82	32,49
10000	223,26	195,18	78,94	42,43
20000	445,78	388,26	163,87	93,63
40000	1029,45	792,47	452,67	184,47
80000	1836,63	1343,75	829,41	367,12

Tabulka 8.8: Časy potřebné k provedení všech částí programové smyčky v milisekundách. [28]

Na porovnání výkonnostních hodnot SVG 8.3 s Canvasem 8.8 lze usoudit, že canvas dosahuje v průměru lepších FPS hodnot. Avšak výsledný rozdíl není dostatečný na plynulý běh aplikace při vyšších hodnotách trojic. Obě aplikace překračují hranici plynulého používání aplikace na 10000 trojicích. Při vyšších hodnotách musí uživatel počítat s klesajícím výkonem a delší odezvou změn zobrazení.

Kapitola 9

Závěr

V první části diplomové práce bylo předmětem nastudování a pochopení konceptů grafových databází, formátu RDF a vektorové grafiky v prohlížečích. Výsledkem bylo ověření, že lze tyto koncepty spojit za účelem vytvoření aplikace pro zobrazení dat grafových databází v podobě interaktivního grafu. Dále byla zjištěna široká podpora pro SVG ve všech hlavních prohlížečích, neboli se potvrdila vhodnost volby vektorové grafiky k zobrazování objektů v prostředí prohlížečů.

Po nastudování existujících řešení pro zobrazování RDF dat v podobě grafu bylo zjištěno, že současná řešení nabízejí pouze omezené možnosti bez interaktivity. Často se jedná pouze o statické formáty jako např. PNG nebo jen textový výstup pro další zpracování grafickou knihovnou. Některé také nejsou implementovány jako webová technologie, ale jako spustitelný program. Nebyl tedy nalezen vhodný přístup k řešení vizualizace RDF dat, který by odpovídal dříve zmíněným požadavkům.

Při práci na úkolech semestrálního projektu byla probrána a vyzkoušena možnost vizualizace dat pomocí kombinace SVG a Javascriptu v prohlížeči. Výsledkem bylo potvrzení, že lze prakticky využít vektorové grafiky v prohlížečích pro vykreslení RDF dat v podobě grafu.

Při výběru technologií bylo dbáno na výběr správných technologií pro jednotlivé účely. Byly zvoleny ověřené a vhodné technologie pro část serveru, část klienta a komunikaci mezi nimi. Dále byla na základě nastudování možností zvolena speciální řešení pro komplexní typ zobrazení. Jedná se o technologie RDF4J pro data, Apache Tomcat jako Java servlet container, REST API pro získávání dat ze serveru, JSON jako formát přijímání dat, RDF-XML pro nahrávání dat a matematický jazyk Julia pro realizaci algoritmu zobrazení.

V poslední části semestrálního projektu byl vytvořen návrh samotné aplikace. Návrh probírá základní funkcionalitu aplikace, architekturu aplikace, komunikaci mezi jednotlivými částmi a zahrnuje popis zobrazení dat v prostoru, neboli rozložení bodů grafu. Kromě návrhu jednotlivých zobrazení je taky popsán koncept rozšiřování zobrazených dat nebo iniciální zobrazení databáze. Výsledkem této části je návrh pro intuitivní, interaktivní aplikaci pro zobrazení dat grafové databáze ve správné podobě.

Poté došlo k fázi implementace návrhu a vytvoření dané aplikace. Během implementace docházelo k menším dynamickým změnám pro přizpůsobení aplikace na základě nových poznatků, objevených během práce.

Následovala část testování, kdy byly otestovány limity aplikace, funkcionalita a cíle práce. Během testování docházelo k optimalizaci kódu. Výsledkem testování bylo, že aplikace je schopna zobrazit i velké statisícové objemy trojic, avšak s velkým poklesem výkonu. Aplikace je stále použitelná i při těchto velkých objemech, ale nenabízí plynulý, rychlý

běh. Důvodem pro klesající výkon je vysoká náročnost pro obrazové transformace SVG při zobrazeních přesahujících desetitisíce trojic. Kromě toho vzniká při velkých objemech dat velké množství DOM elementů, jejichž počet ovlivňuje od určitého bodu výkonnost prohlížeče i bez použití animací. Na druhou stranu aplikace při objemech dat menších než deset tisíc běží v dobrých hodnotách FPS a poskytuje možnost interaktivního prohlížení RDF grafů. SVG ukazuje své výhody právě při snadné manipulaci s DOM elementy, která je využívána Javscriptem pro realizaci animací, či dalších doplňujících funkcí. SVG je tedy vhodná technologie pro vizualizaci RDF dat, avšak při značných datových objemech ztrácí výkonnost.

Literatura

[1] *RDF2SVG* [online]. [cit. 2020-27-11]. Dostupné z: https://github.com/rhizomik/redefer-rdf2svg.

[2] *Ontology-visualization* [online]. 2018 [cit. 2020-25-11]. Dostupné z: https://github.com/usc-isi-i2/ontology-visualization.

[3] *DBpedia* [online]. 2020 [cit. 2021-20-2]. Dostupné z: https://databus.dbpedia.org/ontologies/purl.obolibrary.org/obo--cl--owl.

[4] *Julia 1.5 Documentation* [online]. 2020 [cit. 2020-2-12]. Dostupné z: https://docs.julialang.org/en/v1/.

[5] *SVG* [online]. 2020 [cit. 2020-25-11]. Dostupné z: https://caniuse.com/?search=svg.

[6] Alexey Melnikov. *The WebSocket Protocol* [online]. 2011 [cit. 2019-6-12]. Dostupné z: https://tools.ietf.org/html/rfc6455.

[7] Barry Zane. *Semantic Graph Databases: A worthy successor to relational databases* [online]. 2016 [cit. 2020-12-11]. Dostupné z: https://www.dbta.com/BigDataQuarterly/Articles/Semantic-Graph-Databases-A-worthy-successor-to-relational-databases-114569.aspx.

[8] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, Gavin Carothers. *Terse RDF Triple Language* [online]. 2014 [cit. 2020-15-11]. Dostupné z: https://www.w3.org/TR/turtle/.

[9] Erik Dahlström, Patrick Dengler, Anthony Grasso, Chris Lilley, Cameron McCormack, Doug Schepers, Jonathan Watt, Jon Ferraiolo, (FUJISAWA Jun), Dean Jackson. *SVG Document Object Model* [online]. 2011 [cit. 2020-23-11]. Dostupné z: https://www.w3.org/TR/SVG11/svgdom.html.

[10] Fabien Gandon, Guus Schreiber, Dave Beckett. *RDF 1.1 XML Syntax* [online]. 2014 [cit. 2020-18-11]. Dostupné z: https://www.w3.org/TR/rdf-syntax-grammar/.

[11] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, June 2013. ISBN 978-1-491-93200-1.

[12] Ivan Herman, Ben Adida, Manu Sporny, Mark Birbeck. *RDFa 1.1 Primer - Third Edition* [online]. 2015 [cit. 2020-18-11]. Dostupné z: https://www.w3.org/TR/rdfa-primer/.

[13] Jesús Barrasa. *RDF Triple Stores vs. Labeled Property Graphs* [online]. 2017 [cit. 2020-12-11]. Dostupné z: https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/.

[14] Kolektiv autorů. *RDF Grapher* [online]. [cit. 2020-27-11]. Dostupné z: http://www.ldf.fi/service/rdf-grapher.

[15] Kolektiv autorů. *Semantic web made easy* [online]. [cit. 2020-13-11]. Dostupné z: https://www.w3.org/RDF/Metalog/docs/sw-easy.

[16] Kristin P. Bennett, Cagri Ozcaglar, Amina Shabbeer. *Crossing Minimization within Graph Embeddings* [online]. 2012 [cit. 2020-25-01]. Dostupné z: https://arxiv.org/pdf/1210.2041.pdf.

[17] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, May 2007. ISBN 978-0-596-52926-0.

[18] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Niklas Lindström. *A JSON-based Serialization for Linked Data* [online]. 2018 [cit. 2020-18-11]. Dostupné z: https://www.w3.org/2018/jsonld-cg-reports/json-ld/#data-model.

[19] Neo4j. *Graph database* [online]. [cit. 2020-12-11]. Dostupné z: https://neo4j.com/developer/graph-database/.

[20] Nicholas Humfrey. *Easy RDF* [online]. [cit. 2020-27-11]. Dostupné z: https://www.easyrdf.org/converter.

[21] RDF Working Group. *RDF* [online]. 2014 [cit. 2020-15-11]. Dostupné z: https://www.w3.org/RDF/.

[22] RDF working group W3C. *RDF 1.1* [online]. 2014 [cit. 2020-18-11]. Dostupné z: http://www.w3.org/TR/rdf11-new/.

[23] rdf4j. *The Eclipse RDF4J Framework* [online]. 2020 [cit. 2020-30-11]. Dostupné z: https://rdf4j.org/about/.

[24] Richard Cyganiak, NUI Galway, David Wood, Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax* [online]. 2014 [cit. 2020-15-11]. Dostupné z: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[25] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures. Chapter 5. Representational State Transfer (REST)* [online]. 2000 [cit. 2020-2-12]. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.html.

[26] Sumit Purohit, Mark Harrison. *RDF AND JSON-LD UseCases* [online]. 2014 [cit. 2020-18-11]. Dostupné z: https://www.w3.org/2013/dwbp/wiki/RDF_AND_JSON-LD_UseCases/.

[27] The W3C SPARQL Working Group. *SPARQL 1.1 Overview* [online]. 2013 [cit. 2020-21-11]. Dostupné z: https://www.w3.org/TR/sparql11-overview/.

[28] Tomáš, J. *Vizualizace rozsáhlých grafových dat na webu* [online]. 2020 [cit. 2021-18-03]. Dostupné z: https://www.fit.vut.cz/study/thesis-file/22593/22593.pdf.

[29] W3C working group. *SVG* [online]. 2004 [cit. 2020-23-11]. Dostupné z: https://www.w3.org/Graphics/SVG/About.html.

Příloha A

Obsah CD

- Zdrojové soubory k aplikaci (source_files.zip)
- Testovací data (test_data.zip)
- Zdrojové soubory k Latexu (latex_files.zip)
- Technická zpráva (technicka_zprava.pdf)
- Popis k aplikaci (README)