VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ADMINISTRAČNÍ ROZHRANÍ SYSTÉMU PRO EXTRAKCI INFORMACÍ

ADMINISTRATION INTERFACE OF AN INFORMATION EXTRACTION SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAKUB GONGOL**
AUTHOR

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

BRNO 2018

Zadání diplomové práce/20648/2017/xgongo01

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Gongol Jakub, Bc.**
Obor: Informační systémy
Téma: **Administrační rozhraní systému pro extrakci informací**
Administration Interface of an Information Extraction System
Kategorie: Informační systémy

Pokyny:

1. Seznamte se s problematikou extrakce informací z WWW a se souvisejícími projekty.
2. Prostudujte současné technologie pro tvorbu webových aplikací se zaměřením na platformu Java. Rovněž prostudujte technologie sémantického webu, zejména jazyky RDF a OWL.
3. Navrhněte architekturu administračního rozhraní aplikace pro extrakci informací umožňující správu uživatelů, zadávání a správu extrakčních úloh, zdrojů dat a další operace podle konzultací s vedoucím.
4. Implementujte navrženou aplikaci s využitím vhodných technologií. Implementujte také možnost interaktivní editace extrakčních úloh.
5. Proveďte integraci vytvořeného řešení s existujícím systémem pro extrakci informací a proveďte testování na vhodné možině dat.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Swicegood, T.: Programming Node.js, O´REILLY, 2012
- 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
- Pokorný, J.: Webové uživatelské rozhraní nástroje pro extrakci informací. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií.
- Buba, V.: Extrakce informací z webu založená na ontologiích. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Burget Radek, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Práce se zabývá problematikou extrakce informací z webových stránek. Cílem je návrh a vytvoření webové aplikace administračního rozhraní systému pro extrakci informací na platformě Java. Aplikace poskytuje editor pro specifikaci zadaní extrakčních úloh ve formě interaktivních grafů, možnost načítání ontologií ze souboru a automatické vykreslení grafu na základě výběru z načtených ontologických vlastností. Řešení zajišťuje integraci s nástrojem FITLayout.

Abstract

This thesis covers the subject of information extraction from Web. The main objective is the design and implementation of an administration interface for an extraction system implemented as web application on Java platform. The application provides an editor for extraction tasks specification in the form of interactive graphs. It includes the possibility to upload and process an ontology from a file and generate graph according to selected ontology properties. The solution ensures integration with the FITLayout tool.

Klíčová slova

Sématnický web, ontologie, extrakce informací z webu, administrační rozhraní, interaktivní grafy, OWL, RDFS, Java, React.

Keywords

Semantic Web, ontology, information extraction, administration interface, interactive graphs, OWL, RDFS, Java, React.

Citace

GONGOL, Jakub. *Administrační rozhraní systému pro extrakci informací.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Administrační rozhraní systému pro extrakci informací

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana doktora Radka Burgeta. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Gongol
16. května 2018

Poděkování

Na tomto místě bych rád poděkoval panu doktorovi Radkovi Burgetovi za cenné rady a podněty při vypracování této práce.

Obsah

Kapitola 1

Úvod

Tato práce se zabývá problematikou extrakce informací z webových stránek. Hlavním cílem je návrhnout a vytvořit administrační rozhraní systému na extrakci informací, nikoli extrakce samotná. Vytvářená webová aplikace na platformě Java umožní uživatelům, skrze své rozhraní, zejména zadávání extrakčních úloh ve formě interaktivních grafů.

Většina lidí v dnešní době považuje internet za médium, kde si jednoduše vyhledá a přečte to, co je pro ně důležité. Někdy však může být obtížné najít v téměř nekonečném množství stránek pro nás důležité informace. Naštěstí je člověk inteligentní a dokáže si mezi nalezenými daty vytvořit souvislosti a vzájemné vazby podle aktuálního kontextu. Ale jak jsou na tom stroje? I přes vysokou vyspělost technologií je zde situace horší. Vzhledem k velkým objemům dat a absenci samostatného rozhodovaní je automatizace jakékoliv úlohy na webu velmi obtížná.

Za účelem strojové automatizace získávání informací z webových stránek byl zaveden sémantický web, jehož myšlenkou je povznést klasický web na inteligentnější a intuitivnější úroveň pro obsloužení požadavků jeho uživatelů. Tento pohled na web - tedy jako na data s daným významem - umožňuje strojům rozpoznávat výskyt jednotlivých entit a vazeb mezi nimi. Jedním ze způsobů, jak vyjádřit sémantiku dat je jazyk RDF - obecný rámec pro popis zdrojů, metadatový model a základní stavební kámen sémantického webu [4].

Diplomová práce je členěna do devíti kapitol. Za tímto úvodem bude čtenář v druhé kapitole seznámen se sémantickým webem a s problematikou extrakce informací z WWW a s nástroji k tomu potřebnými. Bude zde přiblížen konceptuální model RDF, také RDF schéma, které je považováno za jednoduchý ontologický jazyk. Čtenář bude seznámen také s ontologiemi a rodinou jazyků OWL, společně s jejich systémem klasifikace. Kapitola 3 představí metody extrakce informací z webu a existující nástroje, ze kterých bude vyvíjená aplikace vycházet. Následně se v kapitole 4 dočteme o technologiích a nástrojích využitých pro vývoj webové aplikace na platformě Java. V páté kapitole se kromě návrhu celkové architektury a popisu stěžejních částí aplikace dočteme také o podrobnější specifikaci požadavků na výslednou aplikaci. Kapitola šestá přibližuje způsob implementace nejdůležitějších funkcí systému, jako je kreslení grafů, načítání ontologií ze souboru, automatické vykreslení grafu z vybraných vlastností, dále implementace akcí „zpět“ a „vpřed“ nebo export úlohy. Sedmá kapitola se zabývá návrhem grafického uživatelského rozhraní systému. Zaměříme se zde především na hlavní obrazovku aplikace, popis jednotlivých prvků a jejich význam pro systém a způsob řádného použití. Předposlední kapitola objasní způsob integrace s nástrojem FITLayout a testování klíčových funkcí aplikace. V závěru, kapitola 9, je provedeno celkové shrnutí provedené práce s možnostmi dalšího vývoje.

Kapitola 2

Sémantický web

Pojem sémantický web ustanovil Tim Berners-Lee ředitel organizace W3C, která jej volně definuje jako rozšíření klasického webu dokumentů na "Web tvořený daty". V tomto pojetí webu neexistují propojení pouze na úrovní dokumentů, ale jsou zde vazby a vztahy i mezi daty vyskytujícími se v těchto dokumentech. Díky těmto vazbám je možné provádět strojové zpracování. [1]

Na tato data můžeme nahlížet jako na data, se kterýma jsme zvyklí pracovat v relačních databázích. Jedná se tedy o libovolné datumy, nápisy a titulky, čísla nebo jejich části, chemické prvky a vlastností, apod. [14]

Významným cílem sémantického webu je umožnit počítačům na základě strojově čitelných dat plně využít jejich potenciál a také potenciál webu jako otevřeného prostoru pro sdílení informací. Takto vyjádřil svou vizi sémantického webu sám Tim Berners-Lee:

„I have a dream for the Web in which computers become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A "Semantic Web", which makes this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The "intelligent agents"people have touted for ages will finally materialize.“ [3]

Sémantický web využívá technologie jako RDF, OWL, SPARQL, SKOS, které umožňují vytvářet na webu datová uložiště, slovníky a zapisovat pravidla pro manipulaci s těmito daty. SPARQL[1] představuje dotazovací jazyk nad daty uloženými ve formátu RDF. SKOS[2] je používám jako formát pro prezentaci a publikování slovníků [14]. Jazyky RDF a OWL budou podrobněji rozebrány dále, viz 2.1 a 2.3

2.1 Resource Description Framework

RDF tvoří základ pro zpracování metadat, která popisují data obsažená na webu. Zajišťuje schopnost vzájemné spolupráce a součinnost mezi aplikacemi pro výměnu informací, které jsou pro stroje nejen čitelné, ale kterým také rozumí. Otevírá se tak možnost automatizace zpracování webových zdrojů. Text a ukázkové příklady v této sekci vychází ze specifikace [9].

Pro RDF se nachází využití ve vícero oblastech. Například v oblasti vyhledávání zdrojů lze dosáhnout vetší vyspělosti vyhledávačů nebo lze popsat a rozlišit obsah včetně vztahů určitých stránek, což zvyšuje možnosti katalogizace a hodnocení obsahu.

[1] https://cs.wikipedia.org/wiki/SPARQL
[2] https://cs.wikipedia.org/wiki/Simple_Knowledge_Organization_System

RDF představuje rodinu specifikací navržených konsorciem W3C. Jedním z jeho cílů je umožnit specifikovat sémantiku dat na základě XML standardizovaným způsobem. Proto bylo potřeba definovat vhodný mechanismus umožňující použití pro libovolnou doménu informací, aniž by předem ovlivňoval jejich význam.

Framework používá systém tříd stejně jako objektově orientované programování (OOP) a většina modelovacích systémů. Kolekce tříd pro specifický účel nebo doménu se nazývá schéma. Samotné třídy jsou hierarchicky organizovány a jsou rozšiřitelné definováním podtříd, které je upřesňují.

Základ jazyka RDF tvoří model pro reprezentaci pojmenovaných vlastností a jejich hodnot. Tyto hodnoty si můžeme představit jako atributy zdrojů, v běžném pojetí jako dvojice klíč - hodnota. RDF vlastnosti mohou vyjadřovat také vztahy mezi jednotlivými zdroji a tím pádem se může RDF model podobat ER diagramu.

V terminologii objektově orientovaného paradigmatu bychom zdroje označili za objekty a jejich vlastnosti pak za instanční proměnné.

2.1.1 Datový model

Datový model představuje prostředek, kterým lze vyjádřit RDF výrazy, aniž by musela být dodržována nějaká předepsaná syntaxe. Takový model se skládá z 3 typů objektů:

- **Zdroje** - Za zdroj označujeme vše, co je popsáno pomocí RDF výrazů. Zdrojem tedy může být celá webová stránka, případně nějaká její část jako například specifický HTML nebo XML element uvnitř zdrojového dokumentu, nebo dokonce kolekce stránek, tzn. celý web. Nemusí se nutně jednat o objekt přístupný přes internet, ale může jít např. o tištěnou publikaci. Všechny zdroje jsou vždy pojmenovány pomocí URI a mohou obsahovat libovolné ID.

- **Vlastnosti** - Vlastnost je specifický aspekt, charakteristika nebo vztah použitý pro popis zdroje. Každá vlastnost má konkrétní význam a zároveň definuje svoje povolené hodnoty, typy zdrojů, které může popisovat a vztahy k ostatním vlastnostem.

- **Tvrzení** - Konkrétní zdroj společně s pojmenovanou vlastností a hodnotou pro tento zdroj nazýváme tvrzení. Tyto 3 jednotlivé části tvrzení nazýváme popořadě subjekt, predikát a objekt. Objektem tvrzení, tedy hodnotou vlastnosti, může být jiný zdroj určený URI nebo jen prostý text - literál (takový text je přímý zápis hodnoty, která není dále vyhodnocována procesorem RDF) či primitivní datový typ definovaný v XML.

2.1.2 Příklady

V následujícím textu si ukážeme nějaké příklady vět spolu s jejich grafickou reprezentací odpovídajících tvrzení. Pro tuto reprezentaci použijeme orientovaný graf, kde uzly (znázorněny elipsou) představují zdroje a hrany pojmenované vlastnosti. Uzly obsahující literály budou zakresleny jako obdelník. Samotné vlastnosti zdroje budou nyní pro jednoduchost odkazovány jako podstatná jména. Následující příklady jsou převzaty ze specifikace RDF[9].

Jako příklad uvažujme tuto jednoduchou větu:

Ora Lassila je tvůrce zdroje http://www.w3.org/Home/Lassila.

Podle popisu datového modelu výše, může být věta rozdělena na tyto 3 části:

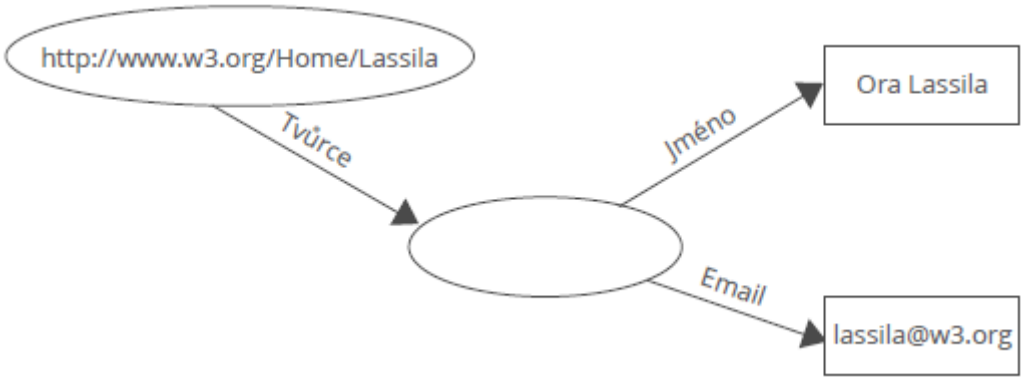
- **Subjekt** (Zdroj) - http://www.w3.org/Home/Lassila
- **Predikát** (Vlastnost) - Tvůrce
- **Objekt** (literál) - „Ora Lassila“

Grafické znázornění diagramem pak může vypadat takto:

Obrázek 2.1: Grafické znázornění uvedené věty. Zdroj obrázku [5].

Je potřeba si uvědomit, že směr šipek v diagramu je důležitý. Hrana vždy vychází ze subjektu a směřuje k objektu tvrzení.

V případě, že bychom chtěli trochu rozvést charakteristiky vlastnosti zdroje, mohli bychom uvažovat větu: Jedinec se jménem Ora Lassila a emailem lassila@w3.org, je tvůrcem http://www.w3.org/Home/Lassila. V takovém případě se z vlastnosti zdroje stává strukturovaná entita. V RDF je taková entita reprezentována jako další zdroj, jelikož však tento zdroj není pojmenován, je v diagramu zobrazen jako prázdná elipsa, viz níže:



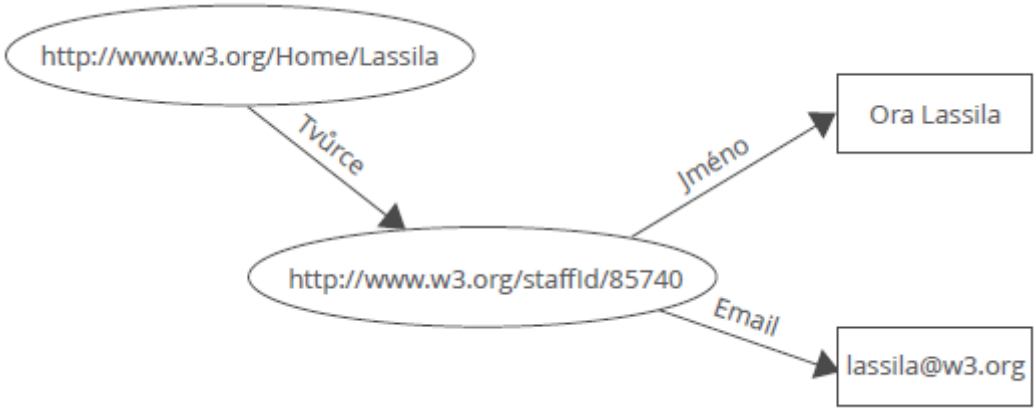
Obrázek 2.2: Strukurovaná entita s anonymním zdrojem. Zdroj obrázku [5].

Strukturovaná entita nemusí mít vždy anonymní zdroj. Zdroji může být přidělen unikátní identifikátor. Tento identifikátor se většinou odvíjí od návrhu databáze. V našem případě se bude jednat o jednoznačný, číselný identifikátor zaměstnance, např. 85740. URI odkazující tento zdroj pak může vypadat takto: http://www.w3.org/staffId/85740.

Nyní můžeme napsat dvě věty: Jedinec vedený jako zaměstnanec s identifikací 85740 se jmenuje Ora Lassila a má emailovou adresu lassila@w3.org. Tento jedinec vytvořil zdroj http://www.w3.org/Home/Lassila. Odpovídající RDF model bude vypadat takto:

2.1.3 Základní syntaxe

Datový model poskytuje abstraktní a konceptuální rámec pro definici a používání metadat. Je zapotřebí pevně daná syntaxe, aby bylo možné vytvářet a vyměňovat si tato metadata. Používají se dva typy XML syntaxe: serializační a zkrácená. Serializační syntaxe umožňuje



Obrázek 2.3: Strukurovaná entita s přiděleným identifikátorem. Zdroj obrázku [5].

plně využít všechny možnosti datového modelu a to běžným způsobem. Zkrácená syntaxe naopak zahrnuje dodatečné konstrukce, které poskytují kompaktnější způsob znázornění podmnožiny modelu. Od interpretů jazyka RDF se očekává implementace jak serializační, tak zkrácené syntaxe a tudíž je možné obě zmíněné kombinovat.

Serializační syntaxe

RDF element je pouhá obálka vyznačující hranice v XML dokumentu, mezi nimiž má být obsah přímo mapován na instance datového modelu. Tento element může být vynechán, pokud lze jednoznačně odlišit definici RDF od aplikačního kontextu. Element Description obsahuje zbývající elementy zajišťující správnou skladbu tvrzení, avšak může být také použít pro pouhé uchování identifikace zdroje, který popisujeme. Běžně se stává, že se k danému zdroji vztahuje více tvrzení. V takovém případě Description poskytuje způsob, jakým zdroj pojmenovat jen jednou, a použit jej pro více tvrzení.

Součástí elementu Description může být atribut about, který určuje URI identifikaci zdroje. Pokud se zde atribut about nevyskytuje, znamená to, že se jedná o nově vytvořený zdroj. Takový zdroj je možné považovat za zástupce nějakého fyzického zdroje, který není identifikovatelný pomocí URI.

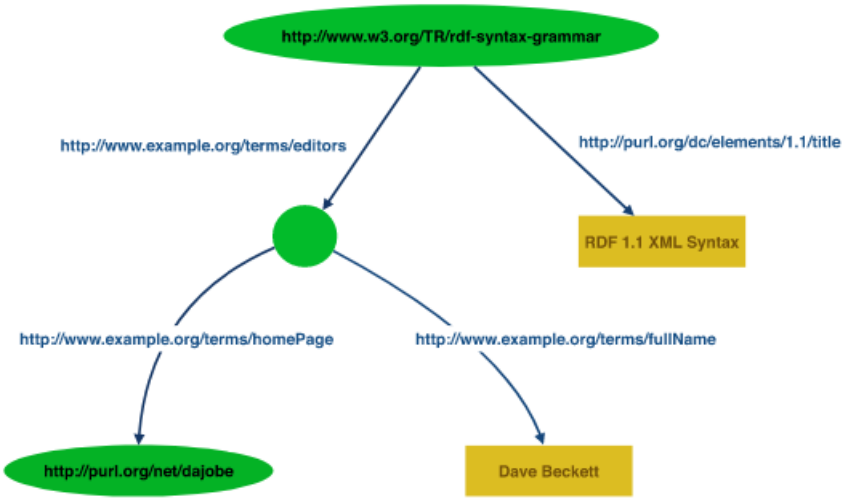
V elementu Descritpion lze zahrnout také atribut ID označující nově vytvořený zdroj. Na zdroj s daným ID je pak možné se odkazovat z jiného elementu Description přes atribut about. Proto nelze atributy ID a about kombinovat a hodnota ID musí být unikátní v rámci celého dokumentu.

Zkrácená syntaxe

Zatímco serializační syntaxe se vyznačuje jasnějším popisem modelu, jsou situace, kdy preferujeme raději kompaktnější zápis. Specifikace definuje tři základní druhy zkrácení serializační syntaxe. Použití zkráceného zápisu je však často omezeno určitými podmínkami, aby nedocházelo k rozdílné interpretaci tvrzení. Pro představu je možné zapisovat vlastnosti zdroje jako atributy elementu Description, ale pouze v případě, že se tyto vlastnosti v rámci elementu neopakují a jejich hodnoty jsou literály. Dalším příkladem je možnost zanořování elementů Description. Zanořování lze aplikovat, je-li objekt tvrzení zároveň dalším zdrojem,

jehož všechny vlastnosti mají jako hodnoty prosté řetězce.

Následující ukázka zachycuje kompletní RDF/XML popis věty v kombinované syntaxi a odpovídající graf. Lze si všimnout, že je v ukázce zahrnuta i definice jmenných prostorů použitých prefixů. Příklad byl převzat ze specifikace pro RDF/XML syntaxi [12].



Obrázek 2.4: Graf pro RDF/XML příklad. Zdroj obrázku [12].

```
<?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>
```

2.2 RDF Schéma

RDF schéma (často zkracováno jako RDFS) je sémantické rozšíření jazyka RDF. Jedná se v podstatě o ontologii - dodává význam jednotlivým URI v sémantickém webu, čímž zajišťuje porozumění mezi počítačovými aplikacemi. Schéma umožňuje definici tříd, binárních relací, hierarchii nad třídami i relacemi. Schémata jsou zapisována pomocí RDF s využitím konstrukcí, které budou představeny v následujícím textu. [6]
Díky RDF schématu máme širší možnosti, než je pouhé určení, jaké má zdroj vlastnosti a jejich případnou strukturalizaci. RDFS používá systém tříd a vlastností, který je podobný typovým systémům používaných v objektově orientovaných programovacích jazycích jako

například Java. Díky tomuto systému můžeme snadno definovat vlastní třídy s určitými vlastnostmi, podle kterých pak zařazujeme zdroje do těchto tříd. Následující popis základních tříd a vlastností čerpá ze specifikace RDF schématu [13].

2.2.1 Základní třídy

Zdroje mohou být rozděleny do skupin nazývaných třídy. Členy třídy pak označujeme jako instance třídy. Třídy samotné jsou však také zdroji. Třídy jsou často identifikovány pomocí IRIs (mezinárodní identifikátor zdroje - řetězec kódovaný prostřednictvním Unicode, který odpovídá syntaxi definované v dokumentu RFC 3987) a mohou být popisovány vlastnostmi RDF.

V RDF rozlišujeme mezi třídou a sadou jejích instancí. Dvě třídy mohou mít stejnou instanci, ale různé třídy. Například daňový úřad může definovat třídu lidí žijících na stejné adrese jako já. Pošta může naopak definovat třídu lidí, jejichž adresa má stejné směrovací číslo jako je moje. Pak by se mohlo stát, že obě tyto třídy budou mít stejné instance, ale různé vlastnosti, protože jedna třída měla vlastnost definovanou poštou a druhá třída vlastnost definovanou daňovým úřadem.

- **rdfs:Resource** - Všechno, co je popisováno RDF výrazy, nazýváme zdroji a tyto zdroje jsou instancemi třídy *rdfs:Resource*. Tzv. třída všeho, všechny ostatní třídy jsou podtřídy této třídy. Je instancí třídy *rdfs:Class*.
- **rdfs:Class** - Základní třída, ze které vycházejí všechny nově vytvořené třídy. Jedná se o třídu zdrojů, které jsou třídami RDF. Platí, že *rdfs:Class* je instancí *rdfs:Class*.
- **rdfs:Literal** - Třída literálních hodnot jako jsou řetězce a čísla. Instance *rdfs:Class* je podtřídou *rdfs:Resource*.
- **rdfs:Datatype** - Třída datových typů. Všechny instance odpovídají RDF modelu datových typů. *rdfs:Datatype* je instancí a zároveň podtřídou *rdfs:Class*. Každá instance Datatype je podtřídou *rdfs:Literal*.
- **rdf:XMLLiteral** - Je instancí *rdfs:Datatype* a podtřídou *rdfs:Literal*.
- **rdf:Property** - Třída pro reprezentaci všech elementů, které jsou vlastností RDF. Je instancí třídy *rdfs:Class*.

2.2.2 Základní vlastnosti

Specifikace popisuje koncept RDF vlastnosti jako vztah mezi subjektem a objektem. Vlastnosti jsou instance třídy *rdfs:Resource*, pomáhají vyjádřit vazby mezi instancemi a jejich třídami, příp. nadtřídami.

- **rdfs:range** - Tato třída je instancí *rdf:Property*, která se používá pro vyjádření, že hodnoty vlastnosti jsou instance jedné nebo více tříd. Jednoduše řečeno se jedná o vymezení hodnot, které daná vlastnost může nabývat, tzn. obor hodnot. Jakýkoliv zdroj s touto vlastností je instancí *rdf:Property*.
- **rdfs:domain** - Vyjadřuje definiční obor vlastnosti. Doména nám určuje, kde může být vlastnost aplikována. Jako příklad můžeme uvést vlastnost *maSyna*. Pak z trojice

„maSyna rdfs:domain Osoba“ víme, že instance třídy *Osoba* mohou mít vlastnost *maSyna*.

Pro upřesnění oboru hodnot bychom patrně psali trojici *„maSyna rdfs:range Muž“* - zde je nutné použít třídu *Muž* nikoliv *Osoba* (žena nemůže být něčím synem).

- **rdfs:subPropertyOf** - Tato vlastnost může vyjadřovat, že jedna vlastnost je podvlastností jiné. Pokud je vlastnost P podvlastností vlastnosti P', pak všechny zdroje související s vlastností P souvisí rovněž s P'.
- **rdfs:subClassOf** - Tranzitivní vlastnost používaná pro vytváření hierarchie mezi vlasnostmi a třídami. Zachycuje skutečnost, že všechny instance jedné třídy jsou zároveň instancemí třídy jiné.
- **rdfs:type** - Určuje, že zdroj jakožto subjekt, je instancí nějaké třídy.
- **rdfs:label** - Instance *rdf:Property* uchovávající jméno zdroje v člověku srozumitelné podobě. Typicky se jedná o řetězec znaků s podporou vícejazyčnosti.
- **rdfs:comment** - Podrobnější popis zdroje (subjektu) napomáhající ujasnit význám tříd a vlastností. Rovněž instance *rdf:Property*.

Poznámka: Základní využití vlastností *rdfs:domain* a *rdfs:range* neposkytuje přímý způsob pro vyjádření omezení vlastností v rámci třídy.

2.3 Jazyk OWL - Web Ontology Language

Web Ontology Language (OWL) tvoří rodinu jazyků pro prezentaci znalostí ve formě ontologií. Jedná se o značkovací jazyk rozšiřující slovníky RDF a RDF schéma o další vyjadřovací schopnosti a elementy vztahující se k třídám a jejich vlastnostem. Následující text, včetně podsekcí, vychází ze specifikace jazyka OWL [10].

Důvodem zavedení OWL byla potřeba mít jazyk, který má větší vyjadřovací schopnosti než je základní sémantika RDFS, aby stroje mohly provádět užitečné úkoly nad dokumenty. RDFS nám dovoluje definovat třídy s mnoha podtřídami, případně nadtřídami (tzv. super třídy) a také definovat vlastností s daným definičním oborem a oborem hodnot. V tomto smyslu lze RDFS považovat za jednoduchý ontologický jazyk, avšak má jisté nedostatky. Pomocí RDFS nemůže například specifkovat, že třída Auto a Osoba nemají žádný společný prvek, nebo že smyčcový kvartet má přesně čtyři členy.

S příchodem OWL tak máme k dispozici ontologický jazyk, který může formálně popsat význam terminologie používané ve webových dokumentech. Vytvářením tříd, definováním vztahů mezi třídami a popisem vlastností prvků těchto tříd modelujeme v jazyce OWL určitý systém znalostí o dané oblasti zájmu.

2.3.1 Ontologie

Než se posuneme dále, bylo by vhodné mít jasnější představu, co je ontologie. Ontologie definuje termíny používané pro popis a reprezentaci určité oblasti znalostí. Jsou využívány lidmi, databázemi a aplikacemi za účelem sdílení nějaké domény informací (doménou zde může být např. medicína, finance, nemovitosti, výroba nábytku, apod.).

Slovo ontologie se používá k popisu artefaktů s různým stupněm sturktury. Může se jednat o jednoduché taxonomie, přes schémata metadat až po logické teorie. Pro účely

sémantického webu je však zapotřebí sturktura vyspělejší úrovně a jako taková musí specifikovat popis pro tyto typy konceptů:

- **Třídy** (obecné věci) v mnoha různých doménách
- **Vztahy**, které mohou existovat mezi třídami/věcmi
- **Vlastnosti** (nebo atributy), které tyto třídy mohou mít

Lze zde spatřit podobnost s objektově orientovaným programováním, zejména pokud jde o třídní hierarchii. Rozdílem je však flexibilita ontologií, kde je potřeba počítat s reprezentací informací pocházejících z mnoha heterogenních datových zdrojů.

Ontologie v informatice mohou být použity pro zdokonalení existujících webových aplikací nebo případně umožnit nějaké nové využití webu.

2.3.2 Rodina jazyků OWL

OWL poskytuje tři varianty jazyků: OWL Lite, OWL DL a OWL Full . Jednotlivé varianty se liší silou vyjadřovacích schopností, proto je každá varianta navržena pro určitou skupinu uživatelů. Každý z jazyků lze považovat za rozšíření svého předchůdce. S každým rozšířením však roste složitost jejich formalismu. Proto by vývojáři používající OWL měli zvážit, který z podjazyků nejlépe splňuje jejich potřeby.

- **OWL Lite** - Nejjednodušší varianta ze tří výše uvedených - je vhodná především pro uživatele, kteří potřebují primárně klasifikací hierarchie a jen jednoduché omezení. Jako příklad omezení lze uvést omezení kardinality - jsou zde povoleny pouze hodnoty 0 a 1. Myšlenkou bylo zajistit snazší vývoj nástrojů a umožnit rychlý přechod z jiných systémů. Postupem času se prokázalo, že vývoj nástrojů pro tuto variantu je téměř stejně složitý a nákladný jako pro OWL DL. Důsledkem toho není verze Lite příliš používána.
- **OWL DL** je zaměřeno na uživatele, kteří mají vysoké požadavky na vyjadřovací schopnosti se zachováním výpočetní úplnosti a nutnosti rozhodnutelnosti (všechny výpočty proběhnou v konečném čase). Varianta DL zahrnuje všechny konstrukce jazyka OWL, avšak jejich použití podléhá určitým omezením. Například třída nemůže být instancí jiné třídy, i přesto že třída může být podtřídou mnoha jiných tříd.Jméno DL souvisí s deskriptivní logikou, jako oblasti výzkumu, která položila formální základy jazyka OWL.
- **OWL Full** je určen pro uživatele maximálně zaměřené na vyjadřovací možnosti bez omezení syntaxí jazyka RDF negarantujícího výpočetní úplnost. V této verzi může být s třídou zacházeno současně jako s kolekcí jedinečných prvků nebo jako s právoplatným prvkem samotným (slovo prvek lze zde považovat za instanci, OWL nazývá instance jako “individuals” ve významu jakéhosi rozšíření třídy). OWL Full umožňuje ontologiím rozšiřovat význam předdefinovaných slovníků. Rozhodnutelnost je zde méně předvídatelná, protože v současné době neexistuje kompletní implementace jazyka.

2.3.3 OWL třídy, podtřídy a instance

Primárním účelem ontologie je klasifikace věcí podle významu. V OWL je toho dosaženo prostřednictvím tříd, vlastností, instancí a operacemi nad třídami.

Třída

Klasifikace instancí/individuí do skupin, které sdílejí společné charakteristiky. Třída může být podtřídou jiné třídy či dědit charakteristiky svého rodiče - super třídy, může mít libovolný počet instancí. Všechny třídy jsou podtřídou kořenové třídy *owl:Thing* a prázdné třídy *owl:Nothing*. Tyto dvě třídy jsou používány pro vyjádření faktů týkajících se všech nebo žádné instance. Třída *owl:Nothing* nemůže mít žádné instance.

Třída je doménou vestavěné vlastnosti *owl:equivalentClass*, která spojuje popis dvou OWL tříd a tím vyjadřuje, že oba tyto popisy mají stejné rozšíření a tedy obsahují presně stejnou množinu instancí. Nejedná se však o skutečnou rovnost tříd, ta se vyjadřuje konstrukcí *owl:sameAs*.

Instance

Instance jsou objekty, které jsou v OWL označovány jako individua. Instance může náležet jedné nebo více třídám, případně také žádné třídě.

Vlastnosti

OWL rozlišuje dvě hlavní kategorie vlastností: vlastností objektů *owl:ObjectProperty* a vlastností datových typů *owl:DatatypeProperty*. První výše uvedená vlastnost představuje vazby mezi instancemi dvou tříd. V druhém případě se jedná o vazby mezi instancemi tříd a datovými typy nebo literály.

Operace

Jazyky zahrnují také podporu pro operace nad třídami. Těmito operacemi jsou: průnik, sjednocení, doplněk, případně výčet, kardinalita a prázdný průnik.

2.3.4 Příklad OWL ontologie

Již jsme si vysvětlili význam ontologií a přínos jazyku OWL spolu s jeho základními prvky potřebnými pro klasifikaci. Nyní se podívejme, jak OWL může vypadat v praxi. Na následujícím jednoduchém příkladu demonstrujeme anotaci dat pomocí sémantických metadat s využitím principů RDFS a OWL. Náš příklad v podstatě představuje jednoduchou ontologii definující druhy nábytku.

Hned na začátku si můžeme všimnou definice jmenných prostorů - zajímavé pro nás jsou především RDFS a OWL. Dále následuje hlavička, jejíž definice není povinná, ale představuje dobré místo, kde můžeme uživatelům přiblížit, jaké oblasti se naše ontologie týká.

Máme zde definovány celkem tři třídy: *Nábytek*, *Židle*, *Stoly*. Třídy *Židle*, *Stoly* jsou podtřídami od *Nábytek*. Třída *Židle* má pak dvě konkrétní instance: *ema*, *jitka*. Ukázka zahrnuje také definici objektové a datatypové vlastnosti, zmíněné výše, viz 2.3.3. Jako objektovou vlastnost zde máme *podobnost*, propojující jednu instanci s druhou. V našem případě tuto vlastnost nalezneme u instance *jitka* a vyjadtřuje podobnost s židlí *ema*.

Datatypovou vlastností je zde *material* (z jakého materiálu je židle vyrobena) s hodnotou v podobě litorálu. Zde si pozorný čtenář může všimnout již zmíněné odlišnosti od objektově orientovaného programování. Vlastnost *material* byla definována naprosto nezávisle na jakékoliv z uvedených tříd - byla pouze přiřazena instanci *ema*. Tedy jiná instance stejné třídy vůbec nemusí mít tuto vlastnost. V OWL nejsou vlastnosti instancí popisovány

v třídách a tedy danou vlasnost *material* může mít i instance úplně jiné třídy. [8]

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:produkty="http://www.drevo.cz/produkty#">

  <!-- OWL Ukazka definice hlavicky -->
  <owl:Ontology rdf:about="http://www.drevo.cz/produkty">
    <dc:title>Priklad ontolotgie nabytku</dc:title>
    <dc:description>Demonstrace pouziti RDFS a OWL</dc:description>
  </owl:Ontology>

  <!-- OWL definice tridy -->
  <owl:Class rdf:about="http://www.drevo.cz/produkty#nabytek">
      <rdfs:label>Typ nabytku</rdfs:label>
      <rdfs:comment>Trida vsech druhu nabytku.</rdfs:comment>
  </owl:Class>

  <!-- OWL definice podtridy - Zidle -->
  <owl:Class rdf:about="http://www.drevo.cz/produkty#zidle">
      <rdfs:subClassOf rdf:resource="http://www.drevo.cz/produkty#nabytek"/>
  </owl:Class>

  <!-- OWL definice podtridy - Stoly -->
  <owl:Class rdf:about="http://www.drevo.cz/produkty#stoly">
      <rdfs:subClassOf rdf:resource="http://www.drevo.cz/produkty#nabytek"/>
  </owl:Class>

  <!-- Definice vlastnosti nezavisle na tride -->
  <owl:DatatypeProperty rdf:about="http://www.drevo.cz/produkty#material"/>

  <!-- Definice vlastnosti vyjadrujici podobnost s~jinym produktem -->
  <owl:ObjectProperty rdf:about="http://www.drevo.cz/produkty#podobnost"/>

  <!-- Konkretni instance - zidle Ema -->
  <rdf:Description rdf:about="http://www.drevo.cz/produkty#ema">
      <rdf:type rdf:resource="http://www.drevo.cz/produkty#zidle"/>
      <!-- Pridani vlastnosti material -->
      <produkty:material>Masiv</produkty:material>
  </rdf:Description>

  <!-- Konkretni instance - zidle Jitka -->
  <rdf:Description rdf:about="http://www.drevo.cz/produkty#jitka">
      <rdf:type rdf:resource="http://www.drevo.cz/produkty#zidle"/>
```

```
      <!-- Zachyceni podobnosti s~zidli Matylda -->
      <produkty:podobnost rdf:resource="http://www.drevo.cz/produkty#ema"/>
  </rdf:Description>

</rdf:RDF>
```

Kapitola 3

Extrakce informací z WWW

Internet nám otevírá bránu k masivnímu množství informací. Vzhledem k tomu, že mnoho tištěných publikací, knih, článků, apod. se v nějaké formě vyskytuje i na webu, mohli bychom internet prohlásit za největší zdroj informací. S denně přibývajícím obsahem je však stále větší potřeba mít k dispozici další způsoby, jak co nejpohodlněji pracovat s těmito daty. Proto se zavadějí různé techniky extrakce informací z webu.

Extrakcí informací se rozumí proces, při kterém se snažíme z informačních zdrojů vyjímat informace relevantní z hlediska obsahu dokumentů nebo z hlediska informačního dotazu [2].

Existuje mnoho způsobů, jakými lze dolovat data z webu a potažmo dokumentů. Některé tyto způsoby jsou velmi sofistikované, ale lze narazit i na jednodušší přístupy k tomuto úkolu. Z těch nejtriviálnějších můžeme zmínit použití regulárních výrazu pro filtraci pro nás relevantních informací. Další možností se otevirají s použitím Document Object Modelu (DOM) umožňujícím procházet hierarchickou strukturou dokumentu. Zásádním nedostatkem těchto dvou technik je zaměřenost na specifickou skupinu dokumentů.

Za účelem dosažení obecnějšího řešení problematiky extrakce je nutné použít metodu aplikovatelnou na libovolný typ dokumentů. Příkladem mohou být metody založené na použití tzv. obalů (wrapperů). Generování těchto obalů lze do jisté míry zautomatizovat - často je k tomu nápomocná syntaxe jazyka HTML. Pro definici obalů se používá ontologický popis dat, která mají být z dokumentu extrahována.

Výše zmíněné metody zde nebudou rozebírany. Pro účely této práce zde bude podrobněji popsána metoda extakce informací založená na vizuální reprezentaci vykresleného dokumentu, viz 3.1. V zavěru této kapitoly budou přestaveny dvě práce týkající se problematiky extrakce informací, 3.2.

3.1 Metoda extrakce informací z webu založená na hledání vizuálních vzorů

Metoda se zaměřuje na extrakci informací z dokumentů, jejiž cíle jsou popsány ontologií. Základním principem metody je automatické nalezení vzorů představující datové záznamy ve vizuální reprezentaci zdrojového dokumentu a jejich použití k extrakci.

Vstupní dokument je převáděn na množinu vizuálních oblastí. Využitím DOM se vyberou pouze textové uzly, které jsou vykreslovacím enginem CSSBox[1] transformovány na

[1] http://cssbox.sourceforge.net/

oblasti reprezentující daný text. Vizuální oblast je formálně definována jako trojice:

$$a = (text, rect, style) \tag{3.1}$$

text představuje text obsažený v oblasti, $rect = (x, y, w, h)$ vytyčuje pozici a hranice oblasti na stránce. Styl textu pak definujeme jako:

$$style = (fs, w, st, c, bs) \tag{3.2}$$

jednotilvé prvky stylu mají popořadě význam: průměrná velikost písma, $w \in [0, 1]$ je průměrná hodnota tloušťky písma, kde 1 znamená, že celá oblast byla vyhodnoce jako tučné písmo, 0 normální. $st \in [0, 1]$ pro průměrnou hodnotu stylu písma, zde 1 představuje kurzívu. Hodnoty c a bs jsou barvy popředí a pozadí. Výsledkem transformace CSSBox enginem je množina všech vizuálních oblastí v dokumentu:

$$A = a_1, a_2, \ldots, a_m \tag{3.3}$$

m údává celkový počet vizuálních oblastí, přičemž jedna oblast představuje jeden textový uzel ve vstupním dokumentu.

Jak jsme již zmínili výše, cílová doména informací je popsána ontologií. Podle této ontologie je vytvořena množina značek, tvz. tagů, které jsou poté přidělovány jednotlivým oblastem spolu s hodnotou na intervalu $(0, 1)$ udavající mírů pravděpodobnosti korespondence dané oblasti a přidělené značky.

Nejprve je provedeno počáteční přidělení značek, které je v dalších fázích zpřesňováno. Mohou nastat situace, kdy je oblasti přiděleno více značek. Pak je nutné uvažovat kontext, v němž se oblast nachází - tedy jaké kombinace datových polí očekáváme v extrahovaných záznamech. Aby bylo možné eliminovat dvojznačné oblasti, je potřeba hledat vizuálně konzistentní záznamy. Záznam je vizuálně konzistentní, pokud mají všechna datová pole konzistetní prezentační styl a zároveň rozložení. Vzorec vyskytující se v dokumentu nejčastěji, tedy takový, který pokrývá co nejvíce vizuálních oblastí, se použije pro konečnou identifikaci cílových záznamů ve vstupním dokumentu. Při hledání takového vzorce musíme brát vpotaz také očekávanou sémantiku vazeb, která je vyjádřená kardinalitou vztahu.

Tato sekce zaměřená na popis metody vychází ze zdrojů [7][5].

3.2 Existující nástroje

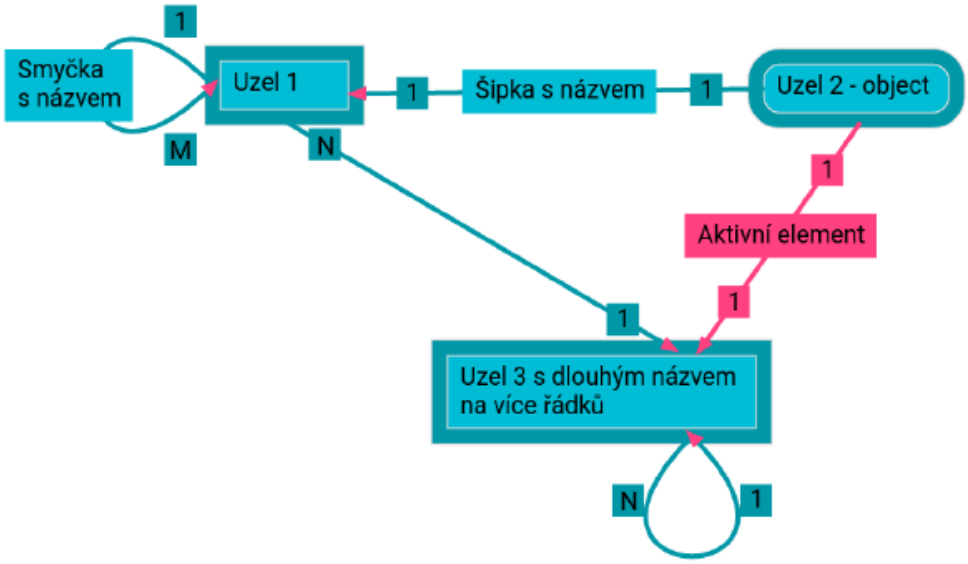
Součásti zadání této práce bylo prostudovat a seznámit se s již existujícími nástroji souvisejícími s extrakcí informací z dokumentů a použití ontologií. Jedná se o bakalářskou práci pana Pokorného[11] a diplomovou práci pana Buby[5].

Následující sekce stručně popíše tyto nástroje společně s jejich případnými přednostmi, které by bylo možné v modifikované podobě využít v mé práci. Stejně tak se pokusím nastínit jisté nedostatky, které bych měl ve své práci vyřešit.

3.2.1 Webové uživatelské rozhraní nástroje pro extrakci informací

Produktem této práce je uživatelské rozhraní pro zadávání extrakčních úloh pro nástroj FITLayout. Aplikace je napsána v JavaScriptu za použití moderních knihoven jako jsou React a Redux (pro správu stavu). Rovněž je zde umožněn přístup více uživatelů a jejich jednoduchá správa.

Jednou z hlavních předností aplikace je možnost vytvářet zadání extrakčních úloh ve formě interaktivních grafů. Uzly grafu představují RDF objekt nebo subjekt, hrany pak znázorňují predikáty. Pro samotné vykreslování grafu je použit vektorově založený grafický formát SVG[2].



Obrázek 3.1: Zadávání extrakční úlohy formou interaktivních grafů. Zdroj obrázku[11].

Zadaní extrakční úlohy je předáváno nástroji FITLayout v JSON formátu specifikujícím seznam všech uzlů a hran, které je propojují. Jednotlivé hrány obsahují název a kardinalitu predikátu. Uzly jsou pojmenovány a obsahují také seznam URI a informaci zda se jedná o objekt nebo subjekt.

Nedostatky

Administrátor nemůže přímo spravovat grafy (extrakční úlohy) ostatních uživatelů. Vidí pouze seznam existujicích uživatelů, a aby měl možnost prohlížet úlohy, musí se nejprve jako daný uživatel přihlásit.

Práci s nástrojem by velice zpříjemnila implementace akcí „zpět“ a „vpřed“ pro snazší manipulaci při tvorbě grafu.

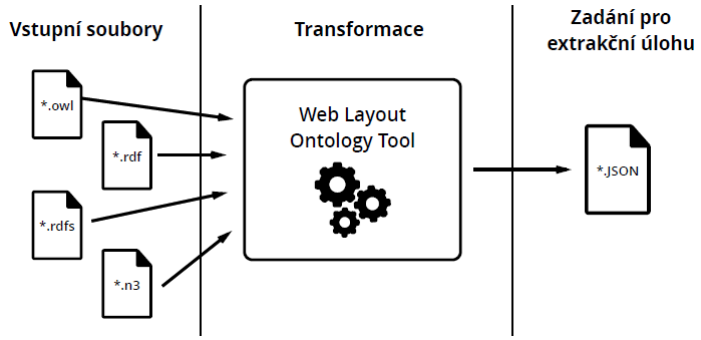
3.2.2 Extrakce informací z webu založená na ontologiích

Tato práce, jejíž autorem je Vojtěch Buba, se zaměřuje na extrakcí informací z webu založenou na konceptuálním modelování pomocí ontologií. Cíle byla implementace nástroje umožňujícího načíst vstupní ontologie zadané ve formátech RDFS nebo OWL. Dále poskytnout uživateli grafické rozhraní pro další práci s načtenými ontologiemi, jako například výběr pouze určitých tříd a vlastností, či editace ontologie.

Nástroj realizuje převod souboru se vstupní ontologií na zadání extrakční úlohy. Toto zadání je ve formátu (JSON), kterému rozumí nástroj FITLayout[3] umožňující analýzu

[2]https://www.w3.org/Graphics/SVG/
[3]http://www.fit.vutbr.cz/ burgetr/FITLayout/

webových stránek. Samotná extrakce informací podle vygenerovaného zadání je pak v režii frameworku FITLayout. Činnost nástroje je zřejmá z nasledujícího schématu:



Obrázek 3.2: Grafické znázornění činnosti popisovaného nástroje. Zdroj obrázku[5].

Nástroj implementuje v jazyce Java výše popsanou metodu extrakce založenou na hledání vizuálních vzorů. Je tedy nutné zpřístupnit uživateli nastavení taggeru, jenž má být použit pro výpočet pravděpodobnosti jednotlivých vlastností a také umožnit zadat dodatečné vlastnosti (např. klíčová vlastnost pro extrahovaný záznam), které není možné vyčíst ze vstupních ontologií.

Nástroj obstojně řeší výpočet kardinality mezi dvojicemi vybraných datatypových vlastností pomoci Dijkstrova algoritmu pro hledání cesty v grafu na základě objektových vlastností načtených tříd.

Kapitola 4

Webová aplikace na platformě Java

Obsahem kapitoly je představení a stručný popis technologií, knihoven a nástrojů, které budou využity během implementace aplikace. Můžeme si všimnout, že jsou zde uvedeny nástroje psané v jazyce Java i JavaScript. Důvodem je použití různých implementačních jazyků pro *backend* a *frontend*.

4.1 Spring Framework

Jeden z velice oblíbených frameworků pro vývoj webových aplikací na platformě Java. Jeho hlavním cílem je usnadnění vývoje těchto aplikací. Správným použitím můžeme získat modulární kód s volnými vazbami mezi moduly, čímž zvyšujeme také čitelnost kódu.

Velikou předností tohoto nástroje je možnost vkládání závislostí *(Dependency Injection)*, díky čemuž se zvyšuje znovupoužitelnost komponent a značně zjednodušuje testování. Toto vkládání závislostí je opravdu jednoduché pomocí systemů anotací. Anotací *@Component* označíme kompnentu určenou pro vložení, které zajistíme pomocí anotace *@Autowired* aplikovatelné podle potřeby na konstruktor, atribut nebo *setter* metodu.

4.1.1 Spring boot

Není žádným tajemstvím, že tvorba *Java Enterprise* aplikace obnáší mnoho konfigurace v počáteční fázi vývoje. Jedná se například o volbu a správné nastavení webového serveru, databázového systému, dalších podpůrných frameworků jako Hibernate, Lombok, Spring Security, apod.

Nástroj Spring Boot si dává za cíl co nejvíce ulehčit tvorbu těchto aplikací, založených na frameworku Spring. Jeho využitím tak dostáváme samostatnou aplikaci schopnou produkčního nasazení s minimální potřebnou konfiguraci.

Samotnou inicializaci projektu můžeme provést skrze webovou stránku *Spring Initializr*[1]. Jednoduše zde nastavíme potřebné parametry jako jazyk *(Java, Kotlin, Groovy)*, nástroj pro automatické sestavení *(Maven, Gradle)*, metadata projektu a hlavně nástroje, které chceme využívat při vývoji. Pro účely vyvíjené aplikace byly zahrnuty tyto nástroje:

- **Web** - jádro pro vývoj aplikace, zahrnuje Tomcat server a Spring MVC

[1] https://start.spring.io/

- **Spring JPA** - zde je jako poskytovatel implementace *Java Persistence API* zahrnuto *Hibernate*. Vyhodou Spring JPA je mimo jiné možnost kdykoliv změnit poskytovatele, což by nebylo možné, kdybychom použili pro ukládání dat samotné *Hibernate*.
- **Spring Security** - zabezpečení aplikace - autentizace a authorizace uživatelů
- **Lombok** - automatická definice metod *getter*, *setter* pomocí jediné anotace *@Data*

4.2 React

React je JavaScriptová knihovna pro tvorbu uživatelských rozhraní. Knihovna je orientovaná na komponenty, které si samy spravují svůj vlastní stav. Kompozicí těchto komponent je pak možné vytvářet složitější a hlavně interaktivní uživatelské rozhraní. Jeho myšlenkou je navhrnout *view* pro každý stav naší aplikace a React za nás efektivně zaktualizuje a vykreslí správnou komponentu podle změn na datech.[2] Za účelem dosažení většího pohodlí při vývoji aplikace bude řešení využívat tyto podpůrné nástroje:

- **Redux** - Knihovna pro správu stavu našich komponent, umožňující držet pro celou aplikaci jediný globální stav a odstínit takto stavy všech komponenty. Obzvláště vhodné pro rozsáhlejší aplikace.
- **Axios** - Poskytuje možnost provádět asynchronní požadavky a odchytávat odpovědi pomocí *callback* funkcí. Pro nás je důležitá zejména podpora JSON formátu.
- **Webpack** - Ve své podstatě se jedná o nástroj poskytující možnost vytvářet JavaScriptové balíčky z jednotlivých modulů kódu. Při své činnosti rekurzivně prochází a sestavuje graf závislosti zahrnující všechny moduly, které aplikace potřebuje a následně je všechny zabalí do jednoho nebo více balíků.[3]

[2] https://reactjs.org
[3] https://webpack.js.org/concepts

Kapitola 5

Návrh řešení

Kapitola zachycuje hlavní myšlenky návhru řešení výsledné aplikace této práce vycházející ze specifikace požadavků. Je zde uveden přehled funkcionality, která bude využita z již exitujících aplikací spolu s jejich nutnou modifikací pro odtranění případných nedostatků popsaných v předchozí kapitole a zajištění kooperace jako celku. Dočteme se zde také o chystaných vylepšeních a krocích potřebných pro jejich úspěšnou implementaci.

5.1 Specifikace požadavků

Výsledným produktem této práce bude webová aplikace na platformě Java fungující jako administrační rozhraní aplikace pro extrakci informací z webových stránek umožňující následující činnosti:

- správa uživatelů
- vytvoření nové extrakční úlohy formou interaktivních grafů
- načtení úložených úloh s možností provádět další úpravy
- načítání souborů s ontologiemi ve formátu OWL, RDFS a Turtle[1]
- možnost výběru požadovaných tříd a vlastností z načtené ontologie
- vykreslení interaktivního grafu představujícího načtenou ontologii nebo vybrané části
- podpora akcí „zpět“ a „vpřed“ při sestavování grafu úlohy
- administrátor může prohlížet příp. editovat úlohy ostatních uživatelů
- zadávání zdrojů dat
- export extrakční úlohy ve formátu vhodném pro nástroj FITLayout

Je patrné, že hlavním úkolem aplikace je možnost vytvářet extrakční úlohy pro nástroj FITLayout. Uživatel bude mít dvě možností přístupu k tomuto úkonu, případně může zkombinovat oba způsoby.

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

První možnost obnáší načíst ontologii ze souboru a následně skrze hierarchii zaškrtávacích políček znázorňující hierarchii tříd a jejich vlastností, vybrat pouze požadované z nich a vykreslit graf extrakční úlohy. Před samotným exportem úlohy bude možné ještě provést dodatečné úpravy prostřednictvím editoru, např. doplnění kardinality vztahů, apod.

Druhou možností bude vytvařet úlohu na základě prostředků poskytnutých editorem pro tvorbu interaktivních grafů. To obnáší postupné přidávání jednotlivých uzlů, určení zda se jedná o subjekt nebo objekt a jejich vzájemné propojení hranami spolu s nadefinováním kardinality vytvořeného vztahu. Nabízí se také možnost vykreslit graf např. část načtené ontologie a k ní přidat další uzly a vztahy.

5.2 Návrh architektury

Z úvodu této kapitoly a ze závěru kapitoly předešlé vyplývá, že chystaná aplikace by měla zahrnovat určitou funkcionalitu z již existujících řešení. Zjednodušeně řečeno se jedná o backend z desktopové aplikace napsané v jazyce Java a frontend z webové aplikace využívající framework React[2]. Musíme tedy navrhnout způsob propojení dvou částí implementovaných v různých jazycích: Java a JavaScript.

Jako vhodný způsob přístupu k tomuto úkolu se jeví aplikace architektury klient-server, přičemž vzájemná komunikace mezi klientem a serverem bude probíhat skrze definované REST rozhraní. Aby byl návrh co nejvíce oddělený, rozhodl jsme se při implementaci realizovat architektonický vzor Model–view–controller[3], dále jen MVC.

5.2.1 Klient

S ohledem na volbu vzoru architektury klient představuje prosté *View*. Jedná se tedy o uživatelské rozhraní s minimem aplikační logiky. Nejnáročnějším úkolem klienta je umožnit uživateli tvořit interaktivní grafy s pamětí „zpět“ a „vpřed“.

Bude kompletně napsán v JavaScriptu s využitím frameworku React. Pro správu stavu využijeme knihovnu Redux a pro navigaci React Route. Knihovna Axios pak poslouží ke komunikaci se serverem přes volání REST API.

5.2.2 Server

Zaštiťuje hlavní logiku aplikace a presistenci dat. Hlavní logikou je zde myšlena především autentizace uživatelů a autorizace jejich akcí. Dále práce s ontologiemi - načtení tříd ontologií spolu s příslušnými vlastnostmi a kardinalitou vztahů, převod do vlastní reprezentace pro další zpracování. Příklad ukázky viz dále.

Je zde zajištěna také definice aplikačního rozhraní pro vzájemnou komunikaci s klientem. Typickou činností serveru se rozumí příjem požadavků klienta, jeho zpracování a odesláni odpovědi.

Ukázka činnosti aplikace

Následující ukázka zachycuje kooperaci klienta a serveru při načtení souboru s ontologií za účelem kategorizace na třídy s vlastnostmi pro jejich následné zobrazení v hierarchii zaškrtávacích políček na straně klienta.

[2] https://reactjs.org/
[3] https://en.wikipedia.org/wiki/Model-view-controller

1. Klient umožní uživateli výběr souboru s ontologií a přečte si obsah souboru.
2. Klient volá službu aplikačního rozhraní serveru *POST /api/load* a odešle JSON objekt: *{"content": "obsah načteného souboru ..."}*
3. Server zpracuje přijatou ontologii s využitím knihovny OWL API, převede získané třídy a vlastnosti do vlastní reprezentace (případně je uloží do databáze) a vrací odpověď klientovi:

```
{
  "classes": [
    {
      "id": 1,
      "name": "Paper",
      "properties" :[
        {
          "id": 2,
          "name": "title"
        }
    ...
}
```

4. Klient zobrazí na základě odpovědi hierarchii zaškrtávacích políček pro výběr požadovaných tříd/vlastností z nichž má být vykreslen graf.

5.3 Správa uživatelů

Jedním ze základních požadavků je víceuživatelský přístup, přičemž k aplikaci budou přistupovat dvě skupiny uživatelů: administrátor a běžný uživatel. Z tohoto důvodu bude nezbytné uchovávat v databázi informace o uživatelích. Jedná se hlavně o informace jako přihlašovací jméno, heslo a role uživatele.

Pro možnost příhlašování prostřednictvním jména a hesla musí být zajištěna unikátnost jména v rámci tablulky uživatelů. Abychom dosáhli alespoň základní úrovně zabezpečení, nebudou se hesla uživatelů nikam posílat ani ukládat v otevřené podobně. V databázi se bude uchovávat pouze otisk hesel a to v podobě SHA-256[4]. Při pokusu o příhlášení se tedy nejprve vygeneruje hash zadaného hesla, který bude porovnávan s otiskem v databázi.

Záznam role uživatele bude pravděpodbně obsahovat řetězec *user* nebo *admin* pro srozumitelné oddělení. Implementace pak definuje třídu výčtového typu *UserRole* s těmito hodnotami.

Musíme zajistit, aby aplikaci mohli využívat pouze přihlášení uživatelé. K tomu využijeme framework Spring Security[5] poskytující jak autentizaci, tak i autorizaci přístupu ke zdrojům. Na straně serveru tak můžeme navíc využít metody třídy *BCryptPasswordEncoder* používající silnou hashovací funkci pro zakódování uživatelského hesla s náhodně zvolenou hodnotou *salt* a také pro ověření správnosti hesla.

Vytvořením potomka třídy *WebSecurityConfigurerAdapter* a překrytím potřebných metod *configure* jsme schopni zajistit, že nepřihlášený uživatel není oprávněn využít funkce

[4] https://en.wikipedia.org/wiki/SHA-2
[5] https://projects.spring.io/spring-security/

aplikace pouze na základě znalostí odkazu dané stránky - v takovém případě bude přesměrován na přihlašovací obrazovku. Stejně tak by mělo být možné filtrovat požadavky běžného uživatele na akce povolené pouze pro administrátora.

Nedílnou součásti aplikace musí být také možnost registrovat nové uživatele. Co se týče zobrazení různých akcí pro obyčejného uživatele a admina lze zajistit přímo na straně klienta s pomocí knihovny React Router umožňující definovat pravidla autorizace podle rolí.

5.4 Načítaní ontologií

Z teoretické části práce jsme se již dočetli o ontologiích a tedy víme, že se skládají zejména z axiomů tříd a jejich vlastností. Vlastnosti mezi třídami jsou vyjádřny skrze objektové vlastosti (atribut *objectProperties* viz diagram níže). Pro načtení a převod ontologie na extrakční úlohu nás však budou zajímat především datatypové vlastnosti (*datatypeProperties*) představující vlastnosti tříd.

Jak bylo předesláno v úvodu kapitoly, bude tato funkcionalita vycházet z práce pana inženýra Vojtěcha Buby[5]. Pro získání jednotlivých prvků ontologie využijeme knihovnu OWL API[6]. Výsledkem aplikace funkcí knihovny obdržíme sadu instancí reprezentující hledané prvky ontologie. Jelikož nám však nestačí jen znát tyto prvky, ale potřebujeme znát také jejich vazby - tedy vztahy mezi třídami a přislušnost vlastností k třídam či kardinalitu - bude pro nás vhodnější vytvořit si vlastní reprezentaci těchto prvku, se kterou se nám bude pohodlněji pracovat.

Obrázek 5.1: Diagram tříd znázorňující vlastní reprezentaci tříd a vlastností z načtené ontologie. Diagram vychází z práce [5].

Třída *OntologyClass* zastupuje třídy z načtené ontologie, *OntologyProperty* pak jejich vlastnosti. Povšimněte si zde také atributu *domain* a *range*, které budeme potřebovat pro vykreslení ontologie jako graf, viz další sekce. Proměnná *iri* slouží jako jednoznačný identifikátor instance nebo-li zdroje tzv. IRI (Internationalized Resource Identifier).

5.5 Vykreslování grafů

Vykreslování grafů bude zcela probíhat na straně klienta. Tato funkcionalita z vetší části převzata z práce pana Pokorného [11]. Je k tomu využit formát SVG umožňující pracovat s vektorovou grafikou přímo v prohlížeči.

Prozatím se spokojíme s vědomím, že graf je na straně klienta uchovávan jako JSON objekt obsahující kromě identifikace a jména grafu především seznam všech uzlů a hran, kterými je tvořen. Samotný způsob vykreslování a manipulace s grafy bude podrobněji ro-

[6] http://owlapi.sourceforge.net

zepsán v následující kapitole Implementace. Zde se zaměříme hlavně na možnosti vykreslení grafu z načtené ontologie.

```
{
  "id": 24,
  " cords ": {
    "x": 216,
    "y": 142
  },
  "width": 100,
  "height": 50,
  "values": {
    "uris": [
      {
        "id": 1,
        "sufix": "name",
        "prefix": ".../foaf/spec/
            index.rdf"
      }
    ],
    "object": true
  }
}
```

Výpis 5.1: Ukázka reprezentace uzlu

```
{
  "id": 33,
  "dst": {
    "id": 24,
    "cords": {"x": 300,"y": 180}
  },
  "src": {
    "id": 25,
    "cords": {"x": 480,"y": 320}
  },
  "width": 23,
  "height": 28,
  "values": {
    "uris": [],
    "title": " Hrana 1",
    "cardinality":
      {"dst": true, "src": false}
  },
  "direction": null
}
```

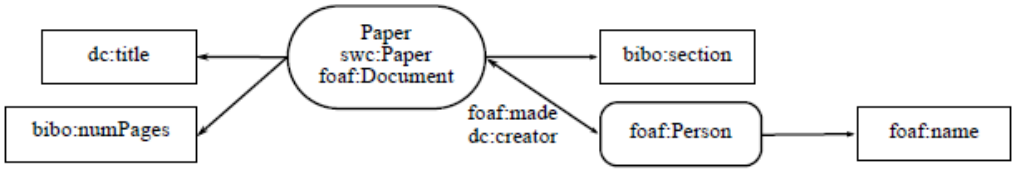
Výpis 5.2: Ukázka reprezentace hrany

V první řadě musí klient odeslat serveru seznam *id* všech vlastností zvolených pro vytvoření nového grafu, případně podgrafu. Spolu s tímto seznamem pošleme také výše zmíněnou reprezentaci již vytvořeného grafu (např. pomocí editoru na klientovi), pokud nějaký je, jinak budou seznamy uzlů a hran v reprezentaci prázdné.

Důvodem proč zasíláme se seznamem identifikátoru také graf je, abychom si mohli vytvořit alespoň základní představu, jaky prostor graf na plátně zabírá a kam přidat nově vykreslovanou část, aby nedocházelo k přílišnému překrývaní. Mimo jiné také chceme odstínit klienta od náročnějších operaci a zajistit, že se opravdu bude chovat jako *View* v naší architektuře. Server tedy zpracuje přijaté vlastnosti ontologie a obohatí seznam uzlů a hran o nové, představující vykreslovaný graf, a odešle toto jako celek zpět klientovi, který pouze vykresli.

5.5.1 Stavba grafu z ontologických vlastností

Pokud by pro vykreslení grafu byly vybrány pouze vlastností příslušející jedné třídě, bylo to snadné - máme totiž již k dispozici vlastní reprezentaci ontologických tříd a vlastností, viz 5.4. Příkladem může být vlastnost *name* třídy *Person* na obrázku níže.
Pokud však uživatel vybere vlastností více různých tříd, tak jako zachycuje obrázek 5.2, je situace obtížnější. V takovém případě si nevystačíme pouze s datatypovými vlastnostmi, ale musíme se zaměřit na objektové, vyjadřující vztah mezi třídami. Na obrázku se jedná o vlastnosti *creator* a *made*. Zde pro změnu využijeme poznatků pro výpočet kardinality mezi dvojicemi vlastností z práce [5].



Obrázek 5.2: Diagram reprezentující ontologi načtenou ze souboru.

Myšlenka je vytvořit orientovaný graf, který budeme procházet Dijsktrovým algoritmem pro hledání cesty v orientovaném grafu. Výsledkem je cesta od první vlastnosti k druhé ve dvojici. Tato cesta je pak využitá pro výpočet kardinality vztahu. My však tento graf využijeme také pro možnost vykreslení (strukturu grafu převedeme do naší reprezentace ve formátu JSON). Postup tvorby orientovaného grafu je následující:

1. Pro každou vybranou ontologickou třídu vytvoříme uzel s identifikátorem rovným URI třídy.
2. Pro každou vybranou datatypovou vlastnost třídy je vytvtořen uzel s identifikací.
3. Třída je se svými vlastnostmi spojena dvěma hranama - jedna ve směru od třídy k hraně a druhá opačně - algoritmus totiž pracuje nad orientovaným grafem.
4. Všechny uzly jsou uzloženy do struktury *HashMap<URI uzlu, uzel>*.
5. Následuje spojování tříd hranami (opět v obou směrech) skrze objektové vlastnosti. Toto je prováděno podle atributu *range* vlastnosti. Pokud je třída v atributu *range* nalezena, vytvoříme hranu v obou směrech. V opačném případě hranu nelze vytvořit.

5.6 Akce „zpět“ a „vpřed“

Za účelem zvýšení pohodlí pří práci s editorem grafů bude implemnována podpora akcí „zpět“ a „vpřed“. Uživatel tedy bude mít možnost zvrátit přibližbě až 30 posledních provedených úprav. Přesná hodnota počtu bude stanovena až při implementaci a testování podle případného dopadu na výkon aplikace.

Podpora této funkcionality bude zajištěna využitím dvou zásobníků, které budou uchovávat stav plátna/grafu před provedením další změny. Na první zásobník budeme ukládat poslední stav vždy při provedení další změny. Při vyvolání akce zpět vložíme aktuální stav na druhý zásobník a plátno se obnoví do stavu z vrcholu prvního zásobníku. Druhý zásobník tudiž umožní zvrátit návrat zpět. Akce vpřed bude zásobníky využívat v opačném pořadí.

Kvůli omezení velikosti bude zapotřebí si vytvořit vlastní implementaci zásobníku. Ten bude reprezentován třídou *Stack*. Důležitá je zde pouze úprava metody *push(item)*, která při pokusu vložit hodnotu do plného zásobníku nejprve odstraní prvek na úplnem dně:

```
push(item) {
    if(this.stack.length >= this.size)
        this.stack.shift();
    this.stack.push(item);
}
```

Samotný zásobník je vytvořen jako pole o zadané velikosti. Pole v JavaScriptu má definované metody *push* a *pop*. Metoda *shift* použitá v ukázce výše pak zajišťuje ono odstranění prvku ze dna zásobníku.

Kapitola 6

Implementace

V aktuální kapitole je podrobněji popsána implementace webové aplikace navržené v kapitole předchozí. Popis je zaměřen na stěžejní funkcionalitu. V některých případech je nastíněno více přístupů k implementaci dané části, společně s objasněním z jakého důvodu byl vybrán jeden konkrétní.

6.1 Kreslení grafů

Ve čtvrté kapitole, viz 4.2, jsme si řekli, že framework React pracuje s komponentami závislými na svém stavu. Zároveň jsme se rozhodli spravovat stav komponent globálně prostřednictvím knihovny Redux. Víme, že při změně stavu komponenty, dojde k jejímu překreslení a na tomto principu je postaveno kreslení a manipulace s grafem.

Kapitola Návrh řešení nám prozradila způsob reprezentace grafu. Graf je uchovávan jako součást stavu aplikace, v podobě objektu obsahujícího především seznam uzlů a hran. Ukázku vyjádření uzlu i hrany si můžeme prohlédnout v sekci 5.5. Takto zachycený uzel či hrana obsahuje všechny potřebné údaje pro jejich vykreslení za pomoci grafického formátu SVG. V případě uzlu se jedná o pozici jeho levého horního rohu, výšku a šířku. Pro hranu potřebujeme jen pozici počátečního a koncového bodu. Zjednodušeně řečeno nám k vykreslení grafu stačí pouze dvě geometrická primitiva a sice obdelník a čára. SVG oba tyto tvary nabízí jako elementy *rect* a *line*.

Samotné vykreslení probíhá v komponentě *Canvas* představující plátno. Komponentě je předán seznam uzlů a hran grafu. Nad každým uzlem je volána metoda *renderNode* a stejně tak pro hranu *renderEdge*.

Metoda *renderNode* vytváří komponentu uzlu *Node*. V komponentě *Node* je zajištěno vytvoření obdelníku, vyjadřujícího uzel, na základě přijatých parametrů: pozice, výška a šířka. Obdelník je složen z vnější a vnitřní části a pro jeho vykreslení jsou tedy použity dva SVG elementy *rect*. Obdobným způsobem pracuje metoda *renderEdge*. Jednotlivé komponenty vytvořené těmito dvěma metodama jsou obsahem plátna *Canvas*.

6.2 Načítání ontologií ze souboru

Funkcionalita načtení ontologie ze souboru vychází z diplomové práce pana Buby[5] využívající knihovnu OWL API. Na začátku server příjme od klienta obsah souboru s načítanou ontologií a předá jej konstuktoru třídy *Parser*. Proces převodu do naší reprezentace, po-

psané v předchozí kapitole, viz 5.4, je spuštěn zavoláním metody *parse* nad nově vytvořenou instancí.

Převod nám velmi usnadňuje knihovna OWL API. V prvním kroce voláme nad instancí načtené ontologie metodu *getClassesInSignature*, čímž získáváme seznam všech ontologických tříd. V dalším kroce se zaměříme na seznam datatypových a objektových vlastností, abychom mohli zkompletovat naši navrženou reprezentaci.

Postup získání obou seznamů vlastností je podobný. Rozhraní knihovny však nenabízí přímočarý způsob, a proto je potřeba volat nad ontologií metodu *getAxioms* zvlášť pro všechny požadované typy vlastností. Nyní přichází na řadu soustava cyklů zajišťující správně přiřazení seznamů vlastností třídám ontologie, včetně správně inicializovaných atributů *domain* a *range*. Vnější cyklus pro průchod všemi načtenými třídami obsahuje dvojici zanořených cyklů - jeden pro zpracovaní datatypových vlastností, druhý objektových. Pro přiblížení, první zanořený cyklus prochází všechny datatapové vlastnosti a porovnává jejich doménu s třídou z vnějšího cyklu. Pokud je nalezena shoda, vytvoří se instance třídy *OntologyProperty* z naší reprezentace a nastaví se atributy *domain* a *range*. Na závěr se přidá tato instance do seznamu datatypových vlastností zpracovávané třídy typu *OntologyClass*. Obdnobným způsobem pro objektové vlastnosti.

6.2.1 Ekvivalentní třídy

V teoretické části jsme se v podsekci 2.3.3 dočetli o existenci vlastnosti *owl:equivalentClass* spojující popis dvou OWL tříd. Kdybychom tuto vestavěnou vlastnost ignorovali, mohlo by být výsledkem zpracování načtené ontologie více stejnojmenných tříd s různými, případně žádnými, vlastnostmi. To je něco, co bychom si nepřáli, a proto musíme načtené třídy rozdělit do skupin sjednocující tyto ekvivalentní třídy.

Opět využijeme služby knihovny OWL API nabízející získání všech dvojic tříd, které jsou si navzájem ekvivalentní, dříve použitou metodou *getAxioms*. To našemu účelu bohužel nestačí. V práci pana Buby[5] je tento úkol vyřešen následujícím způsobem. Z výsledku produkovaného metodou *getAxioms* vytvoříme seznam tříd, u kterých byla nalezena ekvivalence. Na základě tohoto seznamu je sestrojena matice sousedních uzlů - třída *AdjacencyMatrix*. Matice je tvořena mapou indexovanou URI hodnotou třídy, její položky pak obsahují druhou mapu zachycující sousední uzly/třídy (opět URI pro indexaci) a pravdivostní hodnotu vyjadřující ekvivalenci, inicializovanou na hodnotu *false*. Do takto připravené matice zaneseme vztahy ekvivalnce změnou pravdivostní hodnoty na *true*.

Pomocí matice jsme nyní schopni sestavit graf, ktery budeme prohledávat do hloubky algoritmem *Depth first search*. Výběr právě algoritmu *DFS* je odůvodněn v práci [5] (sekce 5.3.3) společně se způsobem aplikace vyhledávání. Matice sama o sobě představuje hrany grafu - existenci hrany vyjadřuje nastavená pravdivostní hodnota. Sestavený graf prohledáváme pro všechny třídy s indikovanou ekvivalencí. Výsledkem každého průchodu grafem je seznam identifikátorů všech tříd spadajících do jedné skupiny ekvivalence.

Skupinu ekvivalentních tříd reprezentuje třída *ClassEquivalencyGroup*. Jedna z ekvivalentních tříd je vždy vybrána jako takzvaná klíčová třída sjednocující chování celé skupiny. Existence klíčové třídy umožňuje implementaci rozhraní *IOntologyClass*, které implementuje také náš reprezentant ontologické třídy *OntologyClass*, díky čemu můžeme se skupinou ekvivalentních tříd pracovat stejným způsobem jako s bežnou ontologickou třídou.

6.2.2 Zobrazení načtené ontologie

Již jsme si vysvětlili, jak probíhá načtení ontologie společně s převodem do naší reprezentace pro snazší budoucí manipulaci. Nyní potřebujeme zobrazit načtené ontologické třídy a vlastnosti uživateli a to ve formě hierarchie zaškrtávacích políček. Vytvoření struktury ve formátu potřebném pro zobrazení na straně klienta pomoci komponenty *CheckboxTreeView* zajišťuje tovární metoda *generate* třídy *TreeViewNodesFactory*.

Metoda *generate* dostává pouze seznam načtených tříd, které převádí na uzly stromové hierarchie. Uzel je zachycen instancí třídy *TreeNode* obsahující kromě názvu a identifikace ontologické třídy/vlastnosti také atribut *children* typu *TreeNode*, čímž vzniká ona hierarchie. Identifikace je důležitá pro výběr konkretních vlastností a následné automatické vykreslení grafu, viz dále. Server vrací výsledek této metody klientovi jako odpověď na požadavek načtení ontologie ze souboru.

6.3 Automatické vykreslení grafu

Předchozí sekce nám odhalila způsob načtení ontologie ze souboru. Následujícím případem použití bývá nejčastěji výběr požadovaných vlastností z nabídky a automatické vykreslení grafu dle výběru pomocí tlačítka *Generate*. Tímto je na server odeslán seznam identifikátorů všech vybraných vlastností.

V prvním kroce tohoto úkolu musíme rozpoznat třídy z načtené ontologie, do kterých náleží vybrané vlastnosti, podle přijatých identifikátorů. Pro každou takovou třídu vytvoříme instanci *GraphObject* a provedeme průnik jejich datatypových vlastností a vybraných vlastností. Uzly tříd jsou automaticky nastaveny jako RDF objekty. Seznam vytvořených instancí předáme metodě *createGraph* z továrny *GraphFactory*, kde proběhne vygenerování podgrafu, který bude vykreslen na klientovi.

Na počátku metoda *createGraph* postupně vytváří nyní již opravdové uzly grafu pro přijaté RDF objekty zastupující třídy vybraných vlastností. Zároveň se zhotoví také uzly reprezentující vlastnosti třídy a tyto uzly se propojí hranou s uzlem třídy. V případě uzlů vlastností se jedná o RDF subjekty. Hrana je vyjádřená instancí *Edge* pomocí identifikátoru zdrojového a cílového uzlu a identifikátoru samotné hrany. Abstrakci uzlů obstarává třída *Node*, kde kromě jména a identifikace nastavujeme také výšku a šířku uzlu pro vykreslení. Pro jednořádkové názvy uzlu aplikace nastavuje výšku 50 pixelů. Šířka se pak odvozuje podle počtu znaků v názvu jako násobek experimentálně zvolené šířky jednoho znaku. Minimální šírka uzlu je nastavena na 100 pixelů.

V druhé fázi metody *createGraph* dochází ke zpracování objektových vlastností - v našem grafu to obnáší propojení uzlů tříd pojmenovanými hranami. To znamená, že musíme u přijatých RDF objektů postupně projít všechny jejich objektové vlastnosti a zjišťovat, zda jejich atribut *range* obsahuje některý z těchto RDF objektů. V pozitivním případě vytvoříme novou hranu spojující dané dva objekty a nastavíme jí jako jméno název vlastnosti.

Každý vytvořený uzel (i hrana) musí mít unikátní identifikátor, díky kterému je možné uzly rozlišit a propojit je hranami. Instance třídy *IdGenerator* poskytuje požadované identifikátory na zavolaní metody *next*. V okamžiku vzniku instance generátoru se nastaví první dostupný identifikátor roven aktuálnímu času v milisekundách, všechny následující vznikají inkrementací o jedničku.

6.3.1 Rozložení grafu na plátně

Pozorný čtenář si zajisté všiml, že v předchozí sekci nepadla ani zmínka o nastavení pozice uzlů a hran v rámci souřadného systému plátna. Musíme si uvědomit, že generovaný graf je vlastně jen podgrafem a jako takový může být součástí manuálně vytvořené úlohy prostřednictvním editoru v aplikaci. Abychom předešli překrývání manuálně vytvořené části a vytvářeného podgrafu, musíme vyřešit problém rozložení jednotlivých elemetnů. Za tímto účelem jsem se rozhodl použít knihovnu *dagre*[1].

Práce s knihovnou je opravdu jednoduchá. Stačí nastavit jednotlivé uzly a hrany. Pro nastavení uzlu potřebujeme znát pouze jeho identifikátor, výšku a šířku. Pro hranu si vystačíme s identifikátorem zdrojového a cílového uzlu. Všechny tyto potřebné údaje jsme si již nachystali v předchozí sekci. Jako výsledek obdržíme souřadnice středu jednotlivých uzlů a také souřadnice počátečních a koncových bodů hran.

V následujícím textu rozepisuji dva přístupy použití knihovny *dagre*. Dopředu prozradím, že ve výsledné aplikaci byl implementován druhý přístup.

První přístup

První přístup vychází z kapitoly Návrh řešení. Myšlenka spočívala v zasílání společně se seznamem identifikátorů vybraných vlastností také seznam existujících uzlů a hran. Do seznamu existujících uzlů a hran by se přidaly uzly a hrany vygenerovaného podgrafu. Na takto sjednocený graf by se aplikovaly funkce knihovny *dagre*, čímž bychom získali zcela nové rozložení pro výsledný graf.

Toto řešení se během implementace ukázalo jako ne příliš ideální a to hned ze dvou důvodů. Hlavním důvodem bylo, že použitá knihovna v případě složitější úlohy začínala selhávat a některé hrany vedly do prázdna nebo úplně chyběly. Druhým důvodem byla skutečnost, že docházelo ke kompletnímu přeskládání podoby manuálně vytvořené části grafu. Takové chování není zrovna uživatelsky přívětivé, neboť to zásadně ztěžovalo orientaci v modelované úloze.

Selektivní režim

Druhý přístup vychází z implementace selektivního režimu pro označení více uzlů grafu, umožňující zejména hromadnou změnu pozice a další akce. Více se o výběrovém režimu dočteme v následující kapitole, viz 7.5.

V tomto případě se vůbec nezajímáme o existující část grafu. Knihovnu použijeme pouze pro výpočet rozložení uzlů a hran vygenerovaného podgrafu, přičemž podgraf je vždy umístěn do levého horního rohu plátna. Ano, ve většině případů dojde k překrytí existujícího grafu přidaným podgrafem, to nás však nemusí tolik trápit, protože se automaticky aktivuje selektivní režim a označí se všechny uzly podgrafu. Uživatel jedním tahem přesune celý přidaný podgraf do vhodné pozice.

Aby toto mohlo fungovat, musí server vrátit klientovi kromě seznamu uzlů a hran podgrafu, nejlépe také seznam identifkátoru všech přidaných uzlů napovídající, které uzly mají být označeny. Tímto způsobem jsme poměrně pohodlně vyřešili oba problémy týkající se prvního přístupu.

[1] https://github.com/dagrejs/dagre

6.4 Akce „zpět“ a „vpřed“

Jak už zaznělo v návrhu, základ této funkcionality tvoří dva zásobníky reprezentované třídou *Stack* zvlášť vytvořenou pro tento účel. Jeden ze zásobníků je určen pro uchovávání předchozích stavů plátna, pro možnost návratů změn, druhý pak pro zvránení zpětných akcí do původního stavu.

Ve snaze získat větší kontrolu nad oběma zásobníky současně a dosáhnout pohodlnější imlementace, byla vytvořena třída *ChangesCache* zapouzdřující všechny stavy plátna v daném rozsahu stanoveném oběma zásobníky. Třída *ChangesCache* logicky vymezuje práci s oběma zásobníky při vyvolání akce „zpět“ či „vpřed“. Zásadní je zde metoda *pushState* přebírající pouze jeden vstupní parametr představující současný stav plátna. Tato metoda je volána vždy před provedením nějaké změny, která má být monitorována pro zajištění možnosti pozdějšího návratu zpět.

Neméně důležité jsou metody *undoAction* a *redoAction*, které jsou volány, jak název napovída, v případě požadavku na akci zpět/vpřed. Způsob implementace obou metod je v zásadě stejný, jejich rozdíl spočívá v revesibilním použití obou zásobníků. Obě metody mají jako vstupní parametr současný stav rozpracované úlohy a vracejí nový stav plátna.

Vezměme například metodu *undoAction* - na počátku je nutné ověřit, zda je akci zpět možné vykonat, tedy jestli není zásobník prázdný. Poté ze zásobníku *undoStack* sejmeme vrchol, který bude vrácen jako nový stav. Vstupní parametr funkce se uloží do druhého zásobníku *redoStack*. Ještě před tímto uložením proběhne identifikace změněného elementu extrakční úlohy na základě vstupního a obnovovaného stavu a tento element je nastaven v grafu jako aktivní. Díky tomu je používání návratu zpět a vpřed více intuitivní, jelikož uživatel přesně vidí, kterého elementu se změna týkala.

Třída *ChangesCache* je inicializována hned v konstruktoru třídy *Graph* představující hlavní komponentu pro tvorbu extrakčních úloh. Odkaz na instanci naší paměti změn je předáván dceřinným komponentám skrze *props* parametry, čímž je možné i odtud monitorovat změny voláním již změněné metody *pushState*.

Během procesu vytváření extrakční úlohy jsou vratné nejen akce jako je přidání nového uzlu grafu, jeho posun po plátně nebo nová hrana, ale je zajištěna kompletní podpora pro práci s grafem/úlohou, což zahrnuje také změnu kardinality u hran, nastavení prefixů, primárního uzlu a taggeru a také samotného jména elementu. Jméno elementu je vedeno jako textové pole a u takového nechceme zvlášť sledovat změnu každého písmenka názvu, ale jen tehdy, když je změna názvu kompletní a konečná.

Zde využijeme HTML události *onFocus* a *onBlur*, na které můžeme pomoci JavaScriptu reagovat a jasně rozlišit, kdy bylo vstupní pole s názvem elementu pod kontrolou uživatele. V události *onFocus* si poznamenáme aktuální hodnotu vstupního pole, v *onBlur* pak hodnoty porovnáme a pokud došlo ke změně, je stav uložen do paměti změn *ChangesCache*.

6.5 Autentizace a autorizace

Základní princip autentizace uživatelů byl nastíněn již v kapitole Návrh řešení, sekce 5.3. Zde si více přiblížíme možnosti, které nabízí framework Spring. Při pokusu o autentizaci skre přihlašovací formůlář aplikace se metodou *POST* zasílá na server požadavek */api/login* obsahující unikátní uživatelské jméno a otisk hesla.

Na serveru se ověřuje identita uživatele na základě obdržených údajů. V případě úspěšného ověření se vytvoří autentizační žeton/*token* jako instance třídy *UsernamePasswordAuthenticationToken*. Token mimojiné uchovává také informaci o systémové roli přihlášeného

uživatele, což se nám bude hodit pro zajištění autorizace jeho akcí. Samotný token je na závěr uložen do bezpečnostního kontextu serveru a tím je server schopen rozpoznat uživatele dokud nedojde k jeho odhlášení a následnému smazání kontextu. Server musí o úspěchu přihlášení informovat klienta, který si prostřednictvím správce stavu zapamatuje základní údaje klienta a umožní mu vstup do systému.

Chceme-li využívat služeb nástroje Spring Security, musíme vytvořit potomka třídy *WebSecurityConfigurerAdapter* a implementovat metodu *configure*. V ukázce 6.1 můžeme vidět, jak lze jednoduše vymezit autorizaci uvedením vzorů adres zdrojů, viz řádky 3 a 4. Na třetím řádku jsou uvedeny zdroje dostupné všem - tedy i nepřihlášeným uživatelům. Tři tečky zde zastupují požadavek na přihlášení a registraci uživatele a export úlohy. Čtvrtý řádek specifikuje nutnost autentizace. Dále je zde možnost určit vlastní příhlašovací stránku a adresu přesměrování po odhlášení ze systému.

```
http
    .authorizeRequests()
        .antMatchers("/", "/login", "/registration", ...).permitAll()
        .antMatchers("/api/**", "/dashboard").authenticated()
        .and()
    .formLogin()
         .loginPage("/login")
         .permitAll()
         .and()
    .logout()
         .logoutSuccessUrl("/");
```

Výpis 6.1: Ukázka konfigurace zabezpečení pomocí Spring Security.

Tímto možnosti nástroje Spring Security zdaleka nekončí. Pomocí systému anotací lze definovat další podmínky autorizace pro jednotlivá volání našeho koncového bodu. Vezměme například požadavek na smazání uložené úlohy - tento úkon může provést buďto vlastník úlohy nebo administrátor, viz ukázka 6.2.

```
@PreAuthorize("#owner == authentication?.name or hasAuthority('Admin')")
```

Výpis 6.2: Ukázka konfigurace zabezpečení pomocí Spring Security.

6.6 Popisovače uzlů

V kapitole zabývající se extrakci informací jsme si v sekci 3.1 popsali metodu extrakce založenou na hledání vizuálních vzorů. Řekli jsme si, že metoda pracuje se značkami přiřazovanými vizuálním oblastem spolu s hodnotou udávající pravděpodobnost korespondence dané oblasti a přidělené značky.

Náš nástroj proto umožňuje přiřadit jednotlivým uzlům značkovač, tzv. *tagger*. Volba značkovačů nijak zásadně neovlivňuje činnost aplikace, tyto hodnoty jsou zahrnuty do exportované extrakční úlohy a nabývají na významu až při dalším zpracovaní v nástroji FITLayout.

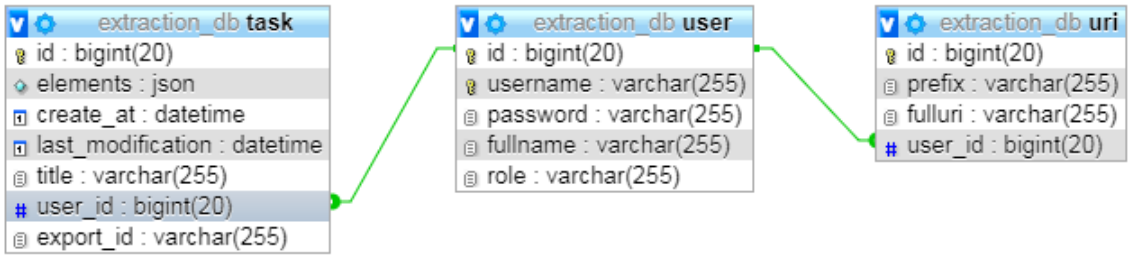
Aplikace uživatelům nedovoluje spravovat dostupné popisovače uzlů přímo skrze rozhraní. Přidávat nové značkovače je možné prostřednictvím konfiguračního souboru *tagger.config.js* - odtud si je aplikace načítá. Seznam značkovačů je uchovávan jako pole jejich

názvu, přičemž pro jejich identifikaci v rámcí aplikace se používá index položky uvnitř tohoto pole.

6.7 Databáze

Server používá pro uchování dat *MySQL* databázi. Výsledná databáze aplikace je poměrně jednoduchá, jak můžeme vidět na obrázku 6.1 zachycujicím databázové schéma. Značného zjednodušení bylo dosaženo právě díky datovému typu JSON nativně podporevanému v *MySQL* od verze 5.7.8[2].

Konkrétně je datový typ JSON použit v tabulce *task* pro položku *elements* reprezentující prvky grafu představujicího exktrakční úlohu. Prvky grafu je myšlen seznam hran a uzlů a identifikátor aktivního elementu grafu.



Obrázek 6.1: Schéma databáze - vygenerováno prostřednictvím nástroje *phpMyAdmin*.

Na straně serveru bylo nutné učinit dodatečná opatření, abychom mohli ukládat do databáze hodnoty ve formátu JSON. V jazyce Java musíme s takovými hodnotami pracovat jako s obyčejným řetězcem, který jsme přijali přes REST rozhraní. Proto byla vytvořena třída *GraphElementsConverter* dedící od *AttributeConverter* za účelem zajištění převodu atributů naší entity do databázové reprezentace a opačně. Převod obstarává implementace metod *convertToDatabaseColumn* a *convertToEntityAttribute*, ve kterých jsme schopni pomocí instance třídy *ObjectMapper* provést potřebná mapování.

Samotné vytvoření třídy *GraphElementsConverter* nestačí, musíme informovat framework Spring o její dostupnosti. Toho dosáhneme pomocí anotací *@Converter(autoApply = true)* v definici třídy a *@Convert(converter = GraphElementsConverter.class)* v místě deklarace atributu vyžadující toto mapování.

[2]https://dev.mysql.com/doc/refman/5.7/en/json.html

Kapitola 7

Grafické uživatelské rozhraní

Kapitola popisuje vytvořené grafické uživatelské rozhraní systému na extrakcí informací z webu. Text se zaměřuje především na hlavní obrazovku aplikace, popis jednotlivých prvků a jejich význam pro systém. Dočteme se zde také o přizpůsobivosti rozhraní aktuální velikosti okna prohlížeče a podpoře provozuschopnosti na různých robrazovacích zařízeních s ohledem na zachování maximálního uživatelského prožitku. Čtenář se seznámí také se způsobem, jakým aplikace informuje uživatele o úspěšně, případně neúspěšně provedené akci.

7.1 Hlavní obrazovka aplikace

Základním účelem vytvořené aplikace je poskytnou uživateli editor pro tvorbu extrakčních úloh ve formě interaktivních grafů. Pokud bychom nebrali v potaz stránku pro přihlášení uživatele a stránku s registračním formulářem, bylo by možné aplikaci označit termínem *Single page application*[1]. Myšlenkou konceptu SPA je poskytnout skrze webový prohlížeč aplikaci, která se svým chováním více podobá běžným desktopovým aplikacím. Na obrázku 7.1 lze vidět hlavní obrazovku aplikace.

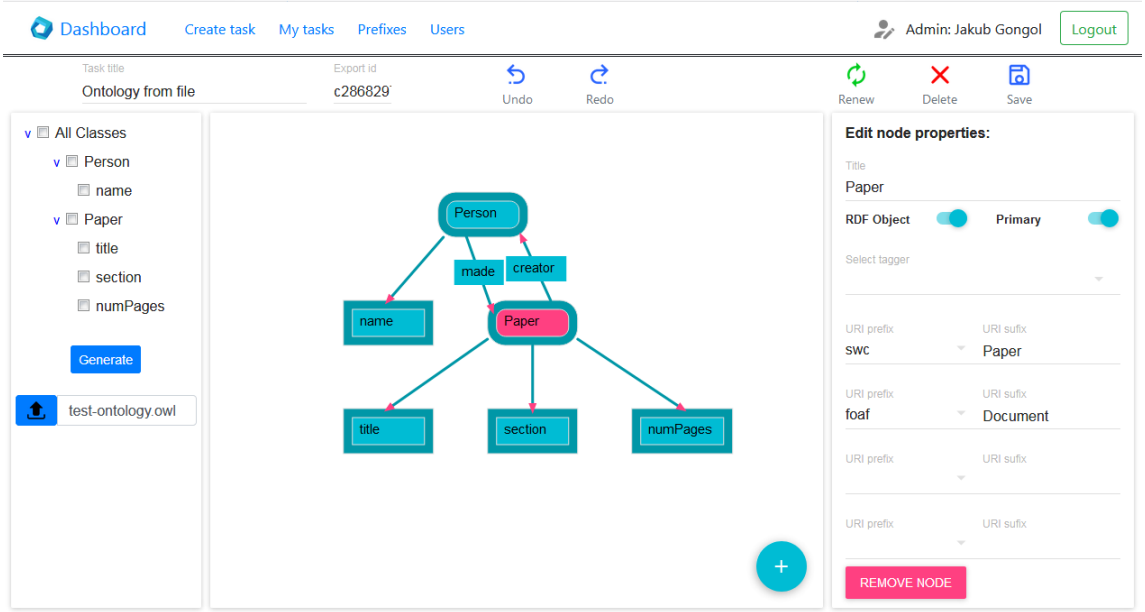
7.1.1 Menu

Hlavní obrazovka se skládá hned z několika sekcí. V horní části je umístěno hlavní menu pro navigaci napříč aplikaci. Menu se skládá z položek: Dashboard, Create task, My tasks, Prefixes a Users. O dashboardu si povíme v následující sekci viz 7.2. Položka Create task představuje možnost vytvoření nové extrakční úlohy, jedná se tedy o okno zobrazeno na popisovaném obrázku 7.1.

My tasks pak zobrazuje seznam všech vytvořených úloh přihlášeného uživatele, viz příloha A. V tomto seznamu jsou úlohy seřazeny podle data poslední modifikace. Kromě data poslední změny zde vidíme jak název úlohy, tak i datum, kdy byla úloha vytvořena, ale hlavně máme prostřednictvním dostuplných akcí přiložitost úlohu načíst a pokračovat v její editaci nebo ji případně úplně smazat.

Další položka menu přináší sekci Prefixes, jejíž podobu si můžeme prohlédnout na obrázku v příloze B. Tato sekce zajišťuje správu uživatelových prefixů určených pro popis uzlů a hran vytvářených grafů. Mimo samotného seznamu dostupných prefixů můžeme v této sekci přidávat nové nebo editovat a mazat stávající.

[1] https://en.wikipedia.org/wiki/Single-page_application



Obrázek 7.1: Hlavní obrazovka aplikace umožňující tvorbu extrakčních úloh formou interaktivních grafů.

Poslední záložka v menu, Users, odkrývá možnost procházet seznam uživatelů registrovaných do systému, viz příloha C. Sekce je však přístupná pouze uživatelům vystupujícím v roli Admin, ostatním uživatelům se záložka vůbec nezobrazí. Opět jsou dostupné určité akce, jež můžeme vykonávat nad jednotlivými uživateli v seznamu. Jedná se zejména o možnost změnit uživatelskou roli, smazat uživatele, ale hlavně zobrazit si všechny úlohy libovolného uživatele. Administrátor tedy může nakládat s úlohami jiných uživatelů stejným způsobem, jako by byly jeho vlastní.

V pravé horní části obrázku si pak můžeme všimnout ikony pro editaci uživatele, zobrazující formulář pro změnu hesla. Dále je zde zobrazeno jméno aktuálně přihlášeného uživatele společně s jeho rolí (běžný uživatel nebo administrátor) a nakonec tlačítko pro odhlášení ze systému.

7.1.2 Panel akcí

Další sekcí hlavní obrazovky je panel akcí, umístěný v horní části okna přímo pod menu. Jedná se v podstatě o ovládací panel pro tvorbu grafů. Z levé strany pořadě zde nalezneme název úlohy a identifikátor pro možnost exportovat úlohu. Samotný export a proces odvození tohoto identifikátoru je popsán v kapitole Integrace a testování, viz 8.1.

Důležité je uporzornit čtenáře, že textové pole *Export id* je při vytvoření nové úlohy prázdné a identifikátor je přidělen až při prvním uložení úlohy. Zároveň stojí za zmínku, že pole nelze modifikovat a při kliku na toto pole dojde k uložení hodnoty identifikátoru do nástěnky pro snazší následné využití.

Zbylou část panelu akcí tvoří pětice tlačítek prezentující se efektem zvětšení při najetí myší. První dvě tlačítka zastávají funkci „zpět“ a „vpřed“ (podrobněji popsáno viz 6.4) a dovolují takto uživateli ovlivnit veškeré změny grafu na plátně, stejně jako změny vlastností uzlů a hran v panelu umístěném na pravé straně obrázku 7.1.

Poslední tři tlačítka slouží pořadě pro vytvoření nové úlohy, přičemž neuložené změny jsou zahozeny, úplné smazání současně zobrazené úlohy (bez zobrazení potvrzovacího dialogu) a nakonec uložení provedených změn. Jak už zaznělo výše, při prvotním uložení úlohy dojde k přidělení identifikátoru pro export.

7.1.3 Panel ontologických tříd a vlastností

Levý postranní panel slouží pro načítání ontologí ze souboru, následný výběr požadovaných vlastností a automatické vygenerování grafu z výběru. Ve spodní části panelu je tlačítko otevírající okno průzkumníka pro výběr souboru s ontologii z uživatelova počítače. Součástí tlačítka je i textové pole, ve kterém je pak zobrazen název vybraného souboru.

Poté, co je soubor vybrán, je jeho obsah odeslán na server, kde proběhně potřebné zpracování popsané v sekci 6.2. Výsledek zpracování na serveru se zobrazí v panelu jako hierarchie zaškrtávacích políček reprezentující ontologické třídy a jejich vlastnosti. Spolu s hierarchií se zobrazí také tlačítko *Generate* zajišťující vykreslení grafu na základě vybraných vlastností v hierarchii. Graf je vykreslen ve středním panelu - Plátno, viz dále.

7.1.4 Plátno

Střední panel je určen ke konstrukci extrakčních úloh ve formě grafu. Kromě automatického vykreslování grafu z načtené ontologie popsaného v předchozí podsekci, můžeme modelovat ulohy také manuálně. V pravém dolním rohu plátna je umístěno tlačítko pro přidání nového uzlu.

Nové uzly jsou přidávány vždy do středu plátna s mírnou náhodnou odchylkou, jejimž cílem je eliminovat kompletní překrytí uzlů v případě, že jich bylo přidáno více najednou. Pro manipulaci s uzlem v rámci plátna se používá myš způsobem *drag and drop*. Uzel se skládá z vnější tmavé části a z vnitřní světlejší části, viz obrázek 7.2.

Obrázek 7.2: Uzel je složen z vnější a vnitřní části ovlivňující způsob ovládání.

Rozdělení uzlu ovlivňuje způsob tvorby grafu. Pro změnu pozice uzlu je potřeba uzel táhnout za vnitřní část. Vnější část je určena pro přichytávání hran. Pokud na vnejší část pouze klikneme myší, vytvoří se u uzlu smyčka vyjadřující unární vztah, viz obrázek 7.2. Bežná hrana se vytvoří tažením z vnější části zdrojového uzlu do cílového.

7.1.5 Panel vlastností uzlů a hran

Poslední panel hlavní obrazovky umístěný v pravé části okna může mít tři různé podoby. Zobrazená podoba závisí na tom, zda je na plátně vybrán jako aktivní element uzel nebo hrana. Třetí podoba se pak zobrazí v případě aktivace selektivního režimu pro označení více elementu a jejich hromadnou manipulaci jako je změna pozice či smazání.

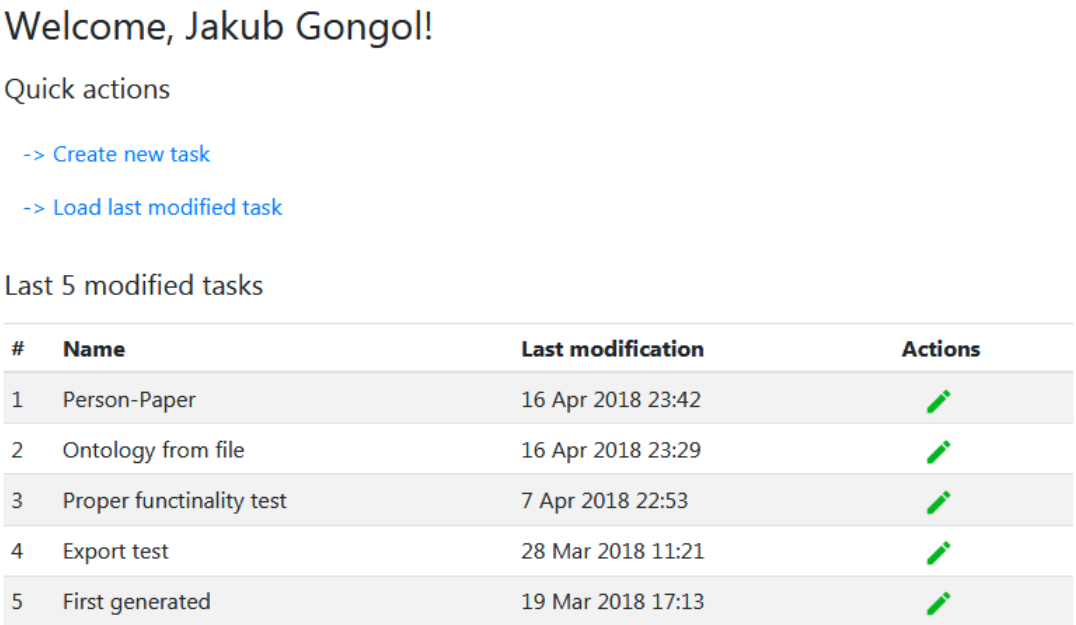
První dvě zmíněné verze panelu jsou v zásadě velice podobné. Jak pro uzel i hranu je možné přidělit název elementu, popsat element až čtyřmi prefixy z naší databáze a

samozřejmě možnost odstranit element. Rozdíly spočívají v možnosti označit uzel jako *RDF Objekt* (vizuálně pak uzel má zaoblené rohy) a přiradit mu takzvaný *tagger*. Uzel navíc může být vybrán jako primární - v rámci celé úlohy je maximálně jeden uzel primární. Naproti tomu u hrany vybíráme kardinalitu vztahu, který vyjadřuje. Implicitní kardinalitu hrany uvažujeme 1 a pro zachování větší přehlednosti na plátně tuto hodnotu nevykreslujeme jako tomu je u kardinality násobné.

7.2 Dashboard

Hned po přihlášení, případně po registraci, systém přesměruje uživatele do aplikace konkrétně na domovskou stránku. Vzhled domovské stránky bez hlavního menu zachycuje obrázek 7.3.

Domovská stránka vítá přihlášeného uživatele a snaží se mu nabídnout nejočekávanější funkce systému společně se základním přehledem jeho poslední činnosti. Obsahuje sekci rychlých akcí tvořenou možnosti vytvořit novou úlohu nebo načíst naposledy upravovanou. Druhá sekce sestává z tabulky obsajující pět naposledy editovaných úloh seřazených dle data poslední modifikace. Z této tabulky máme příležitost jednotlivé úlohy otevřít a pokračovat v jejich úpravách.



Obrázek 7.3: Ukázka domovské stránky aplikace.

7.3 Responzivní chování

Při návrhu a následné implementaci grafického uživatelského rozhraní byl kladen důraz na smysluplné rozložení jednotlivých prvků systému v rámci okna prohlížeče. Jedná se zejména o hlavní obrazovku aplikace pro tvorbu interaktivních grafů. Jelikož vytvářené extrakční úlohy mohou být často složité a tím i odpovídající grafy rozsáhlé, bylo zapotřebí vytvořit

takové rozložení, které se vždy přizpůsobí velikosti okna aplikace, tak aby oblast editoru grafů byla vždy maximální možná.

Bylo tedy nutné se zcela vyhnout volbě optimálních rozměrů plátna editoru, které by pak byly v aplikaci napevno nastaveny, jako tomu bylo v původní verzi aplikace pro kreslení grafů. Jediné napevno nastavené rozměry představují minimální výška a šířka plátna, pod jejichž hranici nelze plátno zmenšit. Tyto minimální rozměry byly odvozeny tak, aby bylo možné aplikaci plnohodnotně používát na obrazovkách s rozlišením od 1366x768 včetně.

7.3.1 Přechod z větší obrazovky na menší

Při aktivním používání aplikace může snadno nastat případ, kdy si vytvoříme extrakční úlohu například na 24 palcovém *Full HD* monitoru a později se k této úloze vrátíme, řekněme na notebooku s pouhým *HD* rozlišením. Zde nastává problém, protože plátno editoru je nyní o přibližně dvě třetiny menší a nebylo by možné kompletně zobrazit náš dříve vytvořený graf.

S cílem vyřešit tento problém, musíme zakročit během procesu nasazení komponenty *Graph* zastřešující obrazovku editoru grafů. Pro tento úkol využijeme metody životního cyklu komponenty frameworku React a také jejího stavu *state*. Těmito použitými metodami jsou:

- *componentWillMount* - komponenta byla úspěšně vytvořena, metoda je volána právě v okamžiku před vložením do modelu *DOM*, ještě před samotným vykreslením
- *componentDidMount* - voláno okamžitě po nasazení komponenty, zde již tedy můžeme bez omezení přistupovat k jednotlivým elementům *DOMu*
- *componentWillUnmount* - zavolá se v okamžiku, kdy je komponenta odstraně z *DOM* a následně zrušená
- *componentDidUpdate* - slouží jako reakce na výskyt aktualizace po prvotním vykreslení, formou přijetí nových *props* parametrů; porovnáním těchto parametrů s předchozími se lze rozhodnout, co se má stát nebo naopak nedělat nic

Nyní, když jsme si přiblížili životní cyklus komponenty a popsali základní význam výše uvedených metod, můžeme je s výhodou použít. Začněme metodou *componentDidMount*, zde kromě samotného načtení grafu můžeme zavolat funkci *updateCanvasDimensions*, která nejprve vyhledá podle přiděleného identifikátoru *canvasArea* element reprezentujicí plátno z *DOMu*. Tímto známe přesné rozměry plátna vzhledem k aktuální velikosti okna. Následně potřebujeme zjistit, jakou plochu zabírá graf na plátně, přesněji jeho krajní body vůči počátku plátna v levém horním rohu. Toho je dosáhnuto průchodem seznamu uzlů a hledáním maximální hodnoty součtu x-ové souřadnice a šířky uzlu, resp. y-ové souřadnice a výšky uzlu.

Se znalostí reálné velikosti plátna a rozměrů načítaného grafu lze pouhým porovnáním rozlišit, kdy je potřeba zobrazit u plátna posuvník, který nám umožní přístup k části grafu ležící mimo oblast plátna. Tedy pokud je šířka plátna menší než je šíře grafu, nastavíme elementu plátna vlastnost *overflow* na hodnotu *auto*, v opačném případě je hodnota rovna *hidden* a plocha grafu je zarovnána s plochou plátna bez rozbrazených posuvníku. Obdobný postup platí i pro výšku s tím, že v závěru metody *updateCanvasDimensions* se ještě zavolá *setState* nastavující nový stav komponenty. Důsledkem nastavení nového stavu se komponenta překreslí se správně nastavenými parametry reflektujicími řešení našeho problému.

Využití zbylých, výše uvedených, metod životního cyklu bude objasněno v následujicím odstavci.

7.3.2 Kontrola hranic plátna

Vzhledem k možnosti proměnlivé velikosti plátna dle aktuální velikosti okna prohlížeče, je nutné správně ošetřit i kontrolu hranic plátna tak, aby nám uzly grafu nemizely mimo přístupnou oblast či naopak, aby šlo využít celou plochu plátna.

Za tímto účelem musíme objektu okna přidat takzvaný *event listener* reagující na událost změny rozměrů okna:

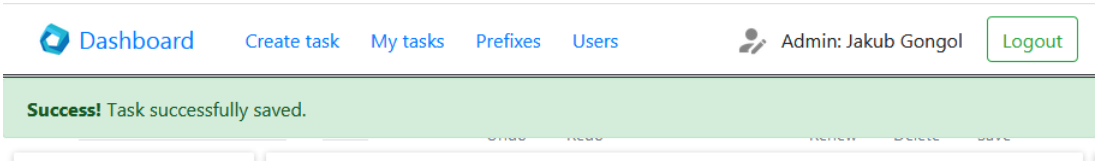
window.addEventListener("resize", this.updateCanvasDimensions);

Nejlepším místem, kde volat tento kód je výše zmíněná metoda *componentWillMount*. Naproti tomu v metodě *componentWillUnmount* tohoto posluchače události odstraníme, jelikož nechceme vykonávat zbytečnou práci pokud komponenta *Graph* není vůbec připojena:

window.removeEventListener("resize", this.updateCanvasDimensions);

7.4 Notifikační panel

Součást aplikace představuje informační panel upozorňující uživatele o úspěchu nebo případném neúspěchu prováděné akce. Umístění panelu pod hlavním menu můžeme vidět na obrázku 7.4 zachycujícím úspěšnou akci.



Obrázek 7.4: Ukázka řešení informačního panelu. Zobrazuje úspěšnou akci.

Na obrázku 7.5 je zobrazena neúspěšná akce spolu s popisem, kde nastala chyba a nezbytnými kroky pro obnovení řádného chodu aplikace. V místě informačního panelu se typicky zobrazuje také indikátor načítání informující o průběhu déle trvající operace, jako například načítaní ontologie ze souboru. Po dokončení takové operace je ukazatel průběhu nahrazen zprávou o úspěchu/neúspěchu.

Obrázek 7.5: Oznámení o neúspěšné akci.

7.5 Selektivní režim

Aplikace disponuje selektivním režimem umožňujícím provádět hromadné akce ovlivňující více elementů současně. Pod hromadnou akcí si můžeme představit změnu pozice všech označených uzlů jedním tahem nebo jejich smazání. Potřeba tohoto režimu je odůvodněná v kapitole Implementace, viz 6.3.1.

Selektivní režim aktivujeme stiskem klávesy s/S a pomocí myší přidáváme požadované uzly do výběru. Prvním kliknutím myší je uzel přidán, opětovaným klikem odebrán z výběru. Označené uzly jsou od ostatních odlišeny barvou použitou pro aktivní element grafu. Spuštěný režim výběru je indikován ikonou umístěnou nad tlačítkem pro přidání nového uzlu, současně pravý postranní panel vlastností uzlů a hran (7.1.5) zobrazuje počet vybraných uzlů s možností hromadného smazání.

Druhý stisk klávesy s/S režim výběru deaktivuje, vybrané uzly však zůstávají označeny a nyní můžeme změnit pozici výběru tažením jednoho z vybraných uzlů. Celý režim je možné ukončit stiskem klávesy c/C a nastolit tak běžné chování editoru.

Kapitola 8

Integrace a testování

První sekce následujícího textu popisuje způsob integrace vytvořeného řešení s existujícím nástrojem FITLayout, vyvíjeným na VUT FIT. Druhá část kapitoly se zaměřuje na testování klíčových funkcí systému.

8.1 Export úlohy

Za účelem jednoduché kooperace s nástrojem FITLayout nevyžaduje export extrakční úlohy autentizaci ani autorizaci. Kdokoliv, kdo zná identifikátor úlohy pro export, může volat službu aplikačního rozhraní */api/task/export/id* a získat tak zadání extrakční úlohy pro FITLayout.

Identifikátor pro export je vytvořen až při prvním uložení úlohy a skládá se z uživatelského jména vlastníka úlohy a časového razítka vytvoření/prvního uložení úlohy v milisekundách. Na takto vytvořený řetězec je aplikována hašovací funkce *SHA-1*[1] produkující identifikátory v hexadecimálním tvaru dlouhé čtyřicet znaků. Tímto je zajištěno, že export id není možné jednoduše uhodnout a bez jeho znalosti export neproběhne.

Formát exportované extrakční úlohy si můžeme prohlédnout v příloze D. Přiložená úloha je tvořena jednoduchým grafem o dvou uzlech vzájemně propojených hranou.

Z ukázky v příloze D je patrné, že formát exportu se liší od reprezentace úlohy nezbytné pro správné vykreslování a manipulaci uvnitř klientské části aplikace. Ku příkladu zde chybí souřadnice uzlů a hran, jejich velikost na plátně a další. Proto byl kromě klasického modelu pro reprezentaci grafů vytvořen speciální model pro export formou jednoduchých Java objektů tzv. *POJO* (Plain Old Java Object). Model je umístěn v samostatném balíčku *export* a názvy jeho tříd začínají předponou *Exported*.

Pro vytváření objektů reprezentujících exportovanou úlohu slouží veřejná metoda *generateExportedTask* z tovární třídy *ExportedGraphFactory*. Tato metoda přebírá jako vstupní parametr instanci třídy *Task* získanou z databáze podle klientem zaslaného identifikátoru exportu. Továrná nabízí pomocné metody *processNodes* a *processEdges* pracující nad seznamy uzlů/hran, nastavující těmto elementům vlastnosti důležité pro export.

8.2 Testování načítání ontologií ze souboru

Hlavním vstupem testu byl soubor zachycující ontologii s třemi různými třídami a několika ekvivalentními verzemi. Třídy mají přiřazeny různý počet datatypových vlastností, jedna

[1] https://en.wikipedia.org/wiki/SHA-1

z nich dokonce nulový. Nechybí také objektové vlastnosti propojující dvojici tříd v obou směrech. Grafické znázornění této ontologie si můžeme prohlédnout na obrázku 5.2 v kapitole Návrh řešení. Obrázek však zachycuje pouze zástupce pro jednotlivé skupiny ekvivalentních tříd a to jen v případě, že mají definované nějaké datatypové vlastnosti. Tabulka 8.1 udává přehled testovacích ontologií a jejich základní charakteristiku. Soubory uvedené v tabulce jsou součásti přiloženého CD disku.

Soubor	Počet tříd	Počet ekvivalentních skupin
burget-papers.owl	6	2
foaf.rdf	19	18

Tabulka 8.1: Tabulka testovacích ontologií se základní charakteristikou.

Test se zaměřuje na řádnou činnost třídy *Parser*, tedy na načtení ontologických tříd společně s jejich datatypovými a objektovými vlastnostmi, ale hlavně na sjednocení ekvivalentních tříd do skupin.

Na počátku proběhla důkladná analýza testovacích vstupů, na jejímž základě bylo postaveno vyhodnocování výsledků ověřením počtů vytvořených ekvivalentních skupin, jejich zástupců a příslušných vlastností.

8.3 Testování automatického vykreslení grafu

Cílem tohoto testování bylo ověřit korektní vytvoření sturktury zachycující generovaný podgraf podle požadovaného výběru z načtench ontologických vlastností.

Hlavní důraz byl kladen především na náležité propojení uzlů, představující RDF objekty, pomocí objektových vlastností. Zde bylo nutné testovat správné nalezení třídy specifikované atributem *range* dané objektové vlastnosti, ve sjednocené množině vzájemně ekvivalentních tříd.

Test byl vyhodnocován pro různou sadu vybraných ontologických tříd a vlasností, ve formě porovnávání grafického výstupu, reprezentující automaticky vygenerovaný podgraf, s očekávaným výsledkem.

8.4 Vyhodnocení testů

Z hlediska načítání ontologií ze souboru neobjevily testy žádný problém. Co se týče automatického vykreslování grafu, i zde byly dosaženy uspokojivé výsledky. Ukázalo se však, že při snaze vygenerovat graf z velmi složité ontologie (např. při výběru všech tříd a vlastností z ontologie *foaf.rdf*), se toto řešení stává poměrně nepřehledné přes velké množství uzlů a hran (a popisků hran). Při opravdu složité struktuře grafu se může stát, že hrana není přímo ukotvena u cílového uzlu. V tomto případě je na vině knihovna *dagre* použitá pro výpočet rozložení grafu na plátně. Je však možné provést jednoduchou korekci pomoci editoru. Celkově tak hodnotím výsledky pozitivně.

Kapitola 9

Závěr

Seznámil jsem se se základními principy sémantického webu a problematikou extrakce informací z webových stránek. Zejména s ontologickými jazyky OWL, RDF schéma a metodami použitelnými pro samotnou extrakci. Rozšířil jsem si přehled v současně využívaných technologiích v oblasti webových aplikací na platformě Java.

Prostudoval jsem dvě práce[5][11] týkající se extrakce informací a analyzoval jsem přednosti a slabiny jejich výsledných nástrojů. Vyvíjená webová aplikace administračního rozhraní systému pro extrakci informací měla za úkol spojit prvky pro načítání ontologií z prvního nástroje a způsob kreslení grafů z nástroje druhého. V rámci této práce jsem navrhl klient–server architekturu, která je schopná propojit nalezené přednosti, za použití patřičných úprav, a zároveň odstranit zjištěné nedostatky výše zmíněných nástrojů používajících různé implementační jazyky a knihovny.

Výsledná aplikace splňuje všechny požadavky zadání. Hlavním smyslem aplikace je tvorba zadání extrakčních úloh formou interaktivních grafů. Tuto činnost zpříjemňuje zabudovaná podpora akcí „zpět“ a „vpřed“. Druhý způsob kompozice zadání úlohy představuje načtení ontologie ze souboru, zobrazení nabídky načtených tříd a vlastností a následné automatické vykreslení grafu na základě výběru. Součást editoru pro práci s grafy tvoří selektivní režim umožňující provádění hromadných akcí, jako je souběžná změna polohy více elementů grafu nebo smazání celého výběru. Je zahrnuta kompletní správa uživatelů, jejich autentizace a autorizace provaděných akcí, včetně zabezpečení aplikačního rozhraní proti zneužití. Aplikace poskytuje koncový bod pro propojení s nástrojem FITLayout skrze export zadání úlohy v požadovaném formátu. Důraz byl kladen také na responzivní chování editoru grafů vzhledem k velikosti okna prohlížeče a jejím změnám, aby editor využíval všechen dostupný prostor. Stejně tak byla zajištěna i přenositelnost a viditelnost úloh při změnách použitého zobrazovacího zařízení.

Z pohledu budoucího vývoje projektu by mohlo být zajímavým námětem přidání podpory převodu zadání extrakční úlohy vytvořené ve formě našeho interaktivního grafu zpět na ontologii popsanou jazykem OWL. Jednalo by se tedy o opačný úkol vzhledem k implementovanému automatickému vykreslování grafů z načtené ontologie. Tímto rozšířením bychom dosáhli snazšího sdílení naších úloh a pohodlnější spolupráce s jinými nástroji zabývajícími se extrakci informací a zpracováváním ontologií.

Literatura

[1] *Semantic Web*. [Online; navštíveno 27.10.2017].
URL https://en.wikipedia.org/wiki/Semantic_Web

[2] Berka, P.: *Inteligentní systémy*. Oeconomica, 2008, ISBN 978-80-245-1436-9.

[3] Berners-Lee, T.: *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web*. Harper, 2000, ISBN 0-06-251587-X.

[4] Boye, J.: *RDF - What's in it for us?* [Online; navštíveno 25.10.2017].
URL https://www.irt.org/articles/js086/

[5] Buba, V.: *Extrakce informací z webu založená na ontologiích*. Diplomová práce, Výsoké učení technické, Brno, 2017.

[6] Burget, R.: *Sémantický web a ontologie*. [Online; navštíveno 1.11.2017].
URL
https://www.fit.vutbr.cz/study/courses/PIS/private/cviceni/semweb2016.pdf

[7] Burget, R.: Information Extraction from the Web by Matching Visual Presentation Patterns. In *Knowledge Graphs and Language Technology: ISWC 2016 International Workshops: KEKI and NLP&DBpedia*, Lecture Notes in Computer Science vol. 10579, Springer International Publishing, 2017, ISBN 978-3-319-68722-3, s. 10–26.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11218

[8] LinkedDataTools.com, T.: *Tutorial 4: Introducing RDFS & OWL*. [Online; navštíveno 13.11.2017].
URL http://www.linkeddatatools.com/introducing-rdfs-owl

[9] Ora Lassila, R. R. S.: *Resource Description Framework (RDF) Model and Syntax Specification*. [Online; navštíveno 13.11.2017].
URL https://www.w3.org/TR/1999/REC-rdf-syntax-19990222

[10] OWL: *OWL Web Ontology Language*. [Online; navštíveno 5.11.2017].
URL https://www.w3.org/TR/2004/REC-owl-features-20040210/

[11] Pokorný, J.: *Webové uživatelské rozhraní nástroje pro extrakci informací*. Diplomová práce, Výsoké učení technické, Brno, 2017.

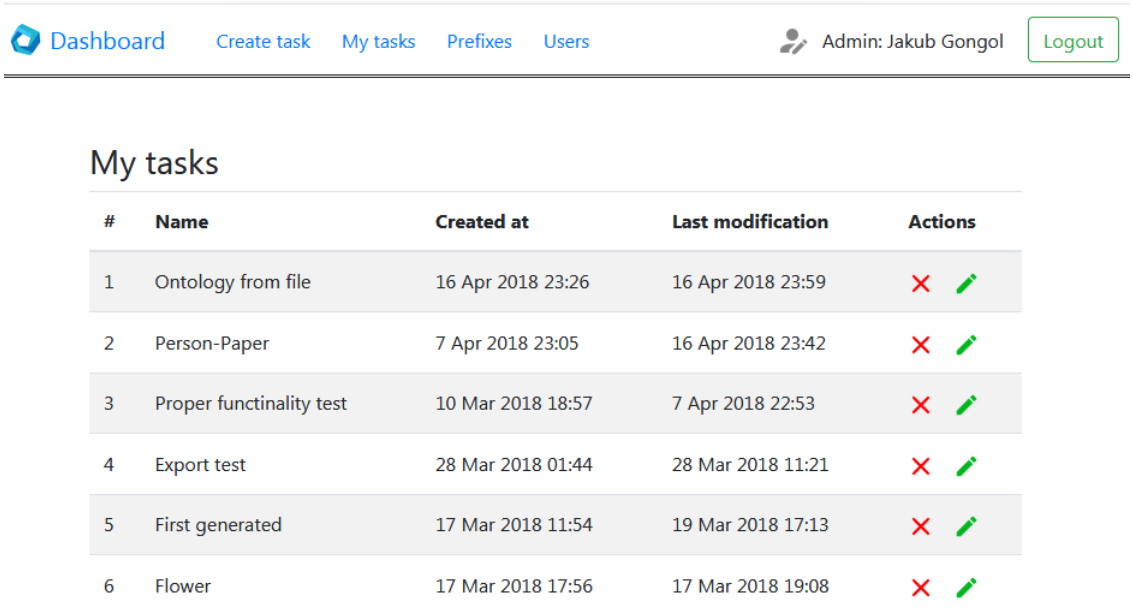
[12] W3C: *RDF 1.1 XML Syntax*. [Online; navštíveno 28.10.2017].
URL https://www.w3.org/TR/rdf-syntax-grammar

[13] W3C: *RDF Schema 1.1*. [Online; navštíveno 1.11.2017].
URL https://www.w3.org/TR/rdf-schema/

[14] W3C: *SEMANTIC WEB*. [Online; navštíveno 27.10.2017].
URL https://www.w3.org/standards/semanticweb

Příloha A

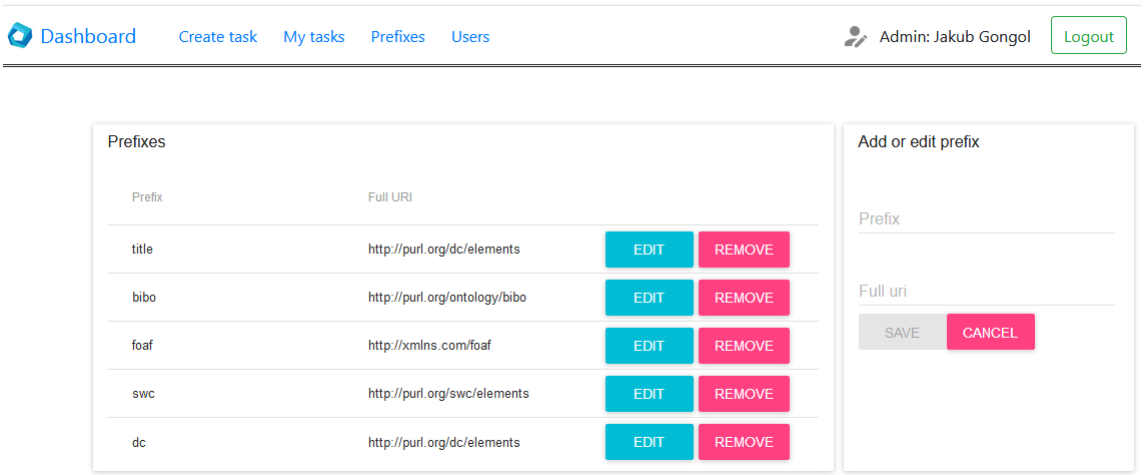
Obrazovka My Tasks



Obrázek A.1: Snímek obrazovky uživatelových úloh.

Příloha B

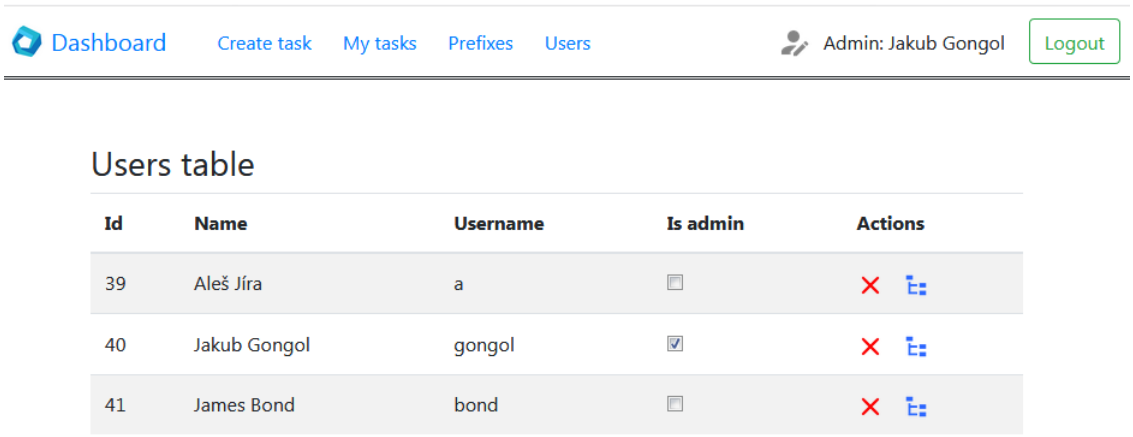
Obrazovka Prefixes



Obrázek B.1: Snímek obrazovky pro správu prefixů.

Příloha C

Obrazovka Users



Obrázek C.1: Snímek obrazovky pro správu uživatelů. Obrazovka je dostupná pouze uživatelům v roli administrátor.

Příloha D

Formát exportované úlohy


```
{
    "graph": {
      "id": 44,
      "values": {
        "title": "Export format appendix"
      },
      "nodes": [
        {
          "id": 1525262555310,
          "values": {
            "title": "name",
            "tagger": "",
            "object": false,
            "primary": false,
            "uris": []
          }
        },
        {
          "id": 1525262555309,
          "values": {
            "title": "Person",
            "tagger": "org.fit.layout.classify.taggers.PersonsTagger",
            "object": true,
            "primary": true,
            "uris": [
              "http://xmlns.com/foaf"
            ]
          }
        }
      ],
      "edges": [
        {
          "srcId": 1525262555309,
          "dstId": 1525262555310,
```

```
          "values": {
            "title": "",
            "cardinality": {
              "src": false,
              "dst": false
            },
            "optional": {
              "src": false,
              "dst": false
            },
            "uris": []
          }
        }
      ]
    }
}
```

Výpis D.1: Ukázka formátu exportu jednoduché extrakční úlohy tvořené grafem o dvou uzlech propojených hranou.

Příloha E

Obsah CD

```
.
|— ExtractionSystem
|      |— src/main
|      |      |— java            // zdrojový kód serverové části
|      |      |— js              // zdrojový kód klientské části
|      |      |— resources
|      |— ...
|— test
|      |— burget-papers.owl
|      |— foaf.rdf
|— tex                  // zdrojové soubory tohoto textu
|— README.txt     // instalační manuál
```