
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**
**BRNO UNIVERSITY OF TECHNOLOGY**

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**
**FACULTY OF INFORMATION TECHNOLOGY**

**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**
**DEPARTMENT OF INFORMATION SYSTEMS**


# EXTRAKCE INFORMACÍ Z WEBU ZALOŽENÁ NA ONTOLOGIÍCH

**ONTOLOGY BASED INFORMATION EXTRACTION FROM THE WEB**

**DIPLOMOVÁ PRÁCE**
**MASTER'S THESIS**


**AUTOR PRÁCE** **Bc. VOJTĚCH BUBA**
**AUTHOR**

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



**BRNO 2017**

*Zadání diplomové práce/19294/2016/xbubav00*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav informačních systémů Akademický rok 2016/2017

# Zadání diplomové práce

Řešitel: **Buba Vojtěch, Bc.**
Obor: Informační systémy
Téma: **Extrakce informací z webu založená na ontologiích**
**Ontology Based Information Extraction from the Web**
Kategorie: Web

Pokyny:

1. Seznamte se s pojmem ontologie a se souvisejícími jazyky pro definici ontologií, zejména RDFS a OWL.
2. Prostudujte metody extrakce informací z WWW se zaměřením na metody využívající konceptuálního modelování.
3. Navrhněte způsob transformace obecné vstupní ontologie např. v jazyce OWL na popis vhodný pro extrakci informací z webových stránek.
4. Navržený způsob transformace implementujte a integrujte do nástrojů pro extrakci informací vyvíjených na FIT.
5. Proveďte experimentální vyhodnocení vytvořeného nástroje na vhodných doménách.
6. Zhodnoťte dosažené výsledky.

Literatura:

- 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
- Dále dle pokynů vedoucího

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 2016
Datum odevzdání: 24. května 2017

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

Tato práce se zabývá problematikou extrakce informací z webu založenou na konceptuálním modelování pomocí ontologií. Cílem této práce je implementace nástroje, který zpracuje vstupní ontologie a nabídne uživateli možnost s ontologií dále pracovat skrze grafické uživatelské rozhraní. Práce seznámí čtenáře s jazyky RDF, RDFS a OWL, které slouží k zápisu ontologií. Dále jsou zde vysvětleny dvě extrakční metody, které jako popis extrahovaných informací používají právě ontologie. Výsledné řešení je navrženo tak, aby zohledňovalo všechny potřeby zadání extrakční úlohy, kterou definoval Ing. Radek Burget Ph.D. Implementovaný nástroj generuje soubor, který představuje zadání extrakční úlohy a to je následně použito pro samotnou extrakci informací. Ta je prováděna pomocí nástroje FITLayout, který je vyvíjen na FIT VUT v Brně.



# Abstract

The main aim of this thesis is an extraction of information from web based on conceptual modeling with ontology. The main goal of this thesis in question is implementation of a tool, which process input ontology and enable additional editing through graphical user interface. Readers of this thesis will be introduced with languages for writing ontologies, for example RDF, RDFS or OWL. Two extraction methods which use ontologies to describe extracted informations are also explained. Final solution is designed to consider all needs of extraction task defined by Ing. Radek Burget Ph.D. Output of this tool is definition of extraction task compatible with solution FITLayout, being developed at FIT BUT.



# Klíčová slova

extrakce informací z webu, ontologie, konceptuální modelování, sémantický web

# Keywords

semantic web, information extraction from web, ontology, conceptual modeling


# Citace

BUBA, Vojtěch. *Extrakce informací z webu založená na ontologiích.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Burget Radek.

# Extrakce informací z webu založená na ontologiích

## Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Vojtěch Buba
24. května 2017

## Poděkování

Děkuji panu Ing. Radku Burgetovi Ph.D. za pomoc při vypracování této diplomové práce.

# Obsah

# Kapitola 1

# Úvod

Internetové stránky jsou pro člověka a stejně tak pro stroje snadno čitelné. Bohužel ale, kde člověk dokáže odvozovat vazby mezi informacemi v textu a hledat potřebná data, je stroj bezmocný. Stroje nedokážou chápat data umístěná na internetových stránkách a odvozovat jejich kontext a vazby. Automatizace jakékoliv úlohy na webu je velice obtížná a vzhledem k objemu dat na internetu je manuální zpracování nemožné.

Aby bylo možné extrakci informací z webu automatizovat, je potřeba, aby stroj dokázal rozpoznat výskyt jednotlivých entit v rámci dokumentu a vazby mezi nimi. Sémantika informací může být v dokumentech vyjádřena, pomocí speciálních anotací. Ty lze zapisovat například v jazyce RDF, který umožňuje popis vazeb mezi daty a jejich vlastnosti.

Takto zapsané informace nazýváme metadata a díky nim mohou stroje lépe chápat obsah vyskytující se na stránce. Bohužel tento přístup má slabinu v podobě nutnosti definovat RDF popis pro každou webovou stránku. Jelikož chceme zpracovávat libovolnou webovou stránku, je tento přístup nedostatečný.

Existují různé přístupy k identifikaci entit vyskytujících se v dokumentu. Jeden z nich je založen, na vytvoření konceptuálního modelu cílové domény a ten je následně porovnán se strukturou informací dostupných v dokumentu. Konkrétně v přístupu z kterého tato práce vychází, jsou ontologie používány jako prostředek konceptuálního modelování, což umožňuje popsat data určená k extrakci pomocí jazyka RDF (OWL).

K obecnému popisu entit, jejich vlastností a vazeb, slouží ontologie. Konkrétně rodina jazyků OWL (Web Ontology Language) založených na ontologiích, které vychází ze standardu RDFS a rozšiřují jeho možnosti zápisu ontologií.

Ontologie slouží jako obecné popisy dat, které chceme z dokumentů získat. Tato práce se zabývá transformací obecných vstupních ontologií v jazyce OWL na popis vhodný pro extrakci informací z webových stránek. Výsledkem této práce je implementovaný nástroj, který umožňuje práci se vstupními ontologiemi a jejich převod do požadované reprezentace.

# Kapitola 2

# Sémantický web

Jedná se o pojem definovaný organizací W3C[1], který zaštiťuje nově vznikající technologie pro podporu tzv. "Webu tvořeného daty". Klasický web jak jej známe dnes, bychom mohli nazvat "Web tvořený dokumenty". Ten je navzájem propojený na úrovni dokumentů, ale data uvedená v dokumentech nejsou nijak propojená a pro strojové zpracování jsou nepoužitelná.

Web tvořený daty jsou tedy datumy, nadpisy, jména autorů, chemické prvky a nespočet dalších částí, z kterých se jednotlivé webové stránky skládají. Technologie sémantického webu nabízí prostředí, umožňující tvorbu dotazů nad těmito daty a vyvozování důsledků. Princip by se tedy dal přirovnat k tomu, co známe z klasických relačních databází.

Základem pro zveřejňování a propojování dat je jazyk RDF. Stejně jako u SQL databází i zde existuje speciální jazyk, pro dotazování se nad daty uloženými v RDF souborech. Jedná se o jazyk SPARQL[2], který umožňuje zasílat dotazy a získávat výsledky skrze HTTP nebo SOAP[3]. Ačkoli se jedná o technologii zajímavou, v této práci se jí dále nebudeme zabývat. Pro účel této práce je důležitý pojem tzv. slovníků, kterým se také říká ontologie. Ontologie definují koncepty a termíny použité k popisu a reprezentaci nějaké oblasti zájmu. Používají se k popisu hmotných i nehmotných věcí na abstraktní úrovni. Můžeme tedy například popsat auto jako celek, který se skládá z dílčích objektů, jako jsou třeba motor, dveře či kola. Dále můžeme definovat vazby mezi jednotlivými objekty. Například, že auto má dveře a omezení, auto má minimálně jedny dveře. Více informací o ontologiích a rodině jazyků na nich založených bude zmíněno později.

## 2.1 Jazyk RDF

Jazyk RDF přichází s možností zápisu metadat o informacích. Mohli bychom také říct, že lze zapisovat sémantiku, tedy význam informací. Příkladem může být knihovní katalog, který popisuje jednotlivé publikace v knihovně. Tato metadata slouží především pro stroje, kterým usnadňují pochopení obsahu, který popisují. Například internetové vyhledávače tak mohou na základě lepšího pochopení stránek poskytovat kvalitnější výsledky vyhledávání.

Při definování tohoto jazyka bylo hlavním cílem vytvořit mechanismus, který umožní použití pro jakoukoliv doménu informací a pro žádnou z nich nebude apriori definovat její význam.

[1] https://www.w3.org
[2] https://cs.wikipedia.org/wiki/SPARQL
[3] https://cs.wikipedia.org/wiki/SOAP

### Terminologie

Hlavním rysem datového modelu tohoto jazyka je tzv. trojice. Jedná se o základní stavební kámen, který slouží k popisu jakékoliv věci, kterou si dokážeme představit:

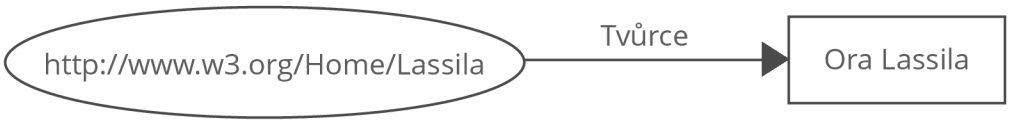
- **Zdroje** - Vše co popisujeme pomocí RDF výrazů nazýváme zdroje. Pod zdrojem si můžeme představit celou webovou stránku (článek na zpravodajském webu), část webového dokumentu (jako např. nadpis či odstavec), nebo celou síť webových dokumentů (celý zpravodajský web). Zdroj který popisujeme, samozřejmě nemusí být online a může jim být například i tištěná kniha. Zdroje jsou vždy pojmenovány pomocí URI a volitelným ID.
- **Vlastnosti** - Jedná se o soubor vlastností, aspektů, nebo vztahů, sloužící k popisu zdroje. Každá vlastnost má definováno jaké povolené hodnoty může nabývat, jaké typy zdrojů může popisovat a vazbu na další vlastnosti.
- **Tvrzení** - Tvrzení je tvořeno daným zdrojem, jeho pojmenovanou vlastností a její hodnotou. Tyto tři samostatné části můžeme také nazvat subjekt, predikát a objekt. Hodnota (data) kterou objekt nese, může být jiný zdroj (URI), nebo libovolný textový řetězec. Právě možnost zápisu zdrojů jako hodnoty objektu nám umožňuje vytváření vazeb mezi jednotlivými tvrzeními.

### Příklady

Jako příklad je uvedena následující věta: *Ora Lassila je tvůrcem zdroje http://www.w3.org/Home/Lassila.* Tato věta může být rozdělena na tři části:

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

Větu uvedenou výše, můžeme demonstrovat následujícím diagramem:

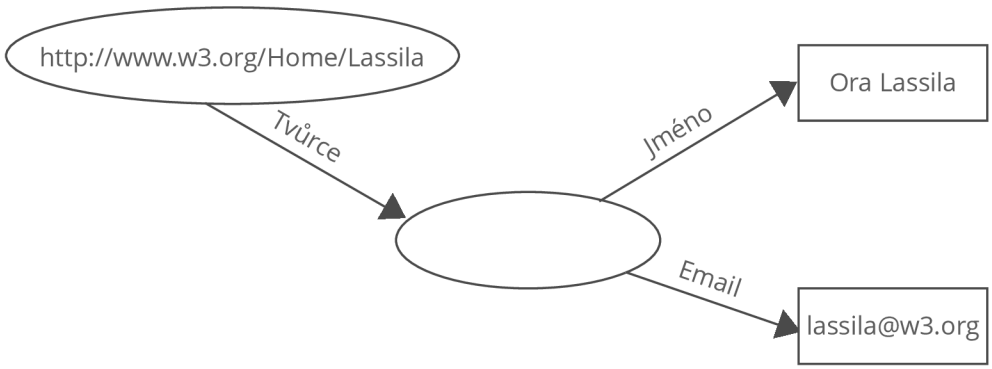



Obrázek 2.1: Diagram demonstrující jednoduché tvrzení.

V další ukázce uvažujme větu: *Jedinec, jehož jméno je Ora Lassila, email <lassila@w3.org>, je tvůrcem http://www.w3.org/Home/Lassila.*

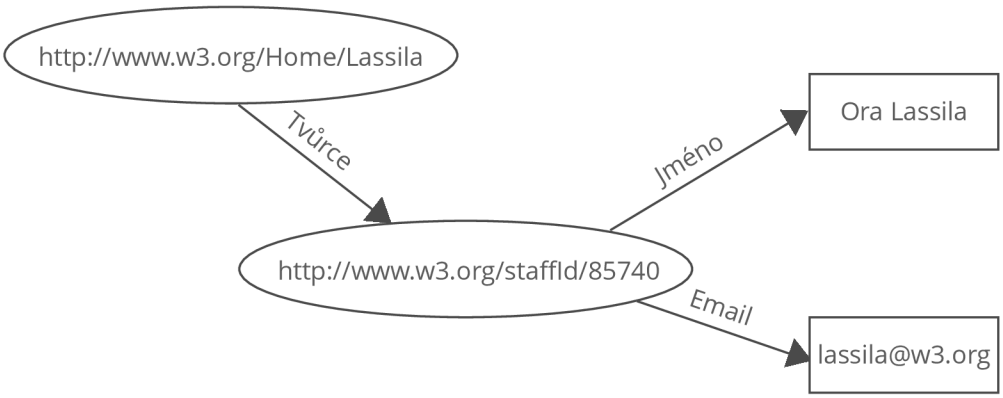
Tato ukázka demonstruje, že hodnota vlastnosti *Tvůrce*, může být také strukturovaná entita. V jazyce RDF je tento případ reprezentován jako další zdroj. Zdroj, který spojuje entity vlastnosti tvůrce, nenese žádné jméno a je anonymní.

V poslední ukázce je anonymnímu zdroji přiřazen jedinečný identifikátor. Volba identifikátoru se odvíjí od reprezentace zaměstnance v databázi a implementaci daného systému.




Obrázek 2.2: Diagram demonstrující vlastnost Tvůrce jako strukturovanou entitu.

V tomto případě je jednoznačný identifikátor ID zaměstnance. Konečné URI reprezentující daného zaměstnance, pak může vypadat například takto: *http://www.w3.org/staffId/85740.* Nyní uvažujme následující dvě věty: *Jedinec,vedený jako zaměstnanec ID 85740 se jmenuje Ora Lassila a má emailovou adresu lassila@w3.org. Tento jedinec je tvůrcem zdroje http://www.w3.org/Home/Lassila.* RDF model pro tuto větu je:




Obrázek 2.3: Diagram demonstrující vlastnost se strukturovanou hodnotou a přiřazeným identifikátorem (názvem).

### Syntaxe

Datový model jazyka RDF poskytuje abstraktní a konceptuální rámec pro definici a používání metadat. Pro účely vytváření a výměnu těchto metadat je potřeba konkrétní syntaxe. Ačkoli existuje vícero syntaxí a nové mohou vznikat, níže bude popsána ta základní založená na jazyce XML. Specifikace definuje dvě XML syntaxe pro kódování instancí datového modelu. První syntaxí je serializační, která poskytuje všechny možnosti zápisu datového

modelu. Druhou variantou je syntaxe zkrácená, která obsahuje další dodatečné konstrukce a její zápis je kompaktnější. Z pohledu interpretů lze použít libovolnou variantu, dokonce je možné syntaxe promíchat.

**Serializační syntaxe** dodržuje všechny konvence jazyka XML. Níže je uveden příklad zápisu věty *Zdroj http://www.w3.org/Home/Lassila vytvořil Ora Lassila.*

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

Listing 2.1: Ukázka serializační syntaxe jazyka RDF.

Uzel *RDF* je jednoduchý obal, který označuje hranice, mezi kterými se nachází instance datového modelu RDF. Tento uzel je volitelný a používá se hlavně v případech, kdy chceme v XML dokumentu explicitně uvést, že daná oblast má být chápána jako RDF. *Description* je uzel, který obsahuje zbylé části tvrzení a může být chápán jako jakýsi držitel identifikace zdroje, který je popisován. V případech kdy se k danému zdroji vztahuje více tvrzení, je tedy *Description* způsobem, jakým lze daný zdroj pojmenovat pouze jednou a na jednom jediném místě.

Dále lze v ukázce vidět použití atributu *about*, který nese hodnotu URI daného zdroje. Pokud uzel *Description* neobsahuje tento atribut, je tento zdroj považován za nově vytvořený. Uzel *Description* bez atributu *about* můžeme považovat jako zástupce nějakého fyzického zdroje, který nemá známé URI. V tomto případě lze použít atribut *ID*, který označuje tento zdroj a lze se na něj odkazovat z jiných uzlů *Description*, nebo uzlů popisující vlastnosti. V každém případě použití atributu *ID* vždy značí nově vytvořený uzel a použití toho atributu nelze kombinovat s atributem *about*. Hodnota atributu *ID* musí být vždy jedinečná v rámci celého dokumentu.

V rámci jednoho uzlu *Description* lze zapisovat více vlastností označených stejným jménem. Každá vlastnost potom vytvoří nový uzel pojmenované vlastnosti jako v diagramu, který jste mohli vidět v ukázkách dříve.

**Zkrácená syntaxe** může být použita v případech, kdy tvůrce RDF modelu chce, aby byl zápis více kompaktní. Ve zkrácené syntaxi jsou definovány tři druhy zkrácení původní serializační syntaxe. Některé z těchto zkrácení ovšem podléhají podmínkám, kdy mohou být použity. Pokud se v rámci uzlu *Description* neopakují vlastnosti a jejich hodnota je čistě textová, lze tyto vlastnosti zapsat jako atributy uzlu *Description*. Předchozí příklad by pak ve zkrácené syntaxi vypadal takto:

```
<rdf:RDF>
  <rdf:Description about="http://www.w3.org/Home/Lassila"
    s:Creator="Ora␣Lassila" />
</rdf:RDF>
```

Listing 2.2: Ukázka zápisu vlastností jako atributů uzlu ve zkrácené syntaxi jazyka RDF.

Toto je pouze obecné seznámení s jazykem RDF. Cílem této podkapitoly je představit čtenáři základní konstrukce jazyka a jeho roli v oblasti sémantického webu. Pro detailní seznámení s jazykem doporučuju specifikaci jazyka viz. [6]. Informace o jazyku RDF v této kapitole byly čerpány z [6].

## 2.2 RDF schémata

RDF schémata (RDFS) jsou jakousi nadstavbou jazyka RDF. RDFS využívá všech konstrukcí a mechanismů popsaných v předchozí podkapitole a doplňuje RDF model o možnost deklarování vlastností zdrojů, vytváření tříd a vazeb mezi jednotlivými zdroji a vlastnostmi. Jazyk RDF říká, jak můžeme nestrukturovaný text obalit značkami tak, aby jej stroje při zpracování mohly pochopit. Pomocí RDF samotného ale dokážeme říct pouze to, že nějaký zdroj má nějaké vlastnosti a ty můžeme případně strukturovat. RDF schémata přichází s vlastním typovým systémem, založeným na třídách a podtřídách, podobně jako je tomu u objektově orientovaných jazyků (OOP). Díky tomuto mechanismu můžeme definovat třídy, do kterých poté můžou zdroje a jejich vlastnosti patřit. Příkladem tedy může být třída *Pes*. O teto třídě můžeme prohlásit, že je podtřídou třídy *Savec* a o této třídě pak zcela jistě, že je podtřídou třídy *Zvíře*.

Motivací pro vytvoření schémat byla skutečnost, že pro konkrétní zdroje lze nalézt vlastnosti, které jsou pro ně typické. Příkladem budiž bibliografická citace. Zde se naprosto běžně vyskytují vlastnosti jako autor, titulek či předmět. Pomocí schémat tedy dokážeme deklarovat tyto vlastnosti, jejich význam a vazby mezi sebou. Samozřejmě lze pomocí schémat zapisovat i zdroje jako takové, ne pouze jejich vlastnosti. Zdroje jsou chápány jako instance tříd, které jim jsou deklarovány.

Samozřejmostí je využívání dědičnosti mezi třídami. Pokud je zdroj typu (patří do třídy) *Pes*, určitě můžeme prohlásit, že je i typu *Zvíře*. Pomocí vytváření tříd, jejich vlastností a vazeb, můžeme vytvářet celá schémata, kde každé může být určeno pro konkrétní doménu použití. Při vytváření RDF dokumentů, se poté používají již vytvořené schémata.

```
<rdf:RDF xml:lang="cs"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description ID="MotoroveVozidlo">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf
   rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Description>

<rdf:Description ID="OsobniVozidlo">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotoroveVozidlo"/>
</rdf:Description>

<rdf:Description ID="NakladniAuto">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotoroveVozidlo"/>
</rdf:Description>

<rdf:Description ID="Dodavka">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotoroveVozidlo"/>
</rdf:Description>
```

```
<rdf:Description ID="MalaDodavka">
  <rdf:type resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Dodavka"/>
  <rdfs:subClassOf rdf:resource="#OsobniAuto"/>
</rdf:Description>

</rdf:RDF>
```

Listing 2.3: Ukázka deklarace tříd a podtříd v RDFS.

Příklad výše demonstruje deklaraci třídy *MotoroveVozidlo* a jeho podtřídy *OsobniAuto*, *NakladniAuto*, *Dodavka* a speciální případ *MalaDodavka*, což je podtřída osobního auta a dodávky. Třída *MalaDodavka* tedy bude moci nabývat vlastnosti obou tříd. V příkladu lze nalézt konstrukce, které v tomto textu ještě nebyly zmíněny. Jednou z nich je odkazováni se na jmenné prostory. Tento mechanismu je právě určený pro odkazování se na již existující schémata. Bavíme se zde konkrétně o *xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns"*. Tento řádek deklaruje identifikátor *rdf* odkazující se na dané schéma. Skrze tento identifikátor se poté přistupuje ke zdrojům a vlastnostem, které jsou v tomto schématu deklarovány. Další neznámé konstrukce budou popsány později.

### Základní třídy

Součásti jmenného prostoru RDFS, vůči kterému jsou implicitně vytvářeny všechny RDF modely, jsou následující základní třídy:

- **rdfs:Resource** - Všechno co je popisováno RDF výrazy nazýváme zdroje a ty jsou považovány za instance třídy *rdfs:Resource*. Tato třída reprezentuje množinu zvanou *Resources* ve formálním modelu RDF.
- **rdf:Property** - Tato třída reprezentuje všechny elementy, které nazýváme vlastnosti. Současně se jedná o podmnožinu *RDF resources*.
- **rdfs:Class** - Tato třída představuje základní třídu, z které vycházejí všechny nově vytvořené třídy.

### Základní vlastnosti

Každý RDF model, který používá schéma, implicitně obsahuje tyto základní vlastnosti. Jsou to instance třídy *rdf:Property* a slouží k vyjadřováni vztahů mezi instancemi a jejich třídami, či nadtřídami.

- **rdf:type** - Slouží k zápisu, že zdroj je členem nějaké třídy. Každý člen třídy musí mít všechny charakteristické vlastnosti pro danou třídu. Hodnota této vlastnosti musí být vždy jiný zdroj, který je instancí třídy *rdfs:Class*.
- **rdfs:subClassOf** - Tato tranzitivní vlastnost vyjadřuje vztah podtřída/nadtřída mezi třídami. Vzhledem k tranzitivitě této vlastnosti platí, že pokud je třída A podtřídou třídy B a třída B je podtřídou třídy C, je třída A zároveň podtřídou třídy C. Tuto vlastnost mohou mít pouze zdroje, které jsou instancí třídy *rdfs:Class*. Třída může být podtřídou vícero tříd.

- **rdfs:subPropertyOf** - Tato vlastnost vyjadřuje specializaci nějaké obecnější vlastnosti. Vlastnost může být specializací jedné, vícero, či žádné vlastnosti. Platí zde pravidlo, že pokud má nějaký zdroj vlastnost, která je specializací jiné vlastnosti, vztahuje se na něj i obecnější varianta dané vlastnosti.
- **rdfs:seeAlso** - Tato vlastnost slouží ke specifikování zdroje, který by mohl poskytnout další informace o zdroji, kterého se vlastnost týká. Tuto vlastnost mohou mít deklarovanou pouze instance třídy *rdfs:Resource* a stejně tak hodnota této vlastnosti musí být instancí této třídy.
- **rdfs:isDefinedBy** - Tato vlastnost je specializací předchozí vlastnosti *rdfs:seeAlso* a indikuje zdroj, ze kterého daný zdroj vychází. Nejčastějším použitím je definování RDF schématu, kterým je definována nějaká vlastnost nebo třída.

Ve specifikaci[3] lze nalézt detailnější informace o výše zmíněných vlastnostech, jejichž popis v tomto textu je pouze informativní a pro účely této práce je dostačující.

### Definice omezení

V rámci definice schématu je možné zapisovat omezení jak nad vlastnostmi, tak nad třídami. Lze určit jakých hodnot mohou nabývat dané vlastnosti a jaké vlastnosti mohou být deklarovány dané třídě. Ačkoli RDFS poskytuje mechanismy jak dané omezení deklarovat, neříká už, jak mají aplikace (programy pracující s těmito informacemi) tyto informace zpracovávat.

Očekává se, že jiným způsobem je bude zpracovávat validátor hledající chyby v zápisu schématu a interaktivní editor navrhující vhodné hodnoty na základě atributů *rdfs:range* a *rdfs:domain*. Deklarace omezení není závislá na jmenném prostoru v kterém jsou deklarována. Lze deklarovat omezení napříč různými nezávislými schématy.

```
<rdf:RDF xml:lang="cs"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description ID="maZaregistrovano">
  <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#MotoroveVozidlo"/>
  <rdfs:range rdf:resource="#Osoba"/>
</rdf:Description>

<rdf:Description ID="mistoNaZadnichSedackach">
  <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#OsobniVozidlo"/>
  <rdfs:domain rdf:resource="#MalaDodavka"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/03/example/classes#Number
</rdf:Description>
</rdf:RDF>
```

Listing 2.4: Ukázka deklarace omezení domény a rozsahu v RDFS.

Příklad výše ukazuje možnosti použití nových omezení *rdfs:domain* a *rdfs:range*. U deklarace vlastnosti *maZaregistrovano* lze vidět omezení domény pouze na zdroj, který je

instancí třídy *MotoroveVozidlo*. Znamená to tedy, že tuto vlastnost lze použít pouze pokud je daný zdroj instancí třídy *MotoroveVozidlo*, nebo její podtřídy. Dále je zde demonstrováno použití omezení *rdfs:range*, které omezuje přípustné hodnoty této vlastnosti, pouze na instance třídy *Osoba*. U deklarace vlastnosti *mistoNaZadnichSedackach* lze vidět, že omezení domény může být deklarováno vícekrát, ovšem omezení rozsahu hodnot pouze jednou.

#### Značky pro základní omezení

- **rdfs:range** - Tato vlastnost říká, hodnoty jaké třídy může zdroj, kterého se tato vlastnost týká nabývat. Z logiky věci může být *rdfs:range* deklarován pouze u zdrojů, které jsou instancemi třídy *rdf:Property*. Například deklarujme vlastnosti třídy *Clovek*, konkrétně deklarujme vlastnost *vek*. U této vlastnosti chceme, aby její hodnota byla vždy číselná a ne jiná. Z tohoto důvodu bude její omezení hodnoty na instance třídy *Number*.

- **rdfs:domain** - Toto omezení deklaruje u instancí jakých tříd můžeme deklarovat tuto vlastnost. Vlastnosti můžou mít neomezené množství omezení typu *rdfs:domain*, ale zároveň nemusí mít žádné. Pokud toto omezení není deklarováno, lze danou vlastnost použít u jakékoliv třídy. Například deklarováni vlastnosti *mistoNaZadnichSedackach* u nákladního auta, které nemá zadní řadu sedadel by nedávalo smysl.

Díky mechanismům RDF schémat je možné plnohodnotně zapisovat sémantiku informací, které pomáhají strojům chápat informace, které zpracovávají.

Příklady a informace v této podkapitole byly čerpány z [3]

## 2.3 Jazyk OWL

Jedná se o rodinu jazyků celým názvem Web Ontology Language, založenou na standardu definující jazyk RDFS a rozšiřující tento standard o další vyjadřovací schopnosti v oblasti zápisu ontologií. Informace v této kapitole byly čerpány z [7] a [2].

### 2.3.1 Ontologie

Ontologie slouží k formálnímu zápisu struktur znalostí v různých oblastech použití. Podstatná jména reprezentují objekty a slovesa vztahy mezi nimi. Pomocí ontologií zapisujeme různé entity, jejich vlastnosti, typy a vztahy mezi nimi. Obecně tedy ontologie slouží k organizaci informací. Jejich struktura se podobá třídním hierarchiím v objektově orientovaném programování (OOP), ovšem je zde pár kritických odlišností. Třídní hierarchie použité ve zdrojových kódech se mění a vyvíjejí poměrně pomalu. Naproti tomu u ontologií se počítá s téměř konstantní změnou informací. Dalším rozdílem je flexibilita, kdy u ontologií se počítá s reprezentací informací pocházejících z různých heterogenních datových zdrojů.

### 2.3.2 Jazyky z rodiny OWL

Rodina jazyků obsahuje celkem tři varianty OWL, které se od sebe liší mírou vyjadřovacích schopností. Každý z podjazyků je syntaktickým rozšířením jeho předka. Vyjadřovací schopnosti jazyků rostou v pořadí v jakém jsou uvedeny níže. [2]

#### OWL Lite

Tato varianta byla původně vyvinuta pro ty uživatele, kteří potřebují především klasifikaci hierarchií a jednoduché podmínky. Příkladem omezenosti jazyka může být zápis mohutností, kde je povolen zápis pouze hodnot 0 a 1. Tvůrci jazyka doufali, že vývoj nástrojů pro tento jazyk bude snazší než pro jeho složitější varianty a umožní tak rychlý přechod z jiných organizačních systémů. V praxi se však ukázalo, že vývoj nástrojů pro OWL Lite je stejně složitý jako pro vyšší OWL DL, což vedlo k tomu, že dnes není OWL Lite příliš používaný.

#### OWL DL

Varianta tohoto jazyka OWL je pojmenovaná na základě jeho korespondence s deskriptivní logikou. OWL DL byl navržen tak, aby poskytoval maximální vyjadřovací možnosti a zároveň zachovával výpočetní úplnost, rozhodnutelnost a dostupnost efektivních rozhodovacích algoritmů. Tato varianta sice obsahuje všechny konstrukce jazyka OWL, ovšem mohou být použity pouze za určitých podmínek. Jednou z nich je například použití početního omezení na vlastnosti, které jsou deklarovány jako tranzitivní.

#### OWL Full

Poslední z variant používá jinou sémantiku, než předchozí dvě varianty. OWL Full byl navržen tak, aby zachovával kompatibilitu s RDF schématem. Díky OWL Full ontologie rozšiřují význam předdefinovaných slovníků. Jazyk jako takový je nerozhodnutelný a tedy žádný software jej není schopen zcela rozhodovat.

### 2.3.3 Terminologie

Jazyky OWL pracují s několika hlavními pojmy. Jsou to třídy, instance a vlastnosti.

#### Instance

Instance jsou objekty a mají stejný význam jako jedinec v deskriptivní logice.

#### Třídy

Třída je kolekce objektů a má podobný význam jako koncept v deskriptivní logice. Třída může mít jakýkoliv počet instancí a každá instance může patřit do žádné nebo vícero tříd. Třídy mohou tvořit hierarchie skrze dědičnost, kdy dědí vlastnosti jejich rodičů. Zároveň jsou všechny třídy podtřídou kořenové třídy *owl:Thing*. Opakem třídy *owl:Thing* je třída *owl:Nothing*, která je automaticky podtřídou každé třídy a na rozdíl od ostatních tříd, nemůže být nikdy instanciována. Tyto dvě třídy jsou používány pro tvrzení faktů, o všech nebo žádných instancích. Třídy a jejich členy můžeme v OWL definovat rozšiřováním, či pomocí stanovení nutných a dostačujících podmínek pro členství ve třídě.

#### Vlastnosti

Vlastnosti charakterizují třídu. Jinými slovy jde o binární relaci, která přiřazuje nějakou vlastnost instancím dané třídy. V deskriptivní logice odpovídá vlastnost pojmu role. Vlastnosti můžeme dělit na datatypové a objektové. Datatypové vlastnosti jsou vazby mezi instancemi tříd a RDF řetězci, či datovými typy XML schémat. Pro příklad jméno modelu je

vlastnost třídy výrobce a její datový typ je řetězec. Objektové vlastnosti jsou vazby mezi instancemi dvou tříd. Pro příklad třída auto má vlastnost majitel a ta může nabývat hodnot třídy člověk.

### 2.3.4 Používané syntaxe

Při práci s jazyky OWL se lze setkat s různými variantami syntaxí. Dělí se na vysokoúrovňové syntaxe zaměřené na specifikaci a syntaxe pro výměnu, které jsou více vhodné pro obecné použití.[8]

#### Vysokoúrovňová syntaxe

```
Ontology(<http://example.com/tea.owl>
    Declaration( Class( :Tea ) )
)
```

Listing 2.5: Ukázka OWL2 functional Syntax

#### Syntaxe pro výměnu

```
<Ontology ontologyIRI="http://example.com/tea.owl" ...>
    <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
        <Declaration>
        <Class IRI="Tea"/>
    </Declaration>
</Ontology>
```

Listing 2.6: Ukázka OWL2 XML syntax

```
Ontology: <http://example.com/tea.owl>
Class: Tea
```

Listing 2.7: Ukázka Manchester syntax

```
<rdf:RDF ...>
    <owl:Ontology rdf:about=""/>
    <owl:Class rdf:about="#Tea"/>
</rdf:RDF>
```

Listing 2.8: Ukázka RDF/XML syntax

Příklady byly čerpány z [2].

# Kapitola 3

# Extrakce informací z webu

V předchozích kapitolách je popsáno, co je to sémantický web a jak se liší od dokumentově orientovaného webu. Extrakcí informací z webu lze rozdělit do dvou skupin. První skupinu tvoří extrakce z dokumentů, které jsou popsány jazykem RDF (RDFS) a stroje dokážou chápat význam dat, které se nacházejí v daném dokumentu. Druhou skupinu tvoří extrakce z nestrukturovaných dokumentů, což je oblast ve které se stále vyvíjejí nové postupy a metody, jak tyto nestrukturované informace zpracovávat.

Jednou z metod je úprava již existujících dotazovacích jazyků tak, aby uměly pracovat s webem a data obsažená na webových stránkách mohly být dotazovány přímo skrze ně (pomocí jazyků). Dalším ze způsobů je metoda pochopení přirozené řeči. Cílem této metody je získáváni konceptuálních informací z textu, skrze kombinaci použití slovníků a analýzy vět. Tato metoda je ovšem závislá na větách, které jsou úplné. Existují i metody založené na využití tzv. obalů (anglicky wrapper), které mohou být zapisovány manuálně, nebo generovány pomocí nějakého stupně automatizace. Jedním z častých způsobů je generování obalů pomocí syntaktických vodítek jako jsou HTML značky. Dále v textu bude popsána jedna z metod, která je založená na používání obalů a kde generování probíhá automaticky ze vstupních ontologií. Posledním ze zde zmíněných způsobů extrakce je způsob, založený na vizuální reprezentaci vykresleného dokumentu. Tento způsob bude také podrobněji popsán v jedné z následujících kapitol.

## 3.1 Metoda extrakce s využitím wrapperů založená na ontologiích

Jedná se o metodu uvedenou v článku autorů D.W. Embley,D.M. Campbell, Y.S. Jiang, Y.-K. Ng, R.D. Smith [5]. Tato metoda používá pro definování wrapperů ontologicky popis dat, které budou z dokumentů extrahovány. Vstupem této metody je tedy instance modelu vztahů mezi objekty a dále slovníky obsahující jména a příjmení. Společně s ontologickým popisem dat tvoří základní kameny, z kterých jsou vytvořeny pravidla pro extrakci konstant a klíčových slov. Dále je na základě vztahů popsaných v ontologii vygenerován popis databáze, která bude obsahovat výstup extrakce. V tomto článku není řešen způsob jak klasifikovat webové stránky, které by byly vhodné pro extrakci požadovaných informací.

Před samotnou extrakcí je potřeba na stránce rozlišit jednotlivé záznamy dat. V příkladech použitých ve článku jsou to pohřební parte. V situaci, kdy může být na jedné stránce zveřejněno vícero pohřebních oznámení, je potřeba rozlišit, kde dané parte začíná a kde končí. Toto je řešeno sestavením stromové hierarchie z HTML značek dané stránky. Ve

stromu se pak pomocí vhodných heureistik hledá podstrom, který by měl obsahovat všechny záznamy (parte) a poté se hledá nejpravděpodobnější oddělovač mezi jednotlivými potomky v podstromu. Jakmile je nalezen oddělovač, jsou jednotlivé záznamy vyextrahovány.

Naplnění databáze probíhá ve dvou krocích. V prvním kroku jsou vyextrahované záznamy zpracovány rozpoznávačem klíčových slov a konstant, který naplní databázovou tabulku záznamy. Každý záznam představuje jeden řádek v tabulce a každá hodnota jednu buňku. Výstupem rozpoznávače je pro každé klíčové slovo či konstantu jeden řádek, který se skládá z popisu klíčového slova či konstanty, samotného klíčového slova či konstanty a počáteční a koncové pozice v textu, na kterých se dané klíčové slovo respektive konstanta nachází. Na základě dat v této tabulce jsou poté v druhém kroku spárovány vlastnosti s jejich hodnotou a uloženy jako databázové n-tice. Při spárování vlastností a hodnot se například využívá posouzení jejich vzájemné pozice v textu.

Pokud například blízko datumu je nalezeno klíčové slovo narozen, je velká pravděpodobnost, že se jedná o datum narození. Dále je využíváno popisu omezení a vztahů, které vycházejí z popisu ontologie a na základě kterých jsou použity heureistiky. Výsledkem aplikování těchto heureistik na nestrukturovaný datový záznam parte je množina vygenerovaných SQL příkazů, pro vložení dat do databáze.

Autoři této metody hodnotí výsledky, kterých dosáhli jako povzbuzující. Uvádějí 90% přesnost správného nalezení faktu v dokumentu, 75% přesnost správného nalezení jmén a 95% přesnost u všech ostatních faktů.

Uvedený popis je zjednodušený, jelikož mechanismy samotné extrakce nejsou předmětem této práce. Uvedený text slouží pouze k seznámení s danou problematikou. V případě zájmu o daný problém doporučuju studii uvedeného článku [5].

## 3.2 Metoda extrakce informací z webu založená na hledání visuálních vzorů

Konkrétně se jedná o metodu uvedenou v článku *Information Extraction from the Web by Matching Visual Presentation Patterns* jehož autorem je Ing. Radek Burget Ph.D.[4]. Návrh nástroje pro převod ontologií, který je výsledkem této práce, je navržen tak aby spolupracoval s experimentální knihovnou, která se vyvíjí na základě tohoto článku. Proto bude následující metoda rozepsána podrobněji a následující kapitola se bude zabývat návrhem řešení, který je založen na poznatcích plynoucí ze studia tohoto článku.

Stejně jako u předchozí metody i zde je cílem extrakce informací z dokumentů. Doména informací, které jsou cílem extrakce, je popsána pomocí vstupní ontologie. Klíčovou myšlenkou této metody je automatické nalezení vzorů, které představují datové záznamy ve vizuální reprezentaci zdrojového dokumentu a následně pak využít těchto vzorů k extrakci datových záznamu. Tato metoda se skládá ze tří fází:

1. **Vykreslení dokumentu** - Díky vykreslení dokumentu s kterým bude tato metoda pracovat, je metoda jazykově neutrální a použitelná pro jakýkoliv formát.

2. **Přibližné označkování obsahu** - Každému prvku v dokumentu, který by potenciálně mohl korespondovat s konkrétním datovým polem je přidána značka a hodnota pravděpodobnosti.

3. **Párování prezentačních vzorců** - Vyhledávají se základní prezentační vzorce použité pro prezentaci skupin označených oblastí, které odpovídají očekávané struktuře

hledaných záznamů. Vzorec, který se v dokumentu nakonec vyskytuje nejčastěji, je použit pro konečnou identifikaci cílových záznamů ve vstupním dokumentu.

V následujícím textu budou podrobně rozepsány principy ve všech zmíněných krocích.

### 3.2.1 Popis metody

Vstupní dokument je převeden na model dokumentu, který tvoří množina *vizuálních oblastí*. Ze stromu DOM vstupního dokumentu jsou uvažovány pouze textové uzly. Ty jsou posléze pomocí vykreslovacího enginu CSSBox transformovány na vizuální oblasti, které reprezentují daný text společně s jeho vypočtenými vizuálními vlastnostmi. Pod vizuálními vlastnostmi si lze představit velikost a pozici textového uzlu, velikost písma, barvy a další vlastnosti. Formálně je vizuální oblast definovaná následovně:

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

kde *text* je text obsažený v dané oblasti a $rect = (x,y,w,h)$ je popis obdélníku, který reprezentuje pozici a velikost ve vykreslené stránce. Styl textu v dané oblasti je definován jako

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

kde $fs$ je průměrná velikost písma použitá ve vizuální oblasti, $w \in [0, 1]$ je průměrná hodnota tloušťky písma (kde 1 znamená, že je celá oblast tučně a 0 znamená celou oblast napsanou normálním písmem), $st \in [0, 1]$ je průměrná hodnota stylu písma (kde hodnota 1 je celý text latinkou a 0 je normálně). Nakonec hodnota $c$ a $bc$ jsou barvy popředí a pozadí. Výsledkem vykreslení dokumentu je množina A všech vizuálních oblastí v dokumentu:

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

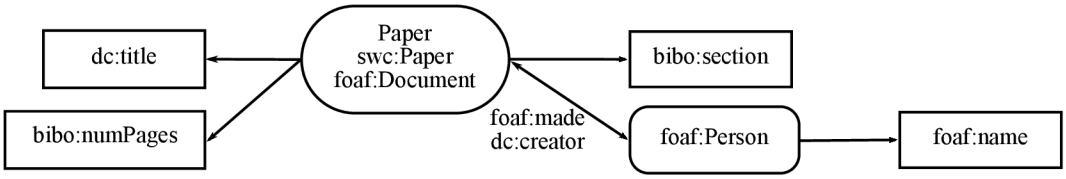
kde $m$ je celkový počet vizuálních oblastí v dokumentu. Každá vizuální oblast reprezentuje jeden textový uzel ve zdrojovém dokumentu.

Na základě ontologie popisující doménu informací, kterou chceme extrahovat je vytvořena množina značek (tagů), které budou přiřazeny jednotlivým vizuálním oblastem. Tagy jsou reprezentovány jejich jménem a reprezentují konkrétní datové pole, které má být extrahováno. Pro každý tag je definovaná funkce, která přiřadí podporu každé kombinaci tagu a vizuální oblasti. Hodnota kterou přiřazuje je od 0 do 1 a vyjadřuje pravděpodobnost, že daná oblast má význam, který koresponduje s přiřazeným tagem. Pokud je hodnota větší než 0, říkáme, že daná značka byla přiřazena dané oblasti s danou pravděpodobností. Pokud je hodnota podpory 0, říkáme, že značka nebyla přiřazena dané vizuální oblasti.

Pomocí počátečního značkování je dosaženo hrubého odhadu, který slouží pouze jako startovací bod pro další vylepšování. Některé značky nelze spolehlivě rozlišit, pokud jsou uvažovány jako vizuální oblasti samostatně. V tom případě daná oblast dostane značky obě (což znamená že koresponduje s oběma značkami), a později je dvojznačnost odstraněna použitím kontextu, ve kterém se vizuální oblast nachází.

Po fázi přibližného označení vizuálních oblastí tagy jsou odstraněny dvojsmysly pomocí uvažování kombinací datových polí, které očekáváme v extrahovaných datových záznamech (titulek - autor, titulek - stránky).

Metoda předpokládá, že všechny datové záznamy jsou prezentovány vizuálně konzistentním způsobem. Na základě tohoto předpokladu jsou definována prezentační omezení,




Obrázek 3.1: Ukázka ontologie reprezentující vědecký článek, jeho data a vlastnosti objektů. Články a vlastnosti vycházejí z bibliografické ontologie, FOAF ontologie a ontologie konference sémantického webu. Zaoblené obdélníky reprezentují vlastnosti objektu a obdélníky reprezentují jeho datové vlastnosti.[4]

která musí platit pro všechny datové záznamy, abychom je mohli považovat za vizuálně konzistentní.

Proces odstranění konfliktů dvojsmyslných vizuálních oblastí, se pak skládá z hledání nejlepšího vizuálně reprezentačního vzorce pro extrahované záznamy, které splňují vizuální konzistenci. Jako konečný vzorec je vybrán ten, který pokrývá co nejvíce označených vizuálních oblastí.

Metoda považuje datový záznam za vizuálně konzistentní tehdy, pokud má každé datové pole z kterého se záznam skládá, konzistentní prezentační styl a zároveň konzistentní rozložení.

### Konzistence stylu textu

Pro každé jednotlivé textové pole je požadováno, aby vizuální oblasti, které mají stejnou značku měly stejný vizuální styl v rámci celého dokumentu. Všechny vizuální oblasti se stejnou značkou představují množinu $A_t$. Následně je definována množina $S_t$, která obsahuje všechny styly prvků z množiny $A_t$. Následně je definován počet $n_f$ vizuálních vlastností, které mají stejnou hodnotu pro všechny styly z množiny $S_t$. Následně je určena prahová hodnota, která určuje jestli má nebo nemá množina $A_t$ konzistentní styl.

### Konzistence v rozložení datových polí

Toto omezení je založeno na předpokladu, že relace vzájemného rozložení jednotlivých datových polí v dokumentu je stejné pro všechny datové záznamy. V metodě jsou definovány čtyři relace, které popisují vzájemné pozice oblastí na stránce. Každá z nich $R \subseteq A \times A$.

- $(a_1, a_2) \in R_{side}$ - Tato relace platí, pokud jsou oba prvky na stejném řádku, $a_2$ je umístěné napravo od $a_1$, mezi těmito prvky není žádná další vizuální oblast a horizontální vzdálenost těchto dvou prvků je maximálně 1 em (délka znaku m).

- $(a_1, a_2) \in R_{after}$ - Tato relace platí, pokud jsou oba prvky na stejném řádku a $a_2$ je umístěno kdekoliv na řádku, napravo od $a_1$.

- $(a_1, a_2) \in R_{under}$ - Tato relace platí pokud jsou oba prvky umístěné zhruba ve stejném sloupci (jejich souřadnice x se překrývají) a $a_2$ je umístěné pod $a_1$ a to tak, že mezi nimi není žádná jiná vizuální oblast a vertikální vzdálenost mezi nimi není větší než $0.8em$.

- $(a_1, a_2) \in R_{below}$ - Tato relace platí tehdy, pokud jsou oba prvky umístěny zhruba ve stejném sloupci a $a_2$ je umístěné kdekoliv pod $a_1$.

Pro každý pár datových polí je vybrána ta relace, která pokrývá největší množství označených vizuálních oblastí.

### Spárováni vizuálních a sémantických vazeb

Při hledání kompletních datových záznamů je hledán nejčastěji se vyskytující vizuální vzor, který splňuje omezení vizuální konzistence a zároveň koresponduje s očekávanou sémantikou vztahů.

Když vezmeme v úvahu doménu popsanou ontologií v obrázku 3.1, najdeme binární relace s kardinalitou 1:N, nebo 1:1 mezi dvěma různými datovými vlastnostmi. Předpokládá se, že stejná sémantika vztahů mezi dvěmi vlastnostmi, bude reprezentována stejnou relaci rozložení, mezi dvěmi korespondujícími vizuálními oblastmi. Toto bude platit pro všechny datové oblasti na stránce. Zároveň všechny vizuální oblasti korespondující se stejnými datovými vlastnostmi budou mít konzistentní vizuální styl.

V ukázkové ontologií je možné identifikovat následující $1:N$ a $1:1$ relace, od kterých se očekává, že budou mít stejnou vizuální reprezentaci: *section - title, title - author, title - pages*. Jak lze vidět z utvořených dvojic, titulek může být považován za datové pole, které identifikuje celý datový záznam. Nyní uvažujme dvě množiny, které bychom mohli formálně definovat následovně:

$$A_{t_1} = \ a \in A : ((a, t_1), s) \in tagging \wedge s \geq s_{min} \tag{3.4}$$

$$A_{t_2} = \ a \in A : ((a, t_2), s) \in tagging \wedge s \geq s_{min} \tag{3.5}$$

Množiny $A_{t_1}$ a $A_{t_2}$ jsou tvořeny vizuálními oblastmi, které jsou označeny tagy $t_1, t_2 \in T$. Hodnota $s_{min}$ je prahová hodnota, která určuje, jestli je, či není tag přidělen dané vizuální oblasti. Na základě těchto dvou množin jsou posléze vybudovány množiny $S_{t_1}$ a $S_{t_2}$, které obsahují všechny styly vizuálních oblastí z množin $A_{t_1}$ resp. $A_{t_2}$. Následující definice je konfigurace extraktoru datových záznamů:

$$c = (s_{t_1}, s_{t_2}, R) \tag{3.6}$$

Konfigurace viz výše se skládá ze stylu $s_{t_1} \in S_{t_1}$, stylu $s_{t_2} \in S_{t_2}$ a jedné z relací rozložení, definovaných výše.

Pro každou takovou konfiguraci jsou nalezeny množiny odpovídajících párů vizuálních oblastí takových, že styl $a_1 \in A_{t_1}$ a $a_2 \in A_{t_2}$ musí být stejný jako prvku $s_{t_1}$ a $s_{t_2}$ uvedených v konfiguraci. Zároveň musí mít prvky z páru mezi sebou relaci definovanou v konfiguraci. Cílem je najít takovou konfiguraci, která bude obsahovat největší množinu korespondujících párů vizuálních oblastí. Tato množina je formálně definovaná viz níže:

$$M_c = (a_1, a_2) : a_1 \in A_{t_1} \wedge a_2 \in A_{t_2} \wedge style(a_1) = s_{t_1} \wedge style(a_2) = s_{t_2} \wedge (a_1, a_2) \in R \tag{3.7}$$

Páry vizuálních oblastí ve výsledné množině odpovídají stejným párům datových polí v datových záznamech. Jinými slovy byly nalezeny vizuální oblasti, které korespondují s individuálními datovými záznamy obsahující tyto dvě pole (použité tagy).

### Shrnutí

Tato metoda byla implementována v jazyce Java, použitím frameworku FITLayout[1]. Framework umí zpracovat HTML a PDF dokumenty za použití vykreslovacího enginu CCSBox[2]. Následná implementace byla otestována pomocí experimentů ve dvou různých aplikačních oblastech. Následující kapitola se bude zabývat návrhem řešení vycházejícího z článku popisující tuto metodu extrakce.

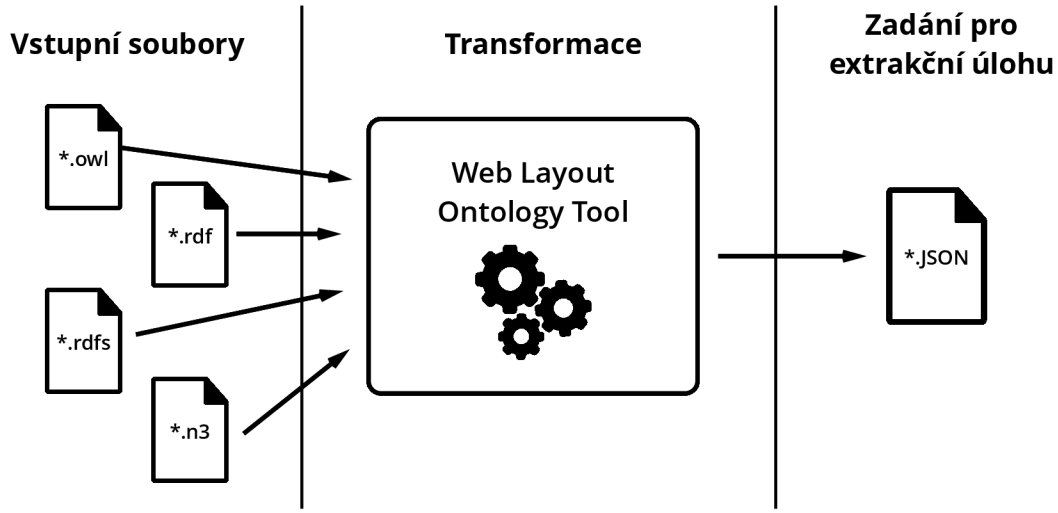
[1] http://www.fit.vutbr.cz/ burgetr/FITLayout/
[2] http://cssbox.sourcegorge.net/

# Kapitola 4

# Návrh řešení

V předchozí kapitole byla popsána extrakční metoda, založená na popisu extrahovaných dat pomocí ontologií. V této kapitole bude popsáno, jak jsou vstupní ontologie převedeny na výstupní reprezentaci, s kterou bude dále pracovat experimentální knihovna FITLayout. Výstupem tedy bude struktura, reprezentující zadání extrakční úlohy.

## 4.1 Analýza požadavků




Obrázek 4.1: Schéma očekávané funkčnosti výsledného řešení.

Z definice extrakční úlohy popsané v předchozí kapitole plyne, že nástroj musí být schopen zpracovávat soubory s konceptuálním modelem domény, která bude zpracovávána. Tento popis se může nacházet v jednom, nebo více souborech. Nástroj tedy musí být schopen tyto soubory zpracovat a zobrazit všechny ontologické třídy a jejich datotypové vlastnosti, které se v nich nachází.

Ačkoli se v rámci jednoho souboru může nacházet různý počet tříd a jejich vlastností, neznamená to, že mají být všechny tyto vlastnosti součásti zadání extrakční úlohy. Nástroj musí umožňovat libovolný výběr vlastností z jakékoliv načtené třídy.

Aby bylo možné úspěšně vytvořit zadání extrakční úlohy, je potřeba doplnit k vlastnostem další informace. Nástroj musí umožňovat nastavení ID taggeru, který bude použit pro výpočet pravděpodobnosti dané vlastnosti. Dále je potřeba vybrat klíčovou vlastnost, aby bylo možné vygenerovat dvojice vlastností. Pro každou tuto dvojici musí být nástroj schopen určit kardinalitu jejich vztahu, která bude při extrakci hledána mezi vizuálními oblastmi dané dvojice.

Nakonec bude možné tyto zpracované informace vyexportovat jako soubor ve formátu vhodném pro další strojové zpracování. Soubor představuje zadání extrakční úlohy a bude vstupem pro nástroj FITLayout.

## 4.2 Parsování vstupních ontologií

Jak již bylo zmíněno v předchozích kapitolách, ontologie se skládají ze tříd a vlastností. Tento popis bude vstupem do výsledného řešení (implementovaný program), který jej musí naparsovat a dále zpracovat. Parsováni je úloha, při které jsou z obecného textového dokumentu v určitém formátu získány požadované informace. K tomuto účelu poslouží knihovna OWL API[1], která dokáže vstupní ontologie převést do požadované reprezentace v jazyce Java. Tato knihovna dokáže zpracovávat tyto vstupní formáty:

- RDF/XML
- OWL/XML
- OWL Functional Syntax
- Turtle
- KRSS
- OBO Flat file format

Výstupem parseru v této knihovně jsou instance tříd, reprezentující vstupní ontologii. V dalším kroku budou tyto instance převedeny do vlastní reprezentace, která bude vhodná pro další zpracování.

## 4.3 Model reprezentace ontologií

Aby mohl nástroj vytvořit validní zadání extrakční úlohy, bude muset uživatel doplnit informace, které nelze z ontologie zjistit. Tuto akci bude provádět skrze grafické uživatelské rozhraní nástroje. Model reprezentace ontologie tedy musí počítat i s těmito dodatečnými informacemi. Z načtené ontologie jsou pro extrakční úlohu důležité především vlastnosti třídy. Ovšem vlastnosti mohou být dvou typů. Jedny jsou ty, které obsahují konečnou hodnotu (například řetězec) a druhými jsou instance jiných tříd (vyjádření vztahu). Pokud bude vstupní ontologie složitější, může obsahovat vícero tříd a vazby mezi nimi.

Objektové vlastnosti budou využívány především jako vyjádření vazby mezi dvěma třídami. Nikdy se ovšem nebudou vyskytovat jako vlastnosti reprezentující data, která se mají extrahovat. Hrají ovšem důležitou roli při výpočtu kardinality vztahu mezi dvěmi datotypovými vlastnostmi. Aby bylo možné provést propojení tříd skrze objektovou vlastnost, musí

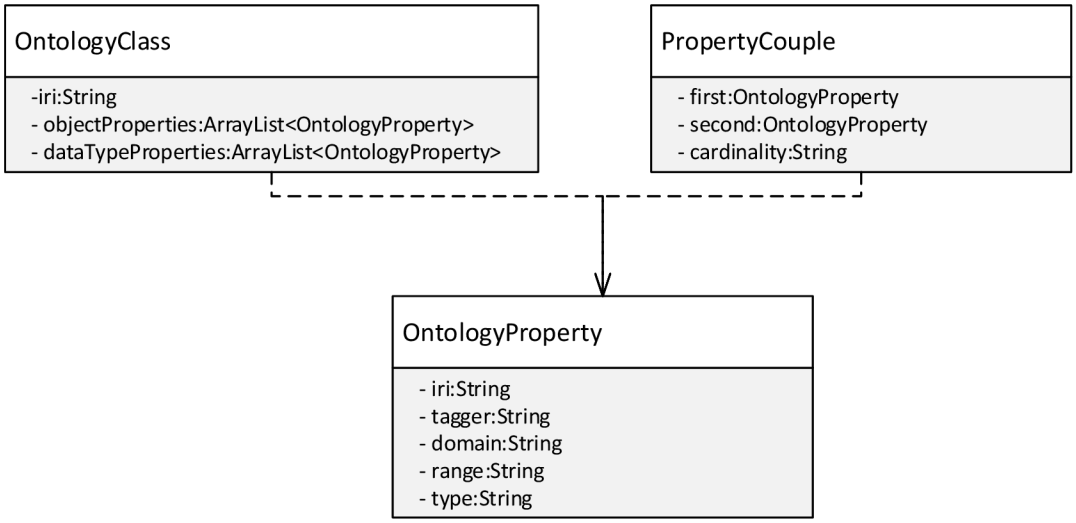
[1] http://owlapi.sourceforge.net

se definice této třídy nacházet v tom samém dokumentu, nebo musí být dodatečně načtena z jiného souboru. Je třeba podotknout, že ačkoliv vlastnosti obsahují tzv. IRI (Internationalized Resource Identifier)[2], které mohou mít tvar URL[3], neznamená to, že lze ontologii na dané adrese nalézt. IRI slouží pouze jako jednoznačný identifikátor. Pokud nebude znám popis ontologie reprezentující danou třídu, nebudou známy ani její datotypové vlastnosti a bude ignorována. Pokud bude popis ontologie nalezen, budou její vlastnosti zpracovány.

Každá vlastnost bude v modelu reprezentována vlastní třídou. Jakmile bude sestavený výběr vlastností určených pro extrakci, bude nutné vybrat jednu z nich, která bude sloužit jako klíčová pro extrahovaný datový záznam. Výběr tohoto klíčového tagu je důležitý při sestavování dvojic.

Z množiny vybraných tagů (vlastností) budou vygenerovány dvojice, skládající se z tagu a označené klíčové vlastnosti. Tyto dvojice budou reprezentovány speciální třídou, která bude obsahovat instanci každé z vlastností. Nástroj bude muset následně provést zjištění vazeb (1:N, 1:1, M:N) mezi datotypovými vlastnostmi v rámci dvojice. Tuto vazbu posléze zaznamená jako atribut instance třídy reprezentující dvojici.

V metodě extrakce je zmíněna tzv. tagovací funkce. Ta přiděluje vizuální oblasti a značce pravděpodobnost s jakou daná značka odpovídá vizuální oblasti. Aby bylo určení pravděpodobností co nejpřesnější, je pro každou značku zvolena funkce, která bude poskytovat co nejlepší výsledek. Tento výběr je založen na datotypové vlastnosti, kterou značka představuje a její výběr bude záležet na uživateli. Daná tagovací funkce bude reprezentována textovým identifikátorem.




Obrázek 4.2: Diagram tříd založený na navrhovaném řešení.

Z vytvořené reprezentace dat bude nakonec vyexportována struktura, reprezentující zadání pro extrakční úlohu.

[2] $https://en.wikipedia.org/wiki/Internationalized_Resource_Identifier$
[3] $https://cs.wikipedia.org/wiki/Uniform_Resource_Locator$

# Kapitola 5

# Implementace

V této kapitole je popsáno jakým způsobem byl výsledný nástroj implementován. Na začátku je podkapitola věnovaná softwarové architektuře nástroje. Podrobně je popsána každá vrstva zvoleného návrhového vzoru Model View Controller, včetně použitých technologií. Další podkapitoly jsou pak věnovány tématu transformace vstupních ontologií do výstupní reprezentace. Jsou zde popsány problémy vzniklé při transformaci, jejich řešení a použité algoritmy. Na úplném konci je podkapitola věnovaná funkci správce taggerů a popis exportu zadání extrakční úlohy.

## 5.1 Softwarová architektura

Při návrhu architektury nástroje byl použit návrhový vzor **Model View Controller**[1] (zkráceně MVC). Tento přístup dělí architekturu do tří vrstev, které mezi sebou komunikují a mají mezi sebou navzájem jen slabé vazby. Díky tomu je kód přehlednější a snáze se udržuje. Dále jsou popsány jednotlivé vrstvy, jejich význam a technologie, které byly použity pro jejich realizaci.

### 5.1.1 Model

Vrstva modelu se zabývá především práci s daty. Stará se o jejich uchovávání a případné změny. Veškerý výpočet je prováděn zásadně zde, a data nesmí být měněna z jiných vrstev. Pokud je potřeba vložit nová data, provést výpočet a nebo nějaká data smazat, je nutné k tomu použít patřičnou metodu, některé ze tříd ve vrstvě model. Díky tomu je vrstva modelu zcela nezávislá na ostatních vrstvách a je možné snadno upravovat zbylé vrstvy, bez nutnosti měnit model.

Vrstva modelu byla implementována v objektově orientovaném jazyce Java. Konkrétně je použito SDK ve verzi 1.8 a Java 8. Vazby mezi jednotlivými třídami, lze vidět na modelu domény níže.

Pro zpracování vstupních ontologií a jejich převod do vlastní datové reprezentace slouží třída *Parser*. Detaily toho převodu budou popsány ve speciální sekci, věnované této problematice později. Načítání vstupních ontologií není vzhledem k variabilitě jejich zápisu triviální úkol. Proto je pro zpracovaní samotných souborů použitá knihovna **OWL API**, která do velké míry načítání souborů usnadňuje. Samotná knihovna poskytuje celou řadu

[1] https://cs.wikipedia.org/wiki/Model-view-controller

dalších služeb pro práci s ontologiemi, ovšem ty nebudou v této práci dále rozebírány. Nejsou v kontextu této práce důležité a při implementaci nebyly využity.

Výsledná datová reprezentace je převedena do formátu JSON a následně uložena do souboru. K převodu tříd byla použita knihovna GSON, vyvinutá firmou Google. Samotný převod bude popsán ve speciální sekci později.

### 5.1.2 View

Jediným a hlavním úkolem vrstvy view, kterou bychom mohli česky nazvat vrstva pohledu, je převádění dat reprezentovaných modelem do vhodné reprezentace. Vrstva pohledu se tedy jednoduše stará o to, jak daná aplikace vypadá. Díky oddělení vrstvy modelu a pohledu, je možné upravovat vzhled aplikace bez nutnosti zasahovat do vrstvy modelu. V dnešní době, kdy se mnohem více dbá na uživatelskou přívětivost aplikací, je tento přístup obzvláště důležitý.

Pro implementaci vrstvy pohledu byla použita platforma JavaFX, která je přímo zaměřená na vývoj uživatelských rozhraní. Zvolena byla především pro její implicitní podporu architektury MVC. JavaFX nabízí programátorovi dva různé způsoby, jakými může být vrstva pohledu realizována. Prvním z nich je programátorský přístup, kdy se vrstva pohledu programuje stejně, jako vrstva modelu. Druhý způsob je realizace pomocí speciálních souborů s příponou **fxml**.

Jak už název přípony napovídá, jedná se o kód dost podobný jazyku **xml**. Zde jsou jednotlivé elementy uživatelského rozhraní definovány pomocí speciálních xml značek a skrze atributy je možné elementy nastavovat dle potřeby. Skrze speciální anotaci *@FXML* a ID elementů, je možné přistupovat k jednotlivým elementům i z třídy řadiče a dále je za běhu programu dle potřeby měnit. Pomocí obou přístupů je možné vytvořit ekvivalentní vzhled programu.

Při realizaci rozhraní nástroje byl použit druhý způsob tvorby a to hlavně s ohledem na přehlednost kódu. Zápis rozhraní pomocí speciálních značek zachovává přirozenou hierarchii elementů v kódu a díky tomu se v něm lépe orientuje. Posledním nezanedbatelným důvodem pro volbu tohoto způsobu je také to, že je možné pro vytvoření jednotlivých šablon rozhraní použít volně dostupný program **SceneBuilder**[2]. Ten tvorbu rozhraní podstatně usnadňuje a urychluje, jelikož je možné vzhled jednoduše tvořit přetahováním jednotlivých komponent myší.

Technologie JavaFX dále nabízí možnost definice vzhledu programu pomocí technologie CSS [3], která je používána především u webových stránek. Tato technologie umožňuje snadné znovupoužití definovaných kaskádových tříd u více komponent a docílit tak lepší konzistence vzhledu napříč celou aplikací. Další výhodou je také fakt, že pro změnu grafiky celé aplikace, stačí pouze vyměnit soubor s definovaným CSS předpisem a není nutné jakkoliv zasahovat do kódu samotné aplikace. Tento fakt opět přispívá ke snadnější udržovatelnosti tvořeného kódu.

### 5.1.3 Controller

Úkolem vrstvy controller, česky řečeno řadiče, je zpracovávat akce, vyvolané uživatelem při používání uživatelského rozhraní. Při kliku na tlačítko, tedy řadič příjme požadovanou akci a provede požadované akce. Důležité je zmínit, že ve vrstvě řadiče nikdy nedochází

[2] http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html
[3] https://www.w3.org/standards/webdesign/htmlcss

k manipulaci s daty, k tomuto slouží vrstva modelu. Řadič pouze deleguje akce na vrstvu modelu a zajišťuje případné změny ve vrstvě view.

Technologie vrstvy řadiče je úzce spjatá s technologií vrstvy pohledu. Každý pohled v aplikaci má svůj vlastní řadič, který obsluhuje všechny akce, které se v něm odehrávají. Propojení třídy řadiče a patřičného view probíhá skrze speciální fxml značku *fx:controller* v souboru view. Toto propojení umožňuje přistupovat k jednotlivým elementům uživatelského rozhraní skrze speciální anotace a také předávat do vrstvy pohledu data, která se mají zobrazit.

Pro přístup k elementům uživatelského rozhraní ze třídy řadiče, je třeba vytvořit proměnnou patřičného datového typu a uvést u ní anotaci *@FXML*. Následně už se s elementem pracuje stejně, jako s elementem vytvořeným programovým způsobem. Skrze anotované proměnné tedy probíhá například inicializace zobrazovaných dat v inicializační fázi.

## 5.2 Zpracování vstupních ontologií a převod do vlastní reprezentace

Jak už zbylo zmíněno v předchozí podkapitole věnované softwarové architektuře, je načítání ontologických souborů řešeno pomocí *OWL API*. Pro načtení ontologie ze souboru je použit tzv. *OntologyManager*, který vrací instanci třídy *OWLOntology* reprezentující celou ontologii.

### 5.2.1 Získáni ontologických tříd

Zavoláním metody *getClassesInSignature* nad instancí třídy reprezentující načtenou ontologii je získán seznam všech tříd, které se v ontologii nacházejí. Seznam obsahuje instance třídy *OWLClass*, která pochází z použité knihovny OWL API. Z těchto instancí je už posléze snadno získáno *IRI* jednotlivých tříd v plné i zkrácené formě.

Tyto dva údaje stačí pro vytvoření nové instance vlastní třídy *OntologyClass*, s kterou už se bude v modelu dále pracovat. Důvodem pro převod ontologických tříd do vlastní reprezentace je usnadnění další práce s třídami, kdy je žádoucí, aby třída obsahovala informace o svých datových a objektových vlastnostech.

### 5.2.2 Získání datových a objektových vlastností

Získávání datových a objektových vlastností je velice podobné, ovšem každá akce se provádí zvlášť. Knihovna bohužel neumožňuje jednoduché získání vlastností všech typů zavoláním metody nad instancí třídy *OWLCLass*. Získáni probíhá voláním metody *getAxioms* nad ontologií a jako parametr se předává typ axiomu, který je požadován.

Celý proces získávání vlastností ontologických tříd je realizován pomocí zanořených cyklů, kdy vnější cyklus prochází seznam instancí *OWLClass* a uvnitř jsou další dva cykly. Každý cyklus prochází seznam získaných axiomů (axiomů vlastností) a pokud axiom patří do aktuální třídy ve vnějším cyklu, je dále zpracován. Náležitost axiomu ke třídě je zjištěna pomocí objektové vlastnosti domain, kterou instance obsahuje. Další zpracování axiomu spočívá v jeho převedení do vlastní reprezentace v podobě třídy *OntologyProperty* a přiřazení instancí třídy *OntologyClass*.

Vytváření instancí se provádí v tovární třídě *OntologyPropertyFactory*, kde na základě typu axiomu (datotypová nebo objektová vlastnost) dochází ke správné inicializaci vytvářených instancí. Ačkoli tovární metoda *createProperty* vrací správně inicializované instance

vlastností, je potřeba dále inicializovat další atributy. Jedná se o atributy *domain* a *range*. Získáni informace o doméně vlastnosti je triviální, jelikož víme, jaké třídě vlastnost náleží. Ovšem získáni informací o atributu range je náročnější.

Získáváni atributu range se provádí pro každý typ ontologické vlastnosti zvlášť. Opět je potřeba z ontologie získat pomocí metody *getAxioms* všechny axiomy, které reprezentují atribut range datové vlastnosti, resp. objektové vlastnosti. Pro každý typ vlastnosti je vytvořená hash mapa a axiomy jsou uloženy jako dvojice klíč a hodnota. Klíč reprezentuje IRI vlastnosti, ke které atribut range patří a hodnota je reprezentována celou instancí axiomu. Takto získané informace o atributech range je poté možné přiřadit vlastnostem, ke kterým náleží. Tyto informace jsou získány ještě před samotným vytvářením instancí ontologických vlastností a jejich přiřazení probíhá současně s přiřazováním informací o doméně vlastnosti.

Takto vytvořené a inicializované instance jsou přiřazeny patřičné třídě, čímž končí převod tříd v ontologii do vlastní reprezentace. V této fázi už chybí pouze kontrola ekvivalentních tříd, kterou se bude zabývat další sekce.

## 5.3 Řešení problematiky ekvivalentních tříd

Jak už bylo zmíněno v teoretické části této práce, ontologické třídy obsahují 0 až N vlastností. Tyto vlastnosti mohou být nově deklarované, nebo použity již existující. Každá tato vlastnost má atribut *range*, který omezuje rozsah hodnot, které může vlastnost nabývat. Obzvláště u objektových vlastností pak může být problémem to, že daná vlastnost má *range* omezený na datový typ, který není v ontologii použitý.

Tento problém je řešen pomoci použití deklarace **ekvivalentních tříd**. Tento konstrukt umožňuje překonat problém s typovým omezením vlastností a používat již existující ontologie a jejich vlastnosti při vytváření ontologií nových.

### 5.3.1 Převod seznamu ekvivalentních tříd na matici sousedních uzlů

Pro získáni seznamu ekvivalentních tříd deklarovaných v ontologii je použito OWL API. Jako u objektových a datových vlastností je zde znovu použita metoda *getAxioms*, která vrátí množinu dvojic. Celkové zpracování této množiny probíhá ve třech fázích.

V první fázi je potřeba získat seznam všech tříd, které mají nějakou ekvivalenci a to tak, aby se v seznamu každá třída vyskytovala pouze jednou. Výsledek tohoto kroku je seznam všech ekvivalentních tříd. Toto je řešeno pomocí dvojitého zanořeného cyklu. Ve vnějším cyklu je z axiomu získán seznam všech tříd, které jsou si navzájem ekvivalentní. Tento nově vytvořený seznam je poté zpracován ve vnitřním cyklu a třídy, které se ještě nenacházejí ve výsledném seznamu jsou do něj přidány.

V druhé fázi je ze seznamu ekvivalentních tříd vytvořená dvourozměrná matice sousedních uzlů, která bude dále použita při procházení grafu pomocí algoritmu **Depth First Search**. Řádky a sloupce matice jsou vytvořeny pomocí dvou zanořených cyklů a to pro každou různou dvojici tříd bez ohledu na to, jestli mezi sebou mají vztah ekvivalence.

Samotná matice je reprezentována jako samostatná třída. Tento přístup byl zvolen vzhledem k netriviálnímu řešení indexace dvourozměrné matice pomocí textových řetězců. Indexace pomocí řetězců byla nutná s ohledem na to, že se v této fázi pracuje především s URI tříd, které jsou navíc unikátní.

V třetí a poslední fázi jsou v matici označeny vztahy ekvivalence pomocí pravdivostní hodnoty *true*.

### 5.3.2 Vybudování grafu

Pro použití Depth first search (prohledávání do hloubky) algoritmu je potřeba vybudovat graf, který bude procházet. Jakýkoliv graf obecně se skládá z uzlů a hran, které je propojují. V předchozí sekci bylo popsáno jak se vytváří matice sousedních uzlů. Ta reprezentuje hrany grafu, kdy hodnota *true* značí existenci hrany a *false* neexistenci. Posledním krokem je vytvoření samotných uzlů. Ty jsou vytvořeny v jednom cyklu a to pro každou třídu ze seznamu ekvivalentních tříd.

### 5.3.3 Proč byl vybrán Depth first search (DFS)

Očekávaným výsledkem po provedení průchodu grafem, je nalezení takových skupin tříd, které budou obsahovat všechny navzájem ekvivalentní třídy. Knihovna OWL API bohužel vrací pouze dvojice ekvivalentních tříd. Ačkoli se může jevit jako vhodné řešení vytvoření tranzitivního uzávěru, není tomu tak. U tranzitivního uzávěru by správné nalezení oněch skupin záleželo na pořadí tříd v seznamu a tudíž by správnost výpočtu nebyla zaručena.

Výsledkem průchodu **DFS** grafem je seznam všech uzlů, které z počátečního uzlu navštívil. Tento výsledek je přesně to, co je od algoritmu očekáváno. Algoritmus prohledávání do šířky[4] nebyl použit, jelikož nás v tomto případě nezajímá nejkratší cesta, ale všechny dostupné uzly.

### 5.3.4 Vyhledávání pomocí DFS

Jedná se o grafový algoritmus pro průchod grafem, používající metodu backtrackingu.

```
procedure DFS-iterative(G,v):
  let S~be a stack
    S.push(v)
      while S~is not empty
        v~= S.pop()
        if v~is not labeled as discovered:
          label v~as discovered
          for all edges from v~to w in G.adjacentEdges(v) do
            S.push(w)
```

Listing 5.1: Ukázka pseudokódu implementace iterativní varianty DFS

Výsledek prohledávání grafu je seznam URI všech tříd, které spadají do jedné ekvivalenční skupiny. Ty jsou uloženy do nového seznam seznamů, který reprezentuje všechny nalezené skupiny. Prohledávání grafu se provádí pro každou třídu ze seznamu ekvivalentních tříd. Pokud už ovšem daná třída spadá do některé z již vytvořených ekvivalenčních skupin, prohledáváni se nekoná. Výsledkem průchodů je tedy seznam ekvivalenčních skupin, které jsou dále převedeny do vlastní reprezentace. Popis toho převodu bude v následující sekci. [1]

### 5.3.5 Vytvoření skupin ekvivalenčních tříd

Několikanásobným průchodem grafu je získán seznam ekvivalenčních tříd, které je potřeba převést do vlastní reprezentace. Ty jsou instancemi třídy **ClassEquivalencyGroup**. Jedná se o speciální třídu, která uchovává informace o všech členech skupiny. Zároveň sjednocuje

[4] https://en.wikipedia.org/wiki/Breadth-first_search

chování celé skupiny tak, že se tváří jako jedna třída. Toho je docíleno tím, že stejně jako třída *OntologyClass* implementuje rozhraní *IOntologyClass*. Celá skupina je reprezentovaná klíčovou třídou, která propůjčuje skupině své jedinečné URI.

Vytvoření ekvivalenčních tříd je velice jednoduché a provádí se průchodem seznamu seznamů získaného v předchozí sekci. První třída v seznamu je vždy určena jako klíčová a následně je s její pomocí vytvořena nová instance ekvivalenční skupiny. Členy skupiny jsou již vytvořené instance třídy *OntologyClass*, které vznikly při zpracování vstupní ontologie. Přiřazení instancí tříd probíhá v cyklu na základě jejich URI.

Nyní jsou zpracovány všechny ontologické třídy, které mají definovanou nějakou ekvivalenci. Pro zjednodušení další implementace jsou ekvivalenční třídy vytvořeny i pro třídy, bez definované ekvivalence. Jsou to tedy ekvivalenční třídy s jedním členem. Pokud by v další načtené ontologii (z jiného souboru) byla definována ekvivalence s některou z těchto tříd, je již ekvivalenční skupina přichystána a pouze se doplní o další ontologické třídy.

## 5.4 Vyhledávání cest v grafu mezi dvojicemi

V kapitole zabývající se metodou extrakce informací založenou na vizuálních vzorech bylo zmíněno, že metoda pracuje s páry vizuálních oblastí. Tyto páry jsou tvořeny datotypovými vlastnostmi jednotlivých tříd. Uživatel má možnost v uživatelském rozhraní nástroje vybrat libovolný počet vlastností, čímž určí, jaké informace se budou z dokumentu extrahovat. Z těchto dvojic je následně vybrána klíčová vlastnost, která tvoří dvojice s každou ze zbylých vlastností.

Pro každou dvojici je nutné zjistit kardinalitu jejich vztahu, jelikož se očekává, že stejná kardinalita vztahu bude reprezentovaná relací rozložení, mezi dvěmi vizuálními oblastmi. Pro zjištění tohoto vztahu je potřeba sestrojit graf a najít cestu mezi dvojicí vlastností. Na základě této cesty je pak určována kardinalita jejich vztahu.

### 5.4.1 Vytvoření dvojic

Dvojice je reprezentovaná speciální třídou **PropertyCouple**, která obsahuje atributy *first* a *second*. První prvek dvojice je vždy klíčový. Jeho výběr záleží pouze na uživateli. Po načtení souboru s ontologií jsou v uživatelském rozhraní zobrazeny všechny dostupné datotypové vlastnosti. Uživatel následně vybírá ty, které chce použít pro vytvoření dvojic. Dvojice jsou následně tvořeny pouze nad tímto seznamem vybraných vlastností. Samotné vytvoření dvojic je triviální a spočívá v pouhém průchodu seznamu a vytvoření instancí dvojic.

### 5.4.2 Vybudování grafu

Graf je vytvořen nad všemi načtenými ontologickými třídami a jeho vytvoření probíhá v několika krocích. V prvním kroku jsou vytvořeny uzly a hrany pro všechny třídy. Pro každou třídu je vytvořen nový uzel, který má jako své ID nastaveno URI třídy. Následně je pro každou datotypovou vlastnost, kterou třída obsahuje, vytvořen nový uzel. Uzel třídy a vlastnosti je poté spojen novou hranou, která vede od uzlu třídy k vlastnosti a další hranou, kterou vede od vlastnosti ke třídě. Důvodem pro toto zdvojení je použití varianty algoritmu, který funguje pouze na orientovaných hranách. Všechny nově vytvořené uzly jsou ukládány jako dvojice *<řetězec, uzel>* do objektu typu *HashMap*. Ten je použit v druhém kroku pro snadné vyhledávání již existujících uzlů na základě jejich ID (URI).

V druhém kroku probíhá spojování ontologických tříd skrze objektové vlastnosti. Nejprve jsou stejně jako u datotypových vlastností vytvořeny hrany vedoucí z objektové vlastnosti do třídy a naopak. Každá objektová vlastnost má atribut *range*, který udává jakých hodnot může vlastnost nabývat. V případě objektových vlastností jsou to vždy vazby na jiné třídy. Dále je provedeno hledání již existujícího uzlu třídy, na základě hodnoty v atributu *range* objektové vlastnosti. Pokud je uzel třídy nalezen, je vytvořena hrana mezi vlastnosti a třídou v obou směrech. Pokud nebude uzel nalezen, neznamená to, že se vyskytla chyba. Následně se provede hledání mezi všemi třídami ve všech skupinách. Pokud je taková třída nalezena, je vytvořena nová hrana mezi klíčovou třídou skupiny ekvivalence a objektovou vlastností v obou směrech. Pokud třída nebyla ani v jednom případě nalezena, znamená to, že v načtených ontologiích nebyla deklarována a nelze k ni vytvořit hranu.

### 5.4.3 Použití Dijkstrova algoritmu

Očekávání od použitého algoritmu pro průchod grafem jsou v tomto případě zcela jiná. Na rozdíl od hledání ekvivalenčních skupin je nyní pomocí dvojice specifikovaný jak start tak cíl. Z toho vyplývá, že bylo potřeba najít vhodný algoritmus pro nelezení cesty v grafu. Vzhledem k tomu, že není k dispozici žádná heuristická funkce, nemohla být zvolena žádná z informovaných metod. Jak už nadpis sekce napovídá, byl zvolen **Dijkstrův algoritmus**. Konkrétně je použita jeho varianta pro orientovaný graf. Tento nedostatek je odstraněn pomocí zdvojení hran při sestavování grafu. Všechny hrany hrany mají stejnou váhu ohodnocení 1.

```
function Dijkstra(E, V, s):
    for each vertex v~in V:
        d[v] := infinity
        p[v] := undefined
    d[s] := 0
    N :=
V~while N is not empty:
        u~:= extract_min(N)
        for each neighbor v~of u:
            alt = d[u] + l(u, v)
            if alt < d[v]
                d[v] := alt
                p[v] :=
u~
```

Listing 5.2: Ukázka pseudokódu implementace Dijkstrova algoritmu

Očekávaným výsledkem hledání Dijkstrova algoritmu je cesta od prvního ke druhému prvku dvojice. Tato cesta je posléze použita k výpočtu Kardinality vztahu ve dvojici. Hledání cesty je samozřejmě nutné provést pro každou dvojici. Graf je ovšem pro všechny výpočty použit pouze jeden, a tak se režie na jeho výpočet rozmělní v počtu dvojic.

## 5.5 Správce taggerů

Jednou z informací, kterou je potřeba u vybraných datotypových vlastností doplnit, je id taggeru, který bude použit pro přiřazení pravděpodobností k vizuálním oblastem. Z tohoto důvodu disponuje nástroj funkcí tzv. správce taggerů. Ta umožňuje uživateli vytvářet nové

taggery a případně je i smazat. Tagger je implementovaný jako jednoduchá třída, která má dva atributy id a jméno. Uživatel zadává do formuláře obě tyto informace a ani jedna se negeneruje automaticky. Je zde ponechána volnost a zadávané informace jsou na zodpovědnosti uživatele. Tento přístup byl zvolen s ohledem na použití výstupu v nástroji FITLayout tak, aby zde byla ponechána dostatečná variabilita.

Správce všechny vytvořené taggery před vypnutím aplikace ukládá do konfiguračního souboru ve formátu json. Tyto informace zase naopak načítá vždy při startu aplikace. Předpokládá se, že uživatel bude často používat stejné taggery pro různá zadání extrakčních úloh a proto by mu tato vlastnost měla usnadnit časté používáni nástroje.

## 5.6 Export zadáni extrakční úlohy

Pro uchování všech informací, potřebných k vygenerování zadání extrakční úlohy, slouží speciální třída *ExtractionTask*. Instance této třídy, udržuje veškeré informace o načtených ontologických třídách, vybraných vlastnostech, vygenerovaných dvojicích, vypočtených cestách mezi dvojicemi, a také o klíčové vlastnosti pro tvorbu dvojic.

Jak už bylo zmíněno dříve, pro převod tříd do formátu JSON byla použita knihovna GSON. Základní použití této knihovny je velice jednoduché. Pro převod třídy do formátu JSON je potřeba pouze zavolat metodu *toJson* nad patřičnou instancí třídy *Gson*, která vrátí požadovanou reprezentaci jako řetězec.

Ve výchozím nastavení jsou exportovány všechny třídní atributy, krom těch statických. Tento způsob lze ovšem použít pouze pokud se ve třídě, či třídách, které jsou odkazovány, nacházejí atributy s běžnými datovými typy. Obzvláště velký problém pak knihovně působí kruhové reference mezi třídami.

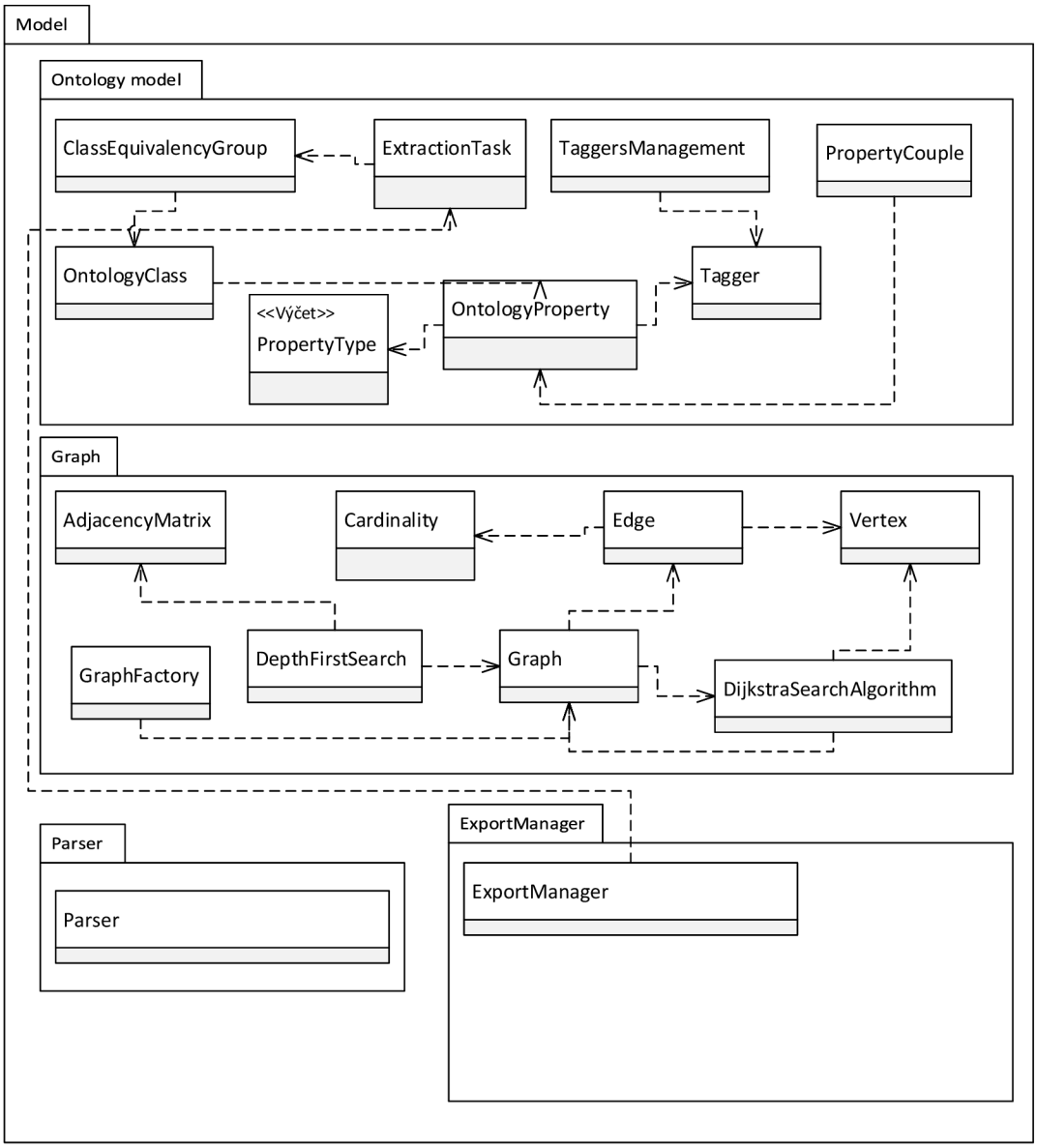
Tento problém se řeší implementací vlastních serializačních a deserializačních adaptérů. Adaptéry definují způsob, jakým je instance převedena na objekt ve formátu JSON a naopak. Například třída **PropertyCouple** obsahuje dva atributy datového typu *IOntologyProperty*. Bez adaptéru je vytvořen json objekt, který obsahuje dva atributy a každý pak jako svou hodnotu další json objekt reprezentující instance typů *IOntologyProperty*. Tato reprezentace ale není žádoucí. Díky adaptéru lze definovat, že místo celého objektu bude exportován pouze jeden jeho atribut. V případě *PropertyCouple* tedy pouze URI. V popsaném případě sice není žádná kruhová závislost, ovšem výběrem pouze jednoho atributu z objektu by došlo k její přerušení.

Jak už bylo řečeno, ve výchozím nastavení jsou exportovány všechny třídní nestatické atributy. Tento způsob ovšem nebyl použit, jelikož třída obsahuje i atributy, které nemají být součásti vyexportovaného zadání. Pro export byl nastaven speciální režim, který bere v úvahu pouze atributy označené speciální anotací *@Expose*.

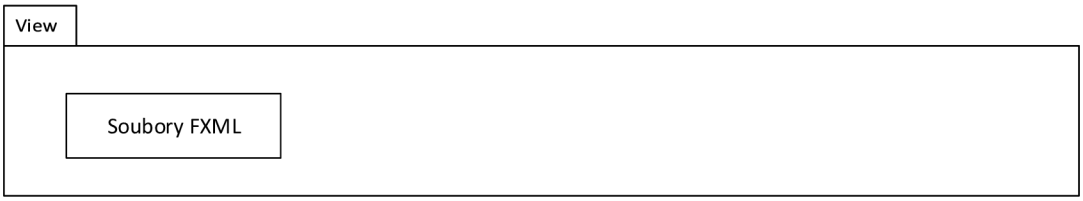
### 5.6.1 Formát zadání extrakční úlohy

```
{
  "keyProperty": {
    "iri": "http://purl.org/dc/elements/1.1/title",
    "domain": "http://fitlayout.github.io/ontology/papers.owl#Paper",
    "range": "http://www.w3.org/2000/01/rdf-schema#Literal",
    "tagger": "null"
  },
  "selectedProperties": [
    {
      "iri": "http://xmlns.com/foaf/0.1/name",
      "domain": "http://fitlayout.github.io/ontology/papers.owl#Person",
      "range": "http://www.w3.org/2000/01/rdf-schema#Literal",
      "tagger": "null"
    },
    {
      "iri": "http://purl.org/dc/elements/1.1/title",
      "domain": "http://fitlayout.github.io/ontology/papers.owl#Paper",
      "range": "http://www.w3.org/2000/01/rdf-schema#Literal",
      "tagger": "null"
    },
    ...
  ],
  "propertyCouples": [
    {
      "first": "http://purl.org/dc/elements/1.1/title",
      "second": "http://xmlns.com/foaf/0.1/name",
      "cardinality": "M:N"
    },
    {
      "first": "http://purl.org/dc/elements/1.1/title",
      "second": "http://purl.org/dc/elements/1.1/title"
    }
        ...
  ]
}
```

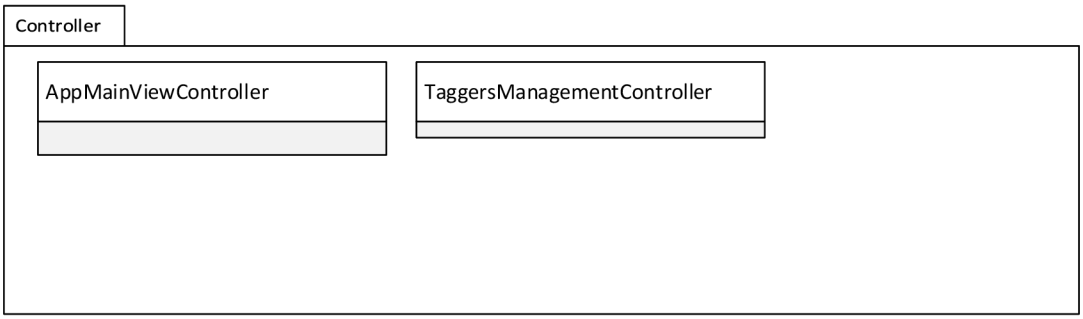
Listing 5.3: Ukázka výsledného formátu zadání extrakční úlohy




Obrázek 5.1: Model domény vrstvy modelu




Obrázek 5.2: Model domény vrstvy view

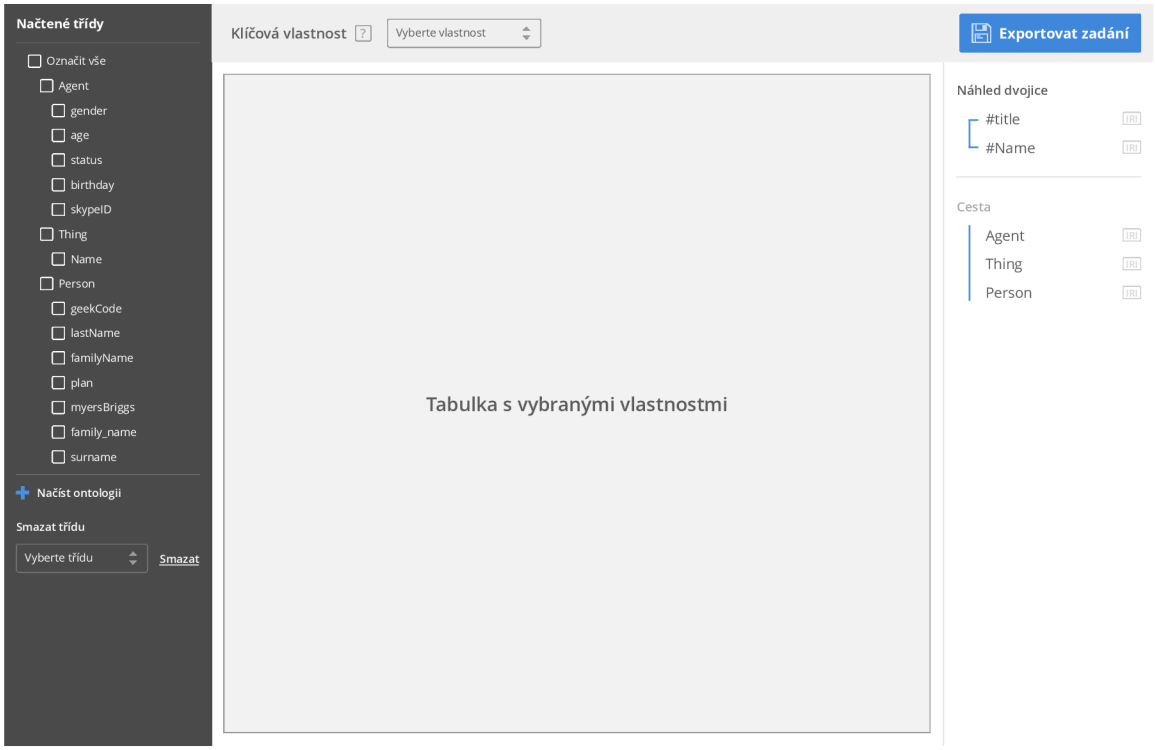



Obrázek 5.3: Model domény vrstvy controlleru

# Kapitola 6

# Grafický návrh uživatelského rozhraní

Při návrhu uživatelského rozhraní (UI) bylo nutné brát zřetel na velké množství informací, které se v něm budou zobrazovat. Velikost načítaných ontologií není omezená a tedy zvolené zobrazení musí umožňovat zobrazení libovolného množství informací. Přitom by mělo být rozhraní stále přehledné a snadno použitelné, bez ohledu na množství zobrazovaných dat.




Obrázek 6.1: Grafický návrh uživatelského rozhraní nástroje.

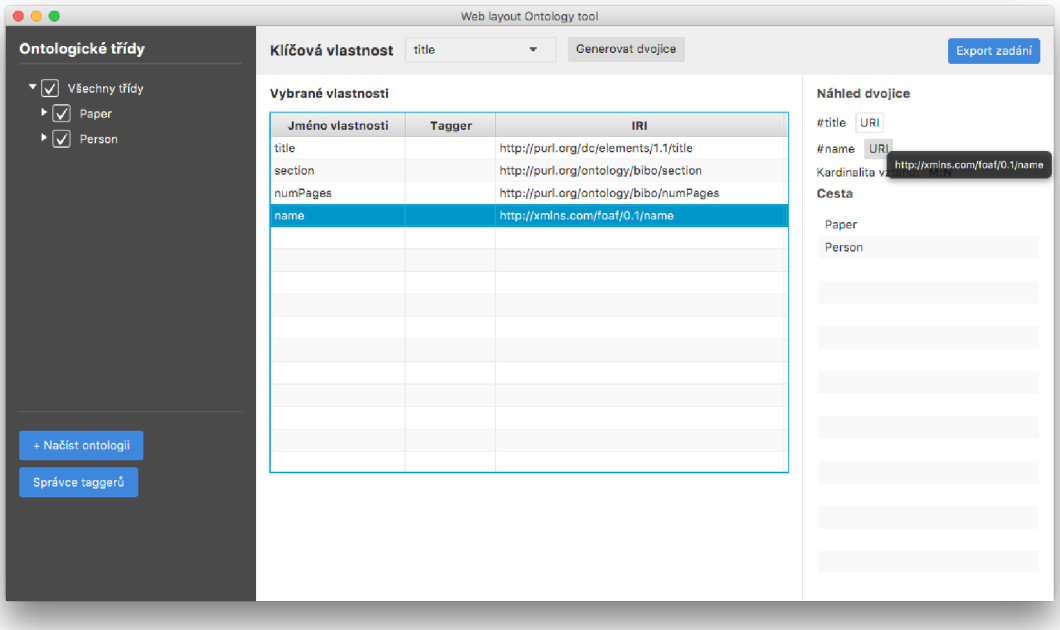
Do nástroje je možné načíst libovolné množství souborů, které obsahují popis ontologií a jejich vlastností. Po načtení ontologií, už není potřeba se souborem dále pracovat, a tudíž není potřeba jej dále v rozhraní jakkoliv reprezentovat. Ontologické třídy a jejich vlastnosti jsou vyobrazeny jako strom se zaškrtávacími políčky. Je tak na první pohled patrný hierar-

chický vztah mezi třídami a vlastnostmi. Jak už je to u tohoto prvku UI běžné, zaškrtnutím rodičovského uzlu se označí i všechny jeho děti.

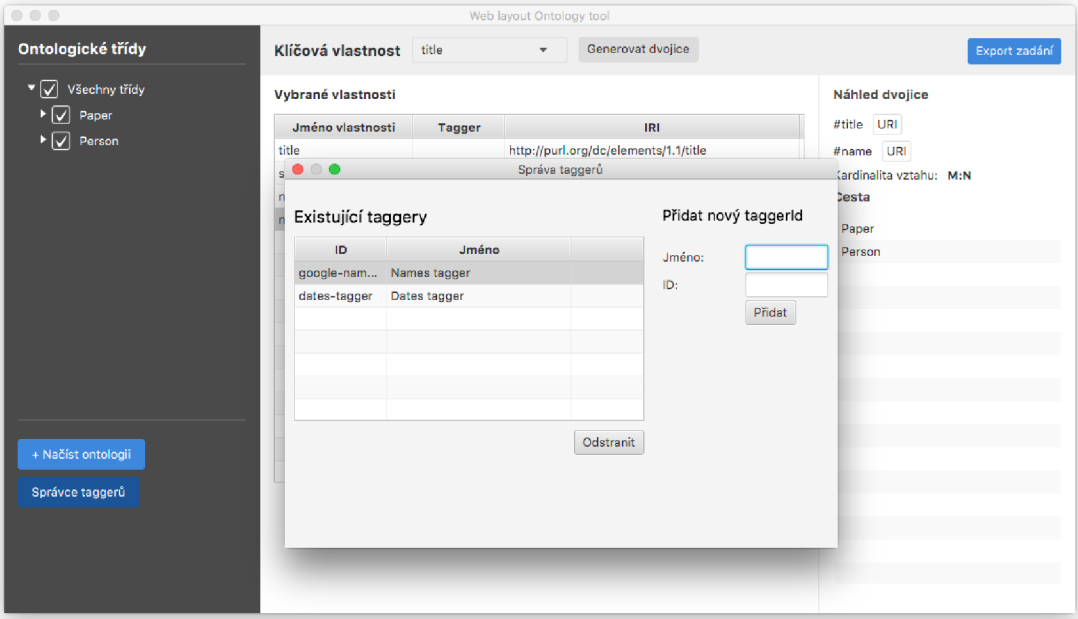
Označením zaškrtávacího políčka dochází k výběru vlastností pro zadání extrakční úlohy. Všechny tyto vybrané vlastnosti jsou pak vyobrazeny v tabulce uprostřed. U každé vlastnosti je zobrazeno její jméno, URI a vybraný tagger. Výběr taggeru je proveden kliknutím do pole a výběrem z již nakonfigurovaných taggerů. Nad tabulkou se nachází panel nástrojů. Zde uživatel vybírá klíčovou vlastnost, která je pak použita při výpočtu dvojic. Klíčovou vlastnost je možné zvolit pouze z vlastností vybraných pro export. Klikem na tlačítko *Generovat dvojice* dojde k vygenerování dvojic a odblokování tlačítka *Export zadání*.

Uživatel si může zkontrolovat správnost vypočtených dvojic označením řádku v tabulce. Dojde k zobrazení informací o vypočtené dvojici, která se k označené vlastnosti vztahuje. Najetím myši na ikonu *URI* je možné zobrazit URI každé vlastnosti a dále je zde zobrazena hodnota vypočtené kardinality vztahu. Níže je pak cesta, která představuje třídy, skrze které vede propojení od první ke druhé vlastnosti.

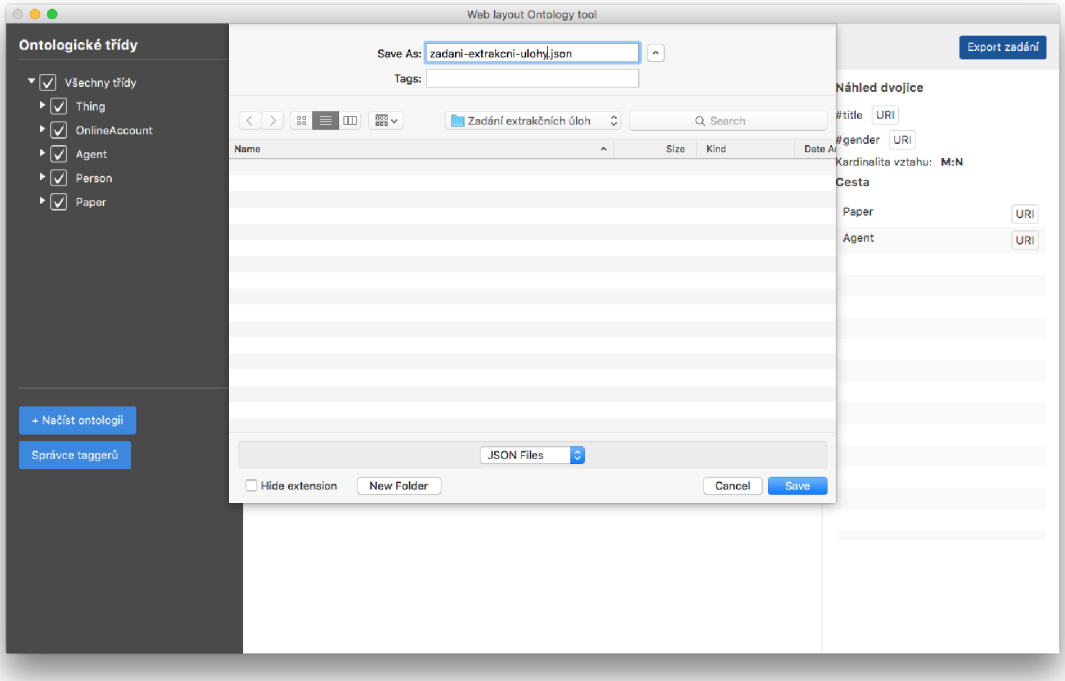
Kliknutím na tlačítko *Export zadání* dojde k zobrazení dialogového okna kde uživatel zvolí název souboru a místo kam se má soubor uložit. V levém panelu se dále ještě nachází tlačítko *Správce taggerů*. Klikem na něj se zobrazí dialogové okno, ve kterém je možné přidávat a odebírat taggery.




Obrázek 6.2: Fotografie uživatelského rozhraní implementovaného nástroje. Výsledné realizované rozhraní není 1:1 shodné s grafickým návrhem. Během implementace rozhraní docházelo k úpravám na základě zkušeností získaných při jeho používání.




Obrázek 6.3: Zobrazení správce taggerů, který umožňuje přidávat a odebírat taggery.




Obrázek 6.4: Dialog pro uložení zadání extrakční úlohy.

# Kapitola 7

# Testování

Pro otestování funkčnosti aplikace byly vytvořeny speciální sady testů, kdy každý sada se zaměřuje na jinou část funkčnosti nástroje. Sady testů byly implementovány pomocí známé testovací knihovny **JUnit** [1] ve verzi 4.12.

## 7.1 Testování výpočtu kardinalit

Pro výpočet kardinality vztahu mezi dvojici vlastností je použita třída *Cardinality*, která implementuje statickou metodu *compute*. Ta požaduje na vstupu jako argumenty dva různé typy kardinalit. Uvnitř funkce je pak definována logika výpočtu po každou kombinaci dvojic typů. Pro ověření správnosti fungování výpočtu je pro každou kombinaci vytvořen jeden test, který ověřuje správnost výsledku pro danou dvojici.

## 7.2 Testování zpracování vstupních souborů

Jedna sada testů se zaměřuje na ověření správného zpracování vstupních souborů. Celá tato sada testů pracuje nad dvěmi předem připravenými soubory. Tyto soubory byly vybrány tak, aby pokryly celou množinu testovaných případů. Ve většině testů je správnost jejich vyhodnocení určena porovnáním s předem známou hodnotou. Tyto hodnoty byly určeny na základě jejich ručního výpočtu.

### 7.2.1 Test načtení použitých souborů

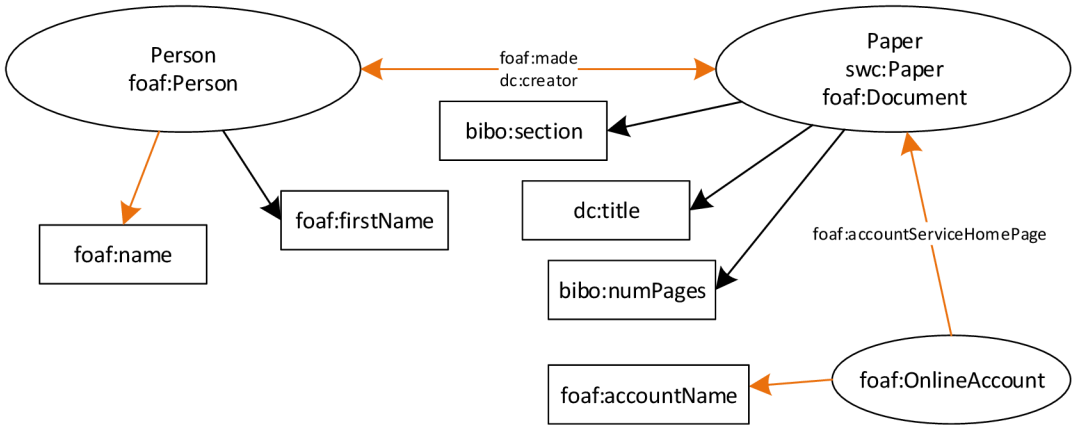
První dva testy se zaměřují na správné načtení a zpracování vstupních souborů. Každý test si načte jeden soubor a provede jeho zpracování pomocí zavolání metody *parse* nad objektem třídy *Parser*. Tato metoda vrací seznam načtených ekvivalenčních skupin. Správnost výpočtu parseru je vyhodnocena pomocí porovnání počtu ekvivalenčních skupin v seznamu a předem známého očekávaného počtu skupin.

### 7.2.2 Test sjednocení ekvivalenčních skupin

Sjednocení ekvivalenčních skupin je dost důležitá část výpočtu pro správnou interpretaci kardinality vztahů dvojic. První soubor ze sady (burget-papers.owl) obsahuje ontologickou třídu *Paper* s datotypovými vlastnostmi *title, numPages, section* a objektovou vlastností

[1] http://junit.org/junit4/

*creator*. Dále má tato třída definované tři ekvivalenční třídy. Další ontologickou třídou v souboru je třída *Person*, která má datotypovou vlastnost *name* a objektovou vlastnost *creator*. Tato třída má dále definovanou jednu ekvivalenční třídu. Konkrétní vztahy jsou vyjádřeny na obrázku níže.




Obrázek 7.1: Zjednodušené grafické znázornění tříd a vazeb mezi nimi. Ovály představují třídy, čtverce datotypové vlastnosti a označené hrany objektové vlastnosti. Oranžová cesta představuje propojení vlastností foaf:name a foaf:accountName přes tři třídy.

Druhý soubor (foaf.rdf) obsahuje celou řadu tříd, ale důležité pro testování jsou především *Person* a *Document*. Jak si lze všimnout, oba soubory obsahují třídu Person a Document. Při načtení prvního souboru je vytvořena ekvivalenční třída pro každou z těchto tříd a při načtení druhého souboru pak již nedochází k vytvoření dalších ekvivalenčních tříd. Při testování je porovnán výsledný počet získaných ekvivalenčních tříd s předem známou hodnotou. Pokud by byl počet tříd větší, znamenalo by to, že nedošlo ke sjednocení ekvivalenčních tříd.

### 7.2.3 Test sjednocení vlastností v rámci ekvivalenční skupiny

Nástroj umožňuje načítáni ontologií a jejich vlastností z vícero souborů. Každý soubor s definicí ontologie může obsahovat libovolný počet ekvivalenčních skupin. Pokud se najde třída taková, že patří do obou skupin (průnik skupin je neprázdný), jsou obě skupiny ekvivalentní a je potřeba je sjednotit. V rámci sjednocení je potřeba také správně sjednotit vlastnosti, pokud již existují instance obou tříd. Tento test se zaměřuje na kontrolu počtu vlastností u třídy, u které došlo ke sjednocení ze dvou souborů. V případě tohoto testu se jedná o sjednocení třídy *Person*, která se vyskytuje v obou souborech.

Při načtení prvního souboru je vytvořena ekvivalenční skupina tříd Person a foaf:Person. Třída foaf:Person neobsahuje žádné datotypové vlastnosti, ale má jednu objektovou. V druhém souboru je naopak definována celá řada datotypových vlastností této třídy. Test tedy kontroluje počet datotypových položek u třídy foaf:Person po sjednocení. Počet musí odpovídat předem známé hodnotě.

## 7.3 Testování výpočtu cesty mezi dvojicemi

Nalezení cesty mezi dvojicemi je stěžejní výpočet pro určení kardinality vztahu dvojice. Nejprve je sestaven graf ze všech načtených tříd a jejich vlastností. Nad tímto grafem pak probíhá výpočet pomocí Dijkstrova algoritmu pro nalezení cesty v grafu. Ontologie a jejich vlastnosti jsou opět načítány ze souboru *burget-papers.owl* a *foaf.rdf*. Sada testů obsahuje test na nalezení cesty v rámci jedné třídy, mezi dvěmi třídami a nakonec mezi třemi třídami. Pro vyhodnocení správnosti výpočtu jsou použita kritéria jako výsledná kardinalita vztahu, počet nalezených tříd na cestě nebo kontrola konkrétních tříd, které se na cestě musí vyskytnout. Správné výsledky pro porovnání byly předem ručně vypočteny.

### 7.3.1 Test výpočtu kardinality mezi datovými vlastnostmi v rámci jedné třídy

Výpočet kardinality vztahu dvojice v rámci jedné třídy je speciální případ, kdy vlastnosti nejsou spojeny pomocí objektové vlastnosti. U datotypových vlastností je předpokládaná kardinalita vztahu vzhledem k jejich třídě vždy 1:1. V rámci toho testu je hledána cesta z vlastnosti *title* do *section* v rámci třídy *Paper*. Tato cesta bude obsahovat přesně dvě hrany a každá z nich bude mít kardinalitu 1:1. Vypočtená kardinalita tedy musí být 1:1.

### 7.3.2 Test výpočtu kardinality vztahu dvou vlastností v rámci dvou tříd

Na rozdíl od průchodu grafem v rámci jedné třídy jsou v tomto případě použity objektové vlastnosti. Ty tvoří vazby mezi třídami a spojují tak jednotlivé třídy v rámci vybudovaného grafu. U hran, které jsou představovány objektovými vlastnostmi je kardinalita určena na 1:N. Toto určení vychází z předpokladu, že objektová vlastnost bude často odkazovat na vícero jedinců dané třídy.

V případě tohoto testu je hledána cesta mezi vlastností *title* třídy *Paper* a *firstName* třídy *foaf:Person*. Tyto dvě třídy jsou spojeny pomocí vztahu *dc:creator* ve směru od *Paper* k *foaf:Person*. Převedením do běžné řeči tento vztah říká, že vědecký článek může mít více autorů, což je logické a v praxi běžné. V tomto testu je kontrolováno jestli je výsledná kardinalita vztahu rovna M:N.

### 7.3.3 Test výpočtu kardinality vztahu dvou vlastností v rámci tří tříd

Tento test kontroluje zdali dojde ke správnému výpočtu cesty mezi dvěmi vlastnostmi napříč třemi třídami. Tato cesta je v obrázku 7.1 vyznačena oranžovou barvou. Při tomto testu je běžné, že výsledná kardinalita vztahu je M:N. Z tohoto důvodu je u tohoto testu kontrolován celkový počet tříd, které byly nalezeny po cestě. Jak lze vidět na obrázku, je hledána cesta mezi vlastnosti *foaf:name* třídy *Person* a *foaf:accountName* třídy *OnlineAccount*. Tyto třídy jsou postupně spojeny pomocí objektových vlastností *foaf:made* a *foaf:accountServiceHomePage*. Správný výpočet nalezne v rámci cesty tři třídy.

Jelikož počet tříd jako takový nemusí odhalit skrytou chybu ve výpočtu, obsahuje sada ještě jeden test, který kontroluje výpočet té stejné cesty. Na rozdíl od předchozího testu jsou ale kontrolovány konkrétní třídy, které se po cestě nachází. V tomto případě to musí být pouze *Person, Paper a OnlineAccount*. Žádná jiná kombinace tříd není přípustná a znamenala by chybu. Správná posloupnost tříd byla předem ručně vypočtena.

## 7.4 Testování správce taggerů

Jedna sada testů byla také vytvořena pro správce taggerů. Testovanými případy jsou načtení prázdného konfiguračního souboru, vytvoření a uložení jednoho taggeru a vytvoření a uložení vícero taggerů. Smyslem těchto testů je otestovat práci se soubory a správnou funkci serializace a deserializace objektů.

### 7.4.1 Načtení prázdného konfiguračního souboru

Konfigurační soubor obsahuje serializované objekty vytvořených taggerů. V tomto případě je testováno načtení prázdného souboru. Pokud takový soubor neexistuje je vytvořen. Pro posouzení správnosti výpočtu je porovnán počet získaných taggerů s nulou.

### 7.4.2 Vytvoření a uložení taggerů

V tomto testu je vytvořen nový konfigurační soubor. Následně je vytvořen nový tagger, který je serializován a uložen do souboru. Posléze je tento soubor znovu načten a je z něj získán objekt taggeru ve formátu json. Pomocí deserializace je opět vytvořen objekt taggeru s vyplněným id a jménem.

V sadě existuje dále ještě jeden test, ve kterém je provedena stejná procedura s tím rozdílem, že je vytvořeno a uloženo více taggerů. Správnost funkce je v obou případech kontrolována porovnáním počtu vrácených instancí taggerů.

## 7.5 Zhodnocení

Vytvořený nástroj úspěšně prošel všemi navrženými testy. Tím bylo ověřeno jeho korektní chování a tvorba validního výstupu.

# Kapitola 8

# Závěr

Na základě nastudované problematiky sémantického webu a článků o extrakci informací z webu, bylo navrženo řešení převodu ontologií do vlastní reprezentace. Ta umožňuje snadnější práci se získanými informacemi a doplnění těch, které nelze z ontologií vyčíst. Na základě tohoto návrhu byl implementován nástroj s grafickým uživatelským rozhraním, který zjednodušuje tvorbu zadání extrakčních úloh. Výstupem nástroje je soubor se zadáním extrakční úlohy ve formátu JSON, který může experimentální knihovna FITLayout snadno zpracovat. Výsledné řešení bylo otestováno na vhodné sadě ontologií.

# Literatura


[1] *Prohledávání do hloubky*. [Online; navštíveno 15.5.2017].
URL https://en.wikipedia.org/wiki/Depth-first_search

[2] *Web Ontology Language*. [Online; navštíveno 10.1.2017].
URL https://en.wikipedia.org/wiki/Web_Ontology_Language

[3] Brickley, D.; Guha, R.: *Resource Description Framework (RDF) Schema Specification 1.0*. Březen 2000, [Online; navštíveno 7.1.2017].
URL https://www.w3.org/TR/2000/CR-rdf-schema-20000327/

[4] Burget, R.: Information Extraction from Web Documents based on Visual and Semantic Relationship Alignment. In *NLP&DBpedia 2016*, Springer International Publishing, 2016, s. 1–12.
URL http://www.fit.vutbr.cz/research/view_pub.php.cs?id=11218

[5] Embley, D.; Campbell, D.; Jiang, Y.; aj.: *A Conceptual-Modeling Approach to Extracting Data from the Web*. 1998, [Online; navštíveno 8.1.2017].
URL http://www.deg.byu.edu/papers/er98.pdf

[6] Lassila, O.; Swick, R. R.: *Resource Description Framework (RDF) Model and Syntax Specification*. Únor 1999, [Online; navštíveno 4.1.2017].
URL https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

[7] McGuinness, D. L.; van Harmelen, F.: *OWL Web Ontology Language Overview*. Únor 2004, [Online; navštíveno 4.1.2017].
URL https://www.w3.org/TR/2004/REC-owl-features-20040210/#s2.2

[8] Motik, B.; Parsia, B.; Patel-Schneider, P. F.: *OWL 2 Web Ontology Language XML Serialization (Second Edition)*. Prosinec 2012, [Online; navštíveno 10.1.2017].
URL https://www.w3.org/TR/owl2-xml-serialization/

# Přílohy

# Seznam příloh

# Příloha A

# Obsah CD

- **/text** Přeložené PDF a text technické zprávy
- **/java** Zdrojové soubory implementovaného nástroje

# Příloha B

# Kompilace a spuštění nástroje

Pro správu projektu je použit nástroj Maven[1]. Díky tomu lze nástroj přeložit a spustit bez větších komplikací. V nastavení projektu jsou uvedeny všechny závislosti (potřebné knihovny), které nástroj sám automaticky stáhne. Při překladu jsou přeloženy všechny třídy a maven může automaticky vytvořit i balíček JAR, který obsahuje celou strukturu projektu. Stažené knihovny jsou umístěny do složky *dependency-jars* aby se zmenšil případný objem balíčku jar.

Vytvořený balíček WebLayoutOntologyTool.jar lze na unixově založených systémech spustit pomocí přiloženého skriptu *Start.sh*. Díky multiplatformnosti jazyka Java, by neměl být problém se spuštěním nástroje na jakékoliv běžné platformě (Windows, Linux, macOS).

[1] https://maven.apache.org