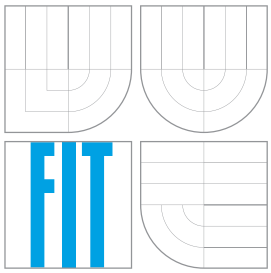
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

METODY EXTRAKCE INFORMACÍ

METHODS OF INFORMATION EXTRACTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ADAM ADAMČEK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN KOUŘIL

BRNO 2015

Abstrakt

Cílem procesu extrakce informací je získání relačních dat z textu psaného přirozeným jazykem na další jednodušší zpracování výpočetní technikou. Oblast využití takto získaných informací je široká – od sumarizace textů, přes vytváření ontologií až po zodpovídání otázek QA systémy. Tato práce popisuje návrh a implementaci systému fungujícího ve výpočetním clusteru, který transformuje výpis článků Wikipedie na množinu vyextrahovaných informací, které jsou následně uloženy do distribuované RDF databáze a je nad nimi možné sestavovat dotazy prostřednictvím vytvořeného uživatelského rozhraní.

Abstract

The goal of information extraction is to retrieve relational data from texts written in natural human language. Applications of such obtained information is wide – from text summarization, through ontology creation up to answering questions by QA systems. This work describes design and implementation of a system working in computer cluster which transforms a dump of Wikipedia articles to a set of extracted information that is stored in distributed RDF database with a possibility to query it using created user interface.

Klíčová slova

extrakce informací, koreference, NLP, QA, Wikipedie, OpenIE, SECAPI, DECIPHER, NER, RDF, SPARQL, 4store, výpočetní cluster

Keywords

information extraction, coreference, NLP, QA, Wikipedia, OpenIE, SECAPI, DECIPHER, NER, RDF, SPARQL, 4store, computer cluster

Citace

Adam Adamček: Metody extrakce informací, diplomová práce, Brno, FIT VUT v Brně, 2015

Metody extrakce informací

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Kouřila. Další informace mi poskytl Doc. Pavel Smrž.

.....
Adam Adamček
26. května 2015

Poděkování

Na tomto místě bych chtěl poděkovat svému vedoucímu Ing. Janu Kouřilovi a Doc. Pavlu Smržovi za jejich čas, cenné připomínky a odbornou pomoc na konzultacích k této diplomové práci. Dále bych rád poděkoval svým rodičům, bratrovi a přátelům za podporu při studiu.

Obsah

Úvod

Problém extrakcie informácií sa dá popísať ako automatická extrakcia štruktúrovanej informácie z neštruktúrovaného alebo pološtruktúrovaného zdroja. Inými slovami, ide o získanie relačných dát z textu písaného prirodzeným jazykom, pretože takto získané dáta sa následne dajú ďalej oveľa ľahšie spracovávať výpočtovou technikou [1]. V kapitole 2 bude popísaná história, formálna definícia a členenie tohoto problému, rovnako ako aj predstavený inovatívny prístup *OpenIE* snažiaci sa vyriešiť problém so spracovávaním rozsiahlych korpusov. V tejto kapitole budú tiež popísané niektoré významnejšie systémy na extrakciu informácií. Ešte pred tým však budú v kapitole 1 uvedené metódy, ktoré predstavujú základ akéhokoľvek automatizovaného spracovania textu.

Kapitola 3 predstavuje návrh systému, ktorý je schopný zo získanej dátovej sady vytvoriť množinu z nej vyextrahovaných faktov a následne tieto dáta dotazovať prostredníctvom používateľského rozhrania za účelom hľadania odpovedí na otázky používateľa tohoto systému. K dosiahnutiu tohoto cieľa je však potrebné vykonanie mnohých krokov – od čistenia, predspracovávania či extrakcie, až po analýzu vyextrahovaných vedomostí a ich ukladanie. Keďže navyše pracujeme s rozsiahlymi dátami, práca sa snaží priniesť riešenie podporujúce výpočet náročných krokov vo výpočtovom klastri.

Na obsah predchádzajúcej kapitoly bude nadväzovať kapitola 4 popisujúca zvolené technológie spolu s odôvodneniami ich výberu, popis implementácie navrhnutého systému a problémy, s ktorými som sa v priebehu vývoja stretol a bolo ich potrebné riešiť.

Vytvorený systém bol otestovaný na dátach anglickej mutácie internetovej encyklopédie *Wikipédia*, pričom výstupy z tohoto testovania v podobe analýzy rýchlosti spracovania jednotlivých jeho komponent, vyhodnotenie úspešnosti ich fungovania a ukážku použitia systému je možné nájsť v kapitole 5.

Možnosť využitia takto vyextrahovaných informácií je široká – od sumarizácie textov, cez vytváranie ontológií až po to, čo je pre bežného používateľa moderných technológií pravdepodobne momentálne najzaujímavejšie a najužitočnejšie, a to zodpovedanie bežných otázok položených v prirodzenom jazyku. Súčasné vyhľadávače hodnotia obsah dokumentov na internete podľa toho, ako pravdepodobne užitočný je ich obsah v súvislosti s kľúčovými slovami, ktoré zadá používateľ do vyhľadávača, a tie najrelevantnejšie hodnotené dokumenty zobrazia medzi výsledkami. Nie sú však schopné spracovať a vyhodnotiť fakty obsiahnuté v týchto textoch, skombinovať vedomosti získané z viacerých dokumentov a pomocou nich čo najkonkrétnejšie zodpovedať otázku používateľa formulovanú ideálne v jeho prirodzenom jazyku. Preto kapitola 6 stavia na dosiahnutých výsledkoch a pojednáva o smere možného napredovania tohoto projektu.

Záverečná kapitola sumarizuje výsledky tejto práce.

Obsah tejto diplomovej práce nadväzuje na výsledky semestrálneho projektu vypracovaného počas zimného semestra, v ktorom bola naštudovaná potrebná teória využívaná pri automatizovaných analýzach textov a spracovaný problém extrakcie informácií. Súčasťou spomínaného projektu bolo aj zoznámenie sa s výpočtovým klastrom a vytvorenie jednoduchého prototypu extrakčného systému. Táto diplomová práca využíva vo svojich prvých dvoch kapitolách už spracované teoretické poznatky a svoj návrh stavia na návrhu spomínaného prototypu extrakčného systému.

Kapitola 1

Automatické spracovanie prirodzeného jazyka

Obsahom tejto kapitoly je uvedenie prehľadu o jednotlivých základných krokoch, ktoré sa bežne využívajú pri automatizovaných lingvistických analýzach textov, a výstupy ktorých často slúžia ako základ pre akékoľvek ďalšie pokročilé spracovanie ich obsahov. Jednotlivé sekcie tejto kapitoly popisujú tieto činnosti v chronologickej nadväznosti, pričom zvyčajne slúži výstup kroku v jednej sekcii ako vstup pre úlohu popisovanú v ďalšej sekcii. Obsah nasledujúcej kapitoly vychádza z informácií uvedených v [2], [3] a [4].

1.1 Korpus a lexikálna databáza

Pod pojmom jazykový **korpus** rozumieme rozsiahlu kolekciu autentických textov uložených v jednotnom formáte a je určený prevažne na účely lingvistickej analýzy, skúmanie rôznych jazykových javov v ich prirodzenom kontexte a využíva sa ako základ pri tvorbe slovníkov. V dnešnej dobe majú korpusy formu digitálnu, čo uľahčuje ich udržiavanie, šírenie a vyhľadávanie v nich [5]. Na digitálne korpusy sa v dnešnej dobe spolieha aj oblasť automatického spracovania prirodzeného jazyka, pretože veľa algoritmov, ktoré sa v tejto oblasti využívajú, funguje na pravdepodobnostnom princípe a obsah korpusu využívajú na natrénovanie svojich interných modelov. Príkladom takéhoto využitia je určovanie slovných druhov popísané v sekcii 1.3 ďalej v tejto kapitole.

Najstarším digitálnym korpusom je ***Brownov korpus*** americkej angličtiny publikovaný na Brownovej univerzite v roku 1964. Obsahuje 500 anglicky písaných textov z pätnástich rôznych žánrov, ktoré dokopy obsahujú zhruba milión slov. V rámci oblasti automatického spracovania prirodzeného jazyka je spomedzi anglických korpusov najpopulárnejší ***Pennov korpus*** z Pensylvánskej univerzity, ktorý obsahuje bežné anglické texty doplnené o syntaktickú anotáciu viet vo forme stromov a taktiež aj označené slovné druhy jednotlivých slov. V Českej republike už od roku 1967 vzniká ***Pražský závislostný korpus***, ktorý obsahuje morfologicky, syntakticky a sémanticky anotované české texty [6].

Lexikálna databáza je digitálnou databázou pozostávajúcou z jedného či viacerých slovníkov, ku ktorej existuje software, ktorý umožňuje pracovať s jej obsahom. Najznámejšou takouto databázou je ***WordNet*** [7], ktorý je vyvíjaný v anglickom jazyku od roku 1985 na Pricetonskej univerzite. Zoskupuje slová do synonymických rád (tzv. *synsety*), zachytáva medzi nimi rôzne sémantické vzťahy, ako napríklad *hyponymické vzťahy* vyjadrujúce podradené slová, *hypernomické vzťahy* pre nadradené slová, *synonymá* (ekvivalentné slová),

antonymá (protikladné slová) a ďalšie. Obsahuje krátke definície týchto slov a na základe princípov tohoto pôvodného pricetonského WordNetu vznikli tiež podobné projekty v pre iné jazyky.

1.2 Segmentácia a tokenizácia

Aj keď človek zvyčajne dokáže v textoch, ktoré sú predmetom pokusov o automatizované spracovanie, často ľahko a na prvý pohľad rozpoznať určité špecifické vzory a tieto texty dekomponovať na menšie jednotky aj bez bližšej znalosti obsahu, z pohľadu stroja, ktorý sa takýto text snaží analyzovať, sa rovnaký vstup javí iba ako bližšie nešpecifikovaná postupnosť znakov, resp. hodnôt. Prvým krokom, ktorému je preto potrebné čeliť, je rozdelenie takéhoto súvislého, často bližšie neštruktúrovaného vstupného bloku textu na menšie jednotky, ktoré je možné následne pomocou ďalších prístupov bližšie analyzovať.

Úlohou **segmentácie (sentence splitting/segmenting)** je rozdelenie textu na jednotlivé vety. Pretože interpunkčné znamienka, ktorými bývajú vety ukončované v písaných formách západných jazykov, majú veľmi dlhú históriu používania, ich úloha v texte môže mať oveľa viac účelov – od značenia dátumov, skratiek, či napríklad aj v rôznych citátoch. V dnešnej dobe sa interpunkcia využíva tiež vo webových adresách, ale kľudne môže byť obsiahnutá aj v názvoch diel, ktoré sa v texte spomínajú. Pri jazykoch nevyužívajúcich latinskú abecedu či cyriliku môžu byť problémy segmentácie ešte komplikovanejšie.

Tokenizér zasa rozdeľuje text na významové jednotky, ktoré sa nazývajú *tokeny*. Token predstavuje postupnosť znakov, ktorá je ďalej nedeliteľná a je nositeľom informačnej hodnoty. Najjednoduchšia forma tokenizácie je rozdelenie podľa bielych znakov a interpunkcie, avšak to v mnohých jazykoch, najčastejšie exotických ázijských, nemusí byť vôbec dostačujúce alebo možné. Taktiež západnému svetu blízke jazyky ako angličtina alebo nemčina často obsahujú v textoch zložené slová, ktoré je niekedy potrebné pre ďalšie spracovanie správne rozdeliť, inokedy naopak identifikovať viacero po sebe idúcich slov ako jednu významovú jednotku.

1.3 Určovanie slovných druhov

Na **určovanie slovných druhov (Part of Speech/PoS tagging)** existujú systémy, ktoré prijímajú tokenizovaný text ako vstup a ku každému tokenu pripoja označenie vyjadrujúce jeho slovný druh. Algoritmy, vďaka ktorým je táto funkčnosť zabezpečená, môžu byť rôzne, aj keď sa zvyčajne využíva pravdepodobnostný prístup vychádzajúci z teórie skrytých Markovských modelov (*Hidden Markov models, HMM*). V závislosti na tom, na akej množine vstupných dát bol nakoniec pravdepodobnostný model jednotlivých realizácií takýchto systémov natrénovaný, sa odvíjajú aj špecifické množiny označení slovných druhov, tzv. *tagset*, ktoré dokážu tieto systémy jednotlivým tokenom priradiť.

Najpoužívanejšou množinou označení slovných druhov je v súvislosti s textami písanými v americkej angličtine sada vychádzajúca z označení použitých v *Pennovom korpuse* obsahujúca 36 PoS tagov a ďalších 12 označení vyčlenených pre interpunkciu či ďalšie špeciálne symboly a je uvedená v tabuľke 1.1. Z veľkej časti je táto sada tiež podobná označeniam použitým v *Brownovom korpuse* a niektorým ďalším skôr vydaným lingvistickým korpusom, avšak, keďže svojím rozsahom tieto korpusy prevyšuje, je na túto úlohu Pennov korpus uprednostňovaný.

Rozlišovanie slovných druhov prináša problémy najmä v súvislosti s *homoformou*, čo je

#	Zn.	Popis
1.	CC	Priraďovacia spojka
2.	CD	Kardinálne číslo
3.	DT	Determinant
4.	EX	Existenciálna varianta slova "there"
5.	FW	Zahraničné slovo
6.	IN	Predložka alebo podraďujúca spojka
7.	JJ	Prídavné meno
8.	JJR	Komparatívne prídavné meno
9.	JJS	Superlatívne prídavné meno
10.	LS	Symbol prvka zoznamu
11.	MD	Modálne sloveso
12.	NN	Pods. m. v j. č. alebo nepočítateľné
13.	NNS	Pods. m. v mn. č.
14.	NNP	Vlastné pods. m. v jednotnom čísle
15.	NNPS	Vlastné pods. m. v množnom čísle
16.	PDT	Vymedzovacie zámeno
17.	POS	Privlastňovací suffix "'s"
18.	PRP	Osobné zámeno
19.	PRP$	Privlastňovacie zámeno
20.	RB	Príslovka
21.	RBR	Komparatívna príslovka

#	Zn.	Popis
22.	RBS	Superlatívna príslovka
23.	RP	Častica
24.	SYM	Symbol
25.	TO	Predložka "to"
26.	UH	Citoslovce
27.	VB	Sloveso v základnom tvare
28.	VBD	Sloveso v minulom čase
29.	VBG	Sl. v prieb. tvare alebo príčastí prít.
30.	VBN	Sloveso v príčastí minulom
31.	VBP	Sloveso v prít. č. inej ako 3. os. j. č.
32.	VBZ	Sloveso v prít. č. 3. os. j. č.
33.	WDT	Determinant začínajúci na "wh-"
34.	WP	Zámeno začínajúce na "wh-"
35.	WP$	Privl. zámeno začínajúce na "wh-"
36.	WRB	Príslovka začínajúca na "wh-"
37.	#	Znak mriežky
38.	$	Znak dolára
39.	.	Bodka ukončujúca vetu
40.	,	Čiarka
41.	:	Dvojbodka
	()’‘"_-	Ďalšie symboly

Tabuľka 1.1: Sada označení slovných druhov použitých v *Pennovom korpuse*. [8]

jav, pri ktorom sa slová podobajú zvukovo aj písomne, ale v skutočnosti majú rozličný pôvod a význam. Najväčšou výzvou je však ich schopnosť vyrovnať sa s neznámymi tokenmi.

1.4 Morfologická analýza, zjednoznačňovanie významu, lemmatizácia a stemming

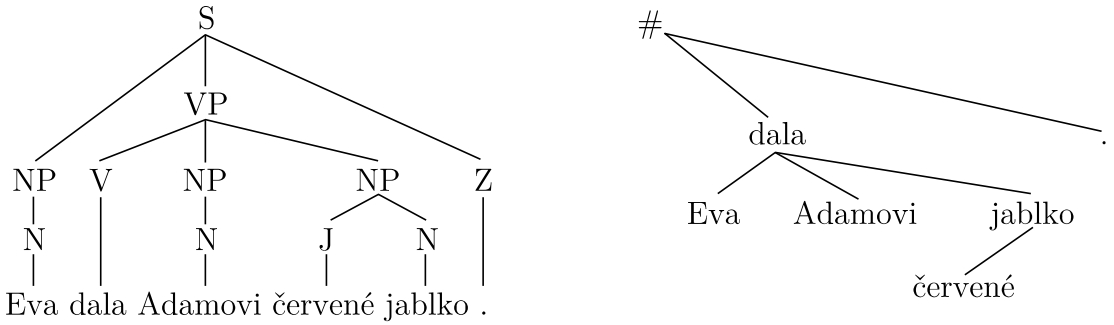
Lemma je základný tvar slova vznikajúci transformáciou z pôvodného tvaru vyskytujúceho sa v texte pri procese nazývanom **lemmatizácia**. Pri tomto procese sa z pôvodnej formy slova odstránia niektoré jeho vlastnosti, ktoré vznikli jeho ohýbaním – aplikovaním časovania a skloňovania. Úlohou **morfologickej analýzy** v lingvistike je tieto vlastnosti identifikovať a pripojiť ich k danému tokenu vo forme značiek gramatických kategórií spolu so slovným vzorom a lemmou. *Gramatické značenie* má väčšinou tvar reťazca, kde jeho jednotlivé znaky označujú napr. rod, pád, čas, číslo a podobne. Niektoré analyzátory tiež poskytujú informácie o *derivačnej morfológii* tokenu a sú schopné poskytnúť aj informácie o slovách, z ktorých bolo zložené.

Ďalším procesom je tzv. **stemming**, ktorý, na rozdiel od vopred menovaného, pomocou jednoduchých pravidiel realizovaných najčastejšie regulárnymi výrazmi odstráni príponu a vráti iba kmeň slova.

Výstup z morfologického analyzátora však často nebýva jednoznačný, a tak je potrebné niekedy navrhované viaceré možnosti dodatočne zjednoznačniť procesom **zjednoznačňovania významu (dezambiguácie)**. Po vykonaní tohoto kroku je k dispozícii *anotovaný korpus*.

1.5 Syntaktická analýza

Syntaktická analýza sa zaoberá vetným rozborom, tzn. určovaním vzťahov medzi slovami alebo časťami viet. Pravidlá, ktoré určujú spôsob, akým sú slová a frázy prepojené a aký to má dopad na poradie vetných častí, sa nazývajú súhrnne ako *gramatika*. Existuje mnoho teórií ohľadom vetnej syntaxe deliacich ju vo všeobecnosti na 2 hlavné kategórie, ktoré je možné vyjadriť jednou z nasledujúcich dvoch štruktúr:



Zložkový strom **Závislostný strom**

Obr. 1.1: Zložková a závislostná syntaktická analýza vety „Eva dala Adamovi červené jablko.".

- **závislostný strom (*dependency tree*)** – štruktúra, v ktorej sú slová uzlami vety, medzi ktorými existujú závislosti vyjadrené orientovanými hranami, pričom koreňom tohoto stromu je práve jeden prísudok
- **zložkový (frázový) strom (*constituency tree*)** – zodpovedá derivačnému stromu bezkontextovej gramatiky a veta sa rekurzívne delí do častí, kde tokeny predstavujú listy stromu

Úlohou *syntaktického analyzátora (parser)* je určiť tieto závislosti. Porovnanie analýzy rovnakej vety zložkovým aj závislostným prístupom je znázornené na obrázku 1.1.

1.6 Vyhodnocovanie kvality

Na posúdenie výkonnosti jednotlivých systémov vykonávajúcich automatické spracovanie prirodzeného jazyka sa využívajú niekoľké miery, pri ktorých sa predpokladá, že vieme ohodnotiť každú časť výstupu tohoto systému, ktorou môže byť napríklad token alebo celá veta, ako *správnu* alebo *nesprávnu*.

Pre jednoduchšie zadefinovanie týchto metrík bude slúžiť tzv. *kontingenčná tabuľka (confusion matrix)* 1.2, do ktorej je možné výsledky systému umiestniť na základe toho ako korešpondujú s predpokladanými výsledkami. Význam jej obsahu je nasledujúci:

- **skutočne pozitívne (TP)** – výsledky, ktoré systém zaradil podľa našich očakávaní ako správne a využije ich na ďalšie spracovanie

- **falošne pozitívne (FP)** – výsledky, ktoré systém chybne zaradil ako správne a aj keď sú nerelevantné ich využije na ďalšie spracovanie, tzv. *Chyba I. typu*
- **skutočne negatívne (TN)** – výsledky, ktoré systém zaradil podľa našich očakávaní ako nesprávne a nebudú použité k ďalšiemu spracovaniu
- **falošne negatívne (FN)** – výsledky, ktoré systém chybne zaradil ako nesprávne a aj keď sú relevantné nebudú použité k ďalšiemu spracovaniu, tzv. *Chyba II. typu*
- ***senzitivita*** = **TP / (TP + FN)** – miera skutočne pozitívnych výsledkov, zodpovedá úspešnosti detekcie
- ***špecificita*** = **TN / (FP + TN)** – miera skutočne negatívnych výsledkov

		Predpoklad	
		Relevantné	**Nerelevantné**
Systém	**Získané**	skutočne pozitívne	falošne pozitívne
	Nezískané	falošne negatívne	skutočne negatívne
		senzitivita	*špecificita*

Tabuľka 1.2: Kontingenčná tabuľka zobrazujúca vzťahy medzi výsledkami určenými systémom a predpokladanými výsledkami.

Ďalej je možné zadefinovať tieto miery vyplývajúce z hodnôt zadefinovaných vo vyššie uvedenej kontingenčnej tabuľke:

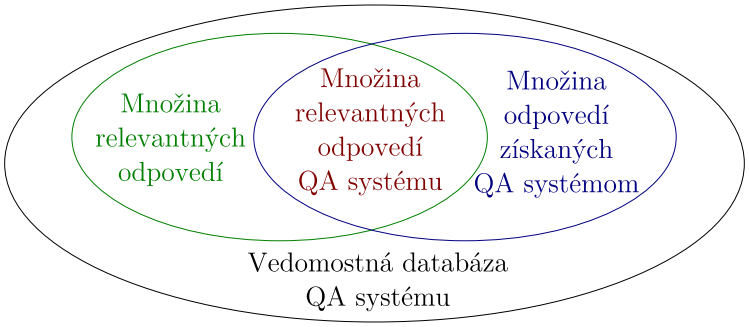
- ***presnosť*** = **TP / (TP + FP)** – miera anglicky nazvaná ako *precision*, ktorá je definovaná ako podiel správne objavených výsledkov a všetkých zistených výsledkov systémom, nadobúda reálne hodnoty v rozmedzí $< 0,1 >$, kde $1,0$ znamená bezchybne fungujúci systém
- ***pokrytie*** = **TP / (TP + FN)** – miera anglicky nazvaná ako *recall*, ktorá je definovaná ako podiel správne objavených výsledkov a všetkých očakávaných výsledkov, nadobúda reálne hodnoty v rozmedzí $< 0,1 >$, kde $1,0$ znamená úplne pokrytie systémom

1.6.1 Presnosť a pokrytie QA systému

Zodpovedanie otázok (**QA, *Question Answering***) je počítačová vedná disciplína v rámci oblasti spracovania prirodzeného jazyka a získavania informácií, ktorá sa zaoberá zodpovedaním otázok položených v prirodzenom jazyku.

QA systém je implementácia, ktorá umožňuje používateľovi prostredníctvom vytvoreného rozhrania zadávať otázky, na ktoré potom tento systém hľadá odpovede v tzv. *vedomostnej databáze*. V tejto databáze sú uložené komplexne štruktúrované informácie, za pomoci ktorých je možné zostaviť používateľovi očakávanú odpoveď. Na to je však potrebné nielen správne transformovanie zadanej otázky do reprezentácie najčastejšie sady správne sformulovaných dotazov určených pre túto databázu, ale najmä identifikácia relevantných

výsledkov, ktoré je nakoniec potrebné transformovať opäť do výslednej zmysluplnej formy odpovede pochopiteľnej pre zadávateľa otázky.



Obr. 1.2: Schematické znázornenie relevantných a QA systémom získaných informácií vo vedomostnej databáze.

Na vyhodnotenie kvality takéhoto QA systému je možné využiť veličiny zadefinované na začiatku sekcie 1.6. Na to je však potrebné správne pochopenie hodnôt vystupujúcich v kontingenčnej tabuľke. Ako už bolo uvedené, po položení otázky QA systému je na jej zodpovedanie potrebné prehľadanie celej databázy a správnosť tejto odpovede závisí od množstva vyhľadaných relevantných informácií. Ako ilustruje obrázok 1.2, vedomostná databáza predstavuje celú dostupnú množinu, z ktorej je možné čerpať výsledky, pričom vyhodnocovateľ kvality takéhoto systému musí mať znalosti o jej obsahu a vedieť určiť podmnožinu správnych informácií – na obrázku znázornených zelenou farbou. Systém pri zodpovedaní otázky tiež určí podmnožinu, označenú modro, ktorú považuje za relevantnú, a v závislosti od prieniku týchto dvoch podmnožín je možné naplniť kontingenčnú tabuľku a vypočítať vyššie uvedené veličiny. Najpopisnejšie na určenie kvality sa z týchto hodnôt sa naskytajú práve **presnosť systému** určujúca koľko skutočne relevantných informácií bral tento systém do úvahy pri vytváraní odpovede a **pokrytie systému**, ktoré informuje o množstve relevantných informácií braných do úvahy z celkového dostupného počtu relevantných informácií obsiahnutých vo svojej vedomostnej databáze. Je však zrejmé, že v závislosti od postupu vykonaného testu môže ísť o hodnotenie subjektívne.

Kapitola 2

Extrakcia informácií

Táto kapitola predstavuje problém extrakcie informácií z hľadiska historického, uvádza jeho formálnu definíciu a členenie viaceré podproblémy. Je v nej tiež vysvetlený inovatívny prístup *OpenIE* snažiaci sa vyriešiť problém so spracovávaním rozsiahlych korpusov a popísané niektoré významnejšie systémy na extrakciu informácií.

2.1 História

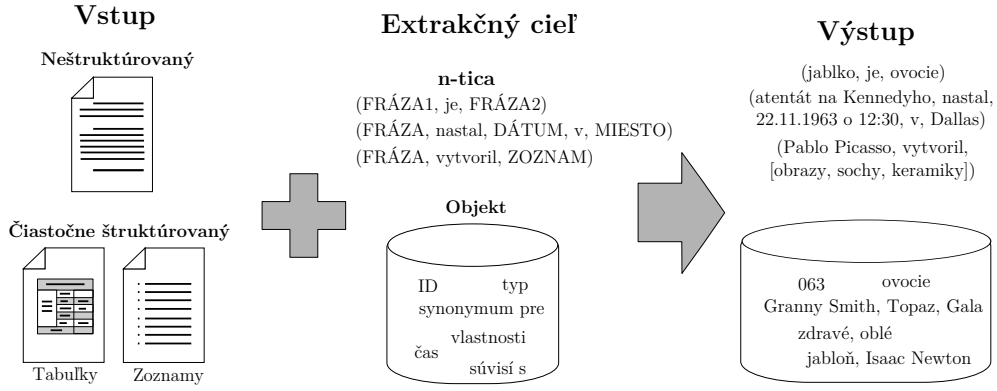
Začiatky extrakcie informácií sa viažu ku komunite zaoberajúcej sa spracovaním prirodzeného jazyka. V jej spoločenstve vznikli dve súťaže, ktoré zastrešili snahy a vymedzujú rámec záujmu v tejto oblasti. Prvá s názvom ***Message Understanding Conference (MUC)***, bola iniciovaná vládnou agentúrou amerického ministerstva obrany DARPA a v jej siedmych kolách prebiehajúcich v rokoch 1987 až 1997 proti sebe súťažili výskumné tímy v úspešnosti pri automatizovanej analýze zadaných textov. Cieľom bolo zvyčajne identifikovať príčiny, činiteľov, dôsledky či časy a miesta udalostí, ktoré z týchto textov týkajúcich sa vládnych záujmov vyplývali [9].

Jej následníkom je program ***Automatic Content Extraction (ACE)***, ktorý vznikol v roku 1999. Motivácia a ciele ACE sú rovnaké ako pri MUC, ale na rozdiel od predchádzajúceho programu sa nezameriava len na identifikovanie správnych slov obsiahnutých v texte, ale identifikovanie správnych entít, vzťahov medzi týmito entitami a udalostí v abstraktnejšom zmysle slova. Taktiež rozširuje pole pôsobnosti z čistého textového vstupu na extrakciu informácií z obrázkov a zvukových záznamov [10].

V súčasnosti výzva extrakcie informácií zasahuje do oblastí strojového učenia, získavania informácií, databáz, webu či analýzy dokumentov a vzájomne ich premosťuje.

2.2 Extrakčná úloha

Ako schematicky znázorňuje obrázok 2.1, **extrakčná úloha** je formálne definovaná jej **vstupom**, ktorý môže byť neštruktúrovaný alebo čiastočne štruktúrovaný, a jej extrakčným cieľom. Tvar neštruktúrovaného textu má väčšina dokumentov na internete, ktoré sú písané prirodzeným jazykom, a príkladom pološtruktúrovaného textu sú zasa napríklad zoznamy. **Extrakčným cieľom** takejto úlohy môže byť získanie n-tice atribútov alebo zložitejší, hierarchicky organizovaný objekt [11].



Obr. 2.1: Schematické znázornenie extrakčnej úlohy.

2.2.1 Výzvy pri realizácii

Pri realizácii systému, ktorý je schopný plniť vyššie definovanú úlohu, je však nutné čeliť mnohým výzvam. Jedna z prvých, ktorá sa prirodzene dostáva do pozornosti, je otázka **presnosti**. Pri rozsiahlom korpuse, s ktorým je zvyčajne nutné pracovať, nie je ľahké detegovať chybné alebo vynechané extrakcie. Tiež je otázne, ako dostatočne je schopná štruktúra definovaná extrakčným cieľom pokryť informácie obsiahnuté v texte, z ktorého extrahujeme. S veľkosťou vstupných dát nutne súvisí aj otázka **výkonu**. Ak systém nasadíme na menej rozsiahle dáta týkajúce sa špecifickej oblasti, potom, samozrejme, algoritmus prinášajúci veľmi presné výsledky za cenu pomalšieho spracovania neprekáža. Avšak, nasadenie obdobného algoritmu na korpus pozostávajúci z miliárd dokumentov nachádzajúcich sa na webe, pričom ich počet každým dňom nezastaviteľne rastie, neprichádza v žiadnom prípade do úvahy. Jedným z mnohých ďalších problémov je napríklad aj **integrácia dát** z viacerých zdrojov a v rôznorodých vstupných formátoch [12].

2.2.2 Extrakčné podúlohy

Existuje päť hlavných podúloh, ktoré zahŕňa oblasť extrakcie informácií [13]:

- **rozpoznávanie pomenovaných entít (*Named Entity recognition, NE*)** – vyhľadávanie a klasifikácia entít ako sú napr. mená, miesta, názvy spoločností, názvy diel a podobne
- **koreferencia (*Coreference resolution, CO*)** – identifikácia a rezolúcia odkazov medzi entitami, najčastejšie v súvislosti so zámenami, ktoré anaforicky odkazujú na iné pomenované entity obsiahnuté v texte
- **konštrukcia vzorových prvkov (*Template Element construction, TE*)** – pridanie ďalších charakteristických popisných informácií k pomenovaným entitám, využíva tiež koreferenciu
- **konštrukcia vzorových relácií (*Template Relation construction, TR*)** – určenie vzťahov medzi vytvorenými vzorovými prvkami
- **produkcia vzorových scenárov (*Scenario Template production, ST*)** – určenie udalostí, v ktorých sú identifikované vzorové prvky a vystupujúce relácie

Uvažujme napríklad nasledujúci text:

> Dnes ráno svet obletela správa o nájdení malej smaragdovej planéty v našej Slnečnej sústave. Jej objaviteľ, prof. Novák, pracuje pre NASA.

V takomto texte rozpoznávač pomenovaných entít určí entity *planéta, Slnečná sústava, prof. Novák* a *NASA*. Koreferencia zistí, že *dnes* odkazuje na dátum publikovania článku a *jej* zasa na planétu. Konštrukcia vzorových prvkov následne určí, že planéta je *malá* a *smaragdová* a konštrukcia vzorových relácií, že sa planéta nachádza v *Slnečnej sústave* a prof. Novák pracuje pre *NASA*. Produkcia vzorových scenárov zaznamená udalosť týkajúcu sa objavenia planéty, v ktorej vystupujú viaceré entity.

2.2.3 Metódy extrakcie

Väčšina prístupov k riešeniu tohoto problému sa zameriava na využitie prístupu učenia s učiteľom na sadách relácií špecificky vytvorených pre danú oblasť záujmu. Algoritmus natrénovaný na takejto sade je potom síce schopný dosiahnuť vysokú presnosť a odozvu na dokumentoch s podobným obsahom, avšak za cenu nízkej dostupnosti trénovacích dát a faktu, že takéto riešenie nie je možné rozšíriť na použitie na celom internete, ktorého obsah je značne rozmanitý. Techniky, ktoré sa na tento prístup využívajú zahŕňajú napríklad použitie skrytých Markovských modelov, podmienených náhodných polí (*Conditional random fields, CRM*) alebo pevne definovaných inferenčných pravidiel.

2.3 Open Information Extraction

Open Information Extraction (OpenIE, OIE) je nový extrakčný prístup, ktorý bol predstavený v roku 2007 kolektívom z Washingtonskej Univerzity s cieľom umožniť doménovo nezávislé extrahovanie relácií z webového korpusu a stavia na dvoch základných postulátoch. Prvým je **jediný priechod cez vstupné dáta** a druhým nevyžadovať nutnosť **žiadneho ľudského zásahu** do extrakčného procesu. Jediným vstupom do *OpenIE* systému je korpus a jeho výstupom množina vyextrahovaných relácií [14]. Dodržanie prvého bodu dáva predpoklad na to, aby bol výsledný systém škálovateľný na web a druhý bod zasa, aby sa bol takýto systém schopný vysporiadať s heterogenitou jeho obsahu.

2.4 Známe systémy

V priebehu posledných rokov bolo vytvorených niekoľko systémov pokúšajúcich sa priblížiť problém extrakcie informácií k dostatočnému automatizovaniu na to, aby bolo riešenie škálovateľné aj na web a nielen na malú, doménovo špecifickú oblasť záujmu. V nasledujúcej časti je uvedený prehľad niekoľkých významnejších systémov v tejto oblasti.

2.4.1 KnowItAll a KnowItNow

KnowItAll je autonómny extrakčný systém, ktorý vychádza z rozšíriteľnej ontológie a malého množstva ručne predpripravených generických vzorových pravidiel, a tak nespĺňa požiadavky *OpenIE*. Je však jedným z prvých univerzálnejších systémov na extrakciu faktov, konceptov a vzťahov z webu, ktorého výkon a úspešnosť extrahovania slúži ako štandard pri porovnávaní s novými systémami v mnohých ďalších štúdiách. Je založený na jazykovo

a doménovo nezávislej architektúre, ktorá pomocou piatich modulov napĺňa svoju ontológiu faktami a reláciami [15].

Vychádza z metafory *informačného potravinového reťazca*, podľa ktorej sú vyhľadávače „bylinožravce" pasúce sa na webe a inteligentní agenti sú informačné „mäsožravce" konzumujúce ich výstup [16]. *KnowItAll*, považujúci sa v tomto smere za „mäsožravca", využíva vyhľadávače na výpočet štatistík ohľadom webového korpusu a na overenie správnosti vyextrahovaných relácií.

Jeho nasledovník ***KnowItNow*** eliminuje potrebu dotazovania sa ku komerčným webovým prehliadačom predstavením vlastného vyhľadávacieho riešenia označovaného ako *Bindings Engine*, a tak niekoľkonásobne zrýchľuje svoju prevádzku, keďže nie je nútený dodržiavať obmedzujúce kvóty na zasielanie požiadaviek [17].

2.4.2 TextRunner

TextRunner vytvorený na Washingtonskej univerzite je pionierom v oblasti *Open Information Extraction* spĺňajúcim všetky jej požiadavky. Skladá sa z 3 kľúčových modulov [18]:

- **samodohliadajúci učiaci sa mechanizmus** pooznačuje trénovacie dáta vytvorené automaticky z malej vzorky korpusu a využije ich na natrénovanie modelu pomocou Bayesovského naivného klasifikátora
- **generátor kandidátnych extrakcií** jediným priechodom cez vstupné dáta pravdepodobnostne pooznačuje vstupný text slovnými druhmi a využije natrénovaný model na odstránenie nepodstatných častí viet a vyextrahovanie n-árnych relácií
- **klasifikátor extrakcií** vyhodnotí, s akou pravdepodobnosťou je každá vyextrahovaná n-tica predstaviteľom relácie, v ktorej vystupuje

2.4.3 ReVerb

V roku 2011 bol publikovaný ďalší systém od tvorcov *TextRunneru* s názvom ***ReVerb***. Na rozdiel od predchodcov v tejto oblasti dosahuje dvojnásobnú oblasť pod krivkou presnosti a pokrytia, pričom vyše 30% extrakcií má 80% presnosť alebo vyššiu. Pomocou syntaktického obmedzenia 2.1 výrazne znižuje počet neinformatívnych extrakcií a pomocou ďalšieho lexikálneho obmedzenia odlišuje detegované platné relácie od takých, ktoré sú príliš konkrétne na to, aby mali ďalšie využitie [19].

$$V \mid V\ P \mid V\ W \star P$$

kde:

```
V = sloveso častica? príslovka?
W = (pods. meno | príd. meno | príslovka | zámeno | člen)
P = (predložka | častica | inf. značka)
```

Kód 2.1: Regulárny výraz vyjadrujúci syntaktické obmedzenie v extrakčnom systéme *ReVerb* [19].

2.4.4 Ďalšie systémy

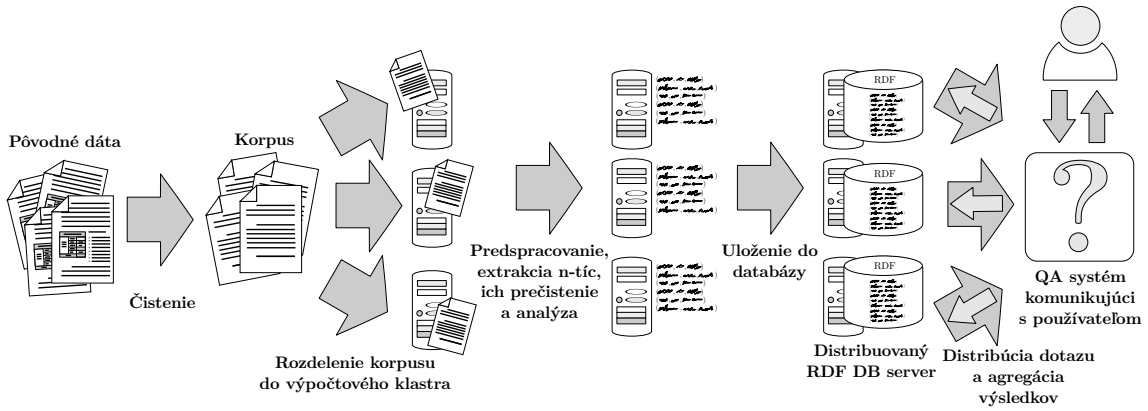
K ďalšiemu zdokonaľovaniu procesu extrakcie prispel tím z Washingtonskej univerzity predstavením systému ***OLLIE***, ktorý dosahuje 2,7-násobne väčšiu plochu pod krivkou presnosti a pokrytia ako predtým vyvinutý systém *TextRunner*. Na dosiahnutie tohoto výsledku využíva na rozdiel od regulárnych výrazov učiaci sa mechanizmus na identifikáciu relácií a analýzu kontextu [20].

Následník systému OLLIE s názvom ***SRLIE*** buduje extrakcie zo SRL rámcov, pri čom využíva *sémantické značenie rolí (SRL)* [21].

Kapitola 3

Návrh systému na extrakciu vzťahov

Nasledujúca kapitola sa venuje návrhu systému, ktorého úlohou je získať zo zvolenej vstupnej dátovej sady v nej obsiahnuté vedomosti, tie vo vhodnej forme uložiť do relačnej databázy. Následne bude prostredníctvom navrhnutého používateľského rozhrania možné k týmto dátam pristupovať a pomocou jednoduchých filtrov simulovať kladenie otázok. Obrázok 3.1 schematicky znázorňuje jednotlivé kroky, ktoré bude potrebné dodržať pri vytváraní takéhoto systému. Tieto kroky sú bližšie popísané v nasledujúcich sekciách tejto kapitoly.



Obr. 3.1: Návrh systému na vytvorenie vedomostnej databázy zo vstupných dát a rozhrania umožňujúceho používateľovi kladenie dotazov.

3.1 Korpus

Navrhovaný systém, ktorý by v ďalších verziách mohol byť schopný pracovať aj s textami s oveľa vyššou mierou nejednoznačnosti a väčším množstvom použitého žargónu, aký sa vyskytuje bežne na komunitných fórach na internete, sa musí vedieť vysporiadať v jeho najzákladnejšej verzii s obsahom písaným v gramaticky a pravopisne spisovnom jazyku.

Ako vstupné dáta na ďalšie spracovanie bol pre túto diplomovú prácu zvolený textový výpis z anglickej mutácie internetového projektu *Wikipédia – slobodná encyklopédia* obsahujúci takmer 5 miliónov článkov a je dostupný z [22]. Dôvodov pre túto voľbu je niekoľko.

V prvom rade sú tieto dáta pre každého ľahko a voľne dostupné, ale čo je podstatnejšie, sú písané v už spomínanom spisovnom jazyku.

Obsah v anglickom jazyku bol zvolený zasa preto, že ide o jazyk, ktorý je predmetom výskumov na svetovej úrovni, a tak k nemu existuje množstvo podporných nástrojov, lexikónov a kvalitne spracovaných anotovaných korpusov. Taktiež, na rozdiel od napríklad slovanských jazykov relatívne voľných v slovoslede s vysokou mierou ohýbania (flexie), ide o jazyk označovaný ako analytický – jazyk s nízkou mierou flexie, komplexnou syntaxou a relatívne stabilným poradím *podmetu - prísudku - predmetu* [23].

Ďalej, zo samotnej podstaty tohoto internetového projektu, sa v jeho textových obsahoch koncentrujú vedomosti, ktoré sa snažia zachytávať objektívnu podstatu tém, o ktorých pojednávajú, čo je bezpochyby užitočné pre projekt cielený na extrakciu faktov.

3.2 Čistenie korpusu

Výpis svojho obsahu, ktorý poskytuje *Wikipédia*, je však vo formáte *XML* a okrem samotného textu obsahuje množstvo redundantných údajov, ako napríklad metadáta udávajúce detaily o vytvorení či editácii jednotlivých článkov. Taktiež samotné telá článkov obsahujú značky modifikujúce jeho vizuálne vlastnosti, obsahujú množstvo hypertextových odkazov, časté špeciálne zápisy pre prvky ako sú zoznamy, tabuľky alebo vložené multimédiá.

Pretože ďalšie kroky tejto práce budú vyžadovať čisto textový obsah v písanom anglickom jazyku, je potrebné pôvodne poskytovaný obsah prečistiť odstránením od nepotrebných častí a užitočné časti skonvertovať do čitateľného obsahu.

3.3 Distribúcia korpusu do výpočtového klastra

Korpus, ktorý máme po kroku čistenia k dispozícii, je ďalej možné rozdeliť medzi viacero výpočtových staníc dostupného výpočtového klastra. Dôvod na začlenenie kroku rozdelenia korpusu do výpočtového klastra až v tejto časti a nie už pri čistení je ten, že výpis z *Wikipédie* je dodávaný ako jediný súbor a samotný proces čistenia stačí vykonať len raz a nie je nijak výpočtovo náročný, takže vôbec neprekáža, ak sa vykoná na iba jednej stanici. To však nie je možné povedať o nasledujúcich úkonoch, ktoré sú nielen výpočtovo náročné, ale je predpoklad, že počas zdokonaľovania systému ich bude potrebné vykonávať opakovane. Našťastie, tieto úlohy je možné vykonávať nezávisle paralelne na viacerých strojoch a aj samotných jadrách procesorov, ak to cieľové stanice výpočtového klastra umožnia.

3.4 Predspracovanie dát

Keďže väčšina dostupných extraktorov, rovnako ako aj extraktor zvolený v našej práci spĺňajúci podmienky *OpenIE*, transformuje vety na množinu vzťahových n-tíc bez akejkoľvek hlbšej analýzy, je v našom záujme text upraviť do takej podoby, aby sme krokom extrakcie neprišli o informácie vyplývajúce zo závislosti na poradí týchto viet a ich kontexte. Samozrejme, nie je v našich silách ošetriť všetky takéto prípady, avšak jednou z najelementárnejších a najdôležitejších úprav je nahradenie nepriamych odkazov na entity za ich výstižnejšie formy. Tento problém už bol spomínaný ako jedna z piatich podúloh extrakcie informácií v podsekcii 2.2.2 označený ako *koreferencia*.

Ďalším úkonom, ktoré je možné v rámci tejto časti vykonať, je odstránenie nadpisov článkov, podnadpisov jeho jednotlivých častí a prípadne iných krátkych kusov textov, ktoré

s veľkou pravdepodobnosťou neobsahujú štruktúru očakávanú extraktorom alebo by sa dalo o nich predpokladať, že neprinesú očakávanú informačnú hodnotu. Za takýto rys by bolo možné považovať rôzne postupnosti znakov, ktorých cieľom je plniť dekoratívny charakter, ale najmä interpunkciou neukončené texty o dĺžke pár slov nachádzajúce sa na samostatných riadkoch oddelených z oboch strán prázdnymi riadkami.

3.5 Extrakcia faktov

Na prečistenú, predspracovanú časť korpusu obsahujúcu v ideálnom prípade iba výstižné a gramaticky správne napísané vety je následne možné použiť náš zvolený *OpenIE* extraktor. Úlohou tohoto extraktora je rozdeliť jednotlivé vety vstupujúceho textu na množinu n-tíc, najčastejšie trojíc tvaru $(e_i, r_{i,j}, e_j)$, kde e_i a e_j sú entity, medzi ktorými reťazec $r_{i,j}$ orientovane vyjadruje ich vzájomný vzťah. V takejto trojici sa zvyčajne dá entita e_i považovať za vetný **podmet**, entita e_j za vetný **predmet** a predikát $r_{i,j}$ označujúci ich vzájomnú reláciu za vetný **prísudok**.

V niektorých prípadoch môže takáto n-tica obsahovať aj ďalšie členy, ktoré najčastejšie rozvíjajú práve spomínaný vetný predmet, obsahujú referencie na iné extrakcie alebo napríklad obsahujú časové modifikátory upresňujúce časové pôsobenie vety.

Extraktor tiež ohodnotí jednotlivé extrakcie váhou vyjadrujúcou pravdepodobnosť, s akou ich považuje za informačne prínosné, napríklad na základe pomeru počtu ich výskytov vzhľadom k celkovému počtu extrakcií. Pri použití takéhoto alebo podobného prístupu potom v prípade, že extraktor nie je schopný vstupný text vhodne spracovať a rozdeliť, môže byť výsledkom veľké množstvo navzájom veľmi podobných ale nie rovnakých extrakcií, ktoré sa teda budú považovať za unikátne, čo môže viesť k ich neadekvátne vysokému ohodnoteniu.

3.6 Čistenie vyextrahovaných faktov

Po dokončení procesu extrakcie je potrebné určiť, ktoré zo získaných n-tíc sú použiteľné pri budovaní znalostnej databázy, a ktoré by mohli jej výslednú kvalitu znižovať alebo komplikovať jej uloženie v databázovom systéme. Nevyhovujúce n-tice je možné úplne odstrániť alebo sa ich pokúsiť previesť do vyhovujúceho tvaru. Medzi takéto úpravy patrí najmä normalizácia predikátu, odstránenie extrakcií obsahujúcich len 2 atribúty, odstránenie extrakcií s nízkou hodnotou užitočnosti, alebo v prípadoch, keď bola vyextrahovaná n-tica s počtom prvkov viac ako štyri pripojiť všetky nadbytočné atribúty k štvrtému, keďže bolo vypozorované, že vo väčšine prípadov spolu súvisia.

Ďalšia vlastnosť mnohých vyextrahovaných n-tíc je tá, že frázy v jej jednotlivých členoch často obsahujú jednu alebo viacero priraďovacích spojok, vďaka ktorým majú tieto frázy nielen nadmernú dĺžku, ale zároveň znižujú výpovednú hodnotu celej n-tice, v ktorej vystupujú.

V nasledujúcej podsekcii 3.6.1 je predstavený vlastný algoritmus na vyriešenie tohoto problému. Za ňou bude nasledovať podsekcia 3.6.2 týkajúca sa normalizovania predikátu za účelom hľadania tried ekvivalencie.

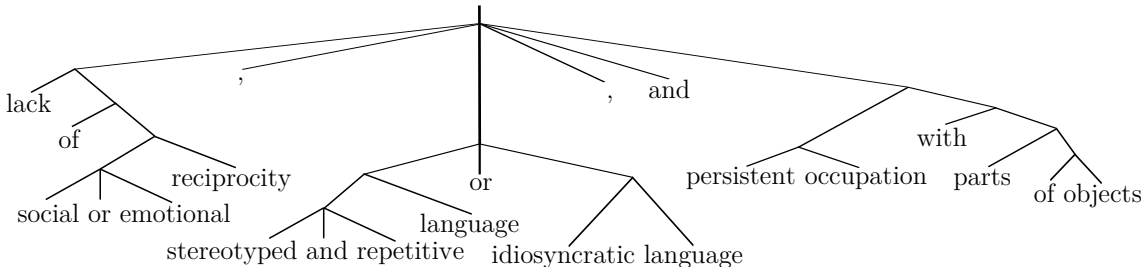
3.6.1 Rozdelenie fráz obsahujúcich priraďovacie spojky

Ako už bolo spomínané v predchádzajúcom texte k tejto sekcii, niektoré atribúty n-tice môžu obsahovať frázy s nadmernou dĺžkou, a to kvôli tomu, že obsahujú niekoľko rovnocenných podfráz pospájaných priraďovacími spojkami *and* (príp. *čiarkou*) a *or*. Nech je

príkladom takéhoto javu nasledujúca veta, kde bodkočiarka naznačuje oddelenie jednotlivých členov n-tice:

> Sample symptoms; include; lack of social or emotional reciprocity, stereotyped and repetitive language or idiosyncratic language, and persistent preoccupation with parts of objects.

Je zrejmé, že problematická je práve pravá strana tejto n-tice. Po aplikácii vhodného syntaktického analyzátora dostávame zložkový derivačný strom znázornený na obrázku 3.2 (označenia jednotlivých vetných členov sú vynechané).



Obr. 3.2: Zjednodušený zložkový derivačný strom frázy „lack of social or emotional reciprocity, stereotyped and repetitive language or idiosyncratic language, and persistent preoccupation with parts of objects".

Vhodným priechodom cez tento strom môžeme získať nasledujúci zoznam fráz, ktoré už priraďovacie spojky obsahovať nebudú: *lack of social reciprocity*, *lack of emotional reciprocity*, *stereotyped language*, *repetitive language*, *idiosyncratic language*, *persistent preoccupation with parts of objects*. Vznik prvých 5 z nich však súvisí s rozgenerovaním aj kvôli vylučovacej spojke *or*, a tak, na rozdiel od pôvodnej frázy, nemusí mať táto nová fráza v danej relácii vety určitú ale má iba *možnú* platnosť.

Na rozdelenie fráz obsahujúcich priraďovacie spojky bol navrhnutý vlastný algoritmus 1, ktorý odstráni priraďovacie spojky z frázy rozgenerovaním na množinu fráz bez týchto spojok. Algoritmus očakáva ako vstup rekurzívne zanorené pole slov organizované ako zložkový derivačný strom danej frázy a jeho výstupom je množina zjednodušených fráz, pričom jednotlivé frázy majú nastavený príznak neurčitosti, ak vznikli rozgenerovaním kvôli vylučovacej spojke *or*. Príznak neurčitosti je následne možné zohľadniť v n-tici zmenením relácie z pôvodného tvaru vyjadrujúceho určitý výskyt udalosti na neurčitý pridaním modifikátoru *can/could/may/might*. Je vhodné podotknúť, že algoritmus považuje akúkoľvek čiarku vyskytujúcu sa vo fráze za rovnocennú zlučovaciu spojku pre *and*, a preto je potrebné vopred zabezpečiť aby do algoritmu vstupovali iba frázy, ktoré neobsahujú podraďovacie súvetia alebo vsuvky.

Vstup : zložkový derivačný strom frázy
Výstup: množina fráz bez priraďovacích závislostí

```
def disambiguateTree(L):
    if isString(L) :
        if L not in ["and", ","] :
            return [L]      // ak ide o slovo, vratime ako jednoprvkove pole
        else:
            return [""]                    // spojku ''and'' a ciarku vynechame
    else:
        S = []                          // zoznam fraz ziskanych v tomto zanoreni
        accS = []                  // akumulator pre novovytvaranych kandidatov
        wasOr = False               // priznak ze sa tu vyskytla aj spojka ''or''
        for e in L :
            if isList(e) :
                dx = disambiguateTree(e)
                if dx == ["or"] :       // spojku ''or'' vynechame az po nastaveni
                priznaku
                    wasOr = True ; dx = [""]
                if dx == [""] :
                    S += accS ; accS = [""]
                elif len(dx) :
                    ...      // vytvor vsetky neprazdne kombinacie konkatenacie
                    accS + '' '' + dx
        S += accS                       // pridaj aj posledny obsah akumulatora
        if wasOr :
            ...                // oznac vsetky prvky v S priznakom neurcitosti
        return [x for x in S if x.length()]
    return []
```

Algoritmus 1: Pseudokód rekurzívneho algoritmu na odstránenie priraďovacích spojok z frázy rozgenerovaním na množinu fráz.

3.6.2 Normalizácia predikátu

Predikáty vo vyextrahovaných n-ticiach obsahujú vetný prísudok vyjadrený slovesom, avšak nielenže toto sloveso môže byť uvedené v rôznych osobách alebo časoch, občas sa tiež môže nachádzať ako súčasť komplikovanejšej frázy zachytávajúcej konkrétne špecifiká relácie, ktorú vyjadruje medzi zvyšnými časťami predmetnej n-tice. Kvôli ďalšej analýze je preto vhodné vedieť takýto predikát normalizovať, aby bolo možné získané extrakcie rozdeliť na špecifické triedy ekvivalencie a kategorizovať napríklad na základe činnosti, ktorá z obsahu ich predikátov vyplýva.

Skúmaním vyextrahovaných predikátov bolo vypozorované, že práve posledné sloveso v postupnosti slov predikátu je nositeľom najvýznamnejšej výpovednej hodnoty. Navrhnutý algoritmus 2 sa spolieha na to, že v gramaticky správnej vete táto vlastnosť naozaj platí vďaka tomu, že, ako už bolo spomínané v sekcii 3.1 tejto kapitoly, pracujeme s analytickým

jazykom, v ktorom je poradie slov dôležité a je pomocou neho vyjadrený význam jednotlivých častí vety.

Samotný algoritmus najprv rozexpanduje skrátené formy negácií, ktoré sa môžu vo vstupnom predikáte nachádzať, a následne využije externé funkcie na jeho rozdelenie na tokeny a určenie zodpovedajúcich slovných druhov. Značky slovných druhov, s ktorými algoritmus ďalej pracuje, vychádzajú z tabuľky 1.1 predstavenej v kapitole 1. V hlavnom cykle algoritmu sa prechádza cez jednotlivé slová a tie, ktoré majú značku vyjadrujúcu modálne alebo akékoľvek iné sloveso, sú pomocou externej funkcie normalizované a následne považované za výsledok normalizácie celého predikátu. Na zrobustnenie algoritmu je tiež modelovaná kontrola, či sa pred identifikovaným slovesom nenachádza aj určitý alebo neurčitý člen, ktorý by zo slovesa robil podstatné meno, čo by síce v prípade bezchybného PoS značkovača nastať nemalo, ale v praxi sa dá stretnúť s obdobne chybnými identifikáciami.

Vstup : predikát určený na normalizáciu
Výstup: normalizovaný predikát

```
def normalizePredicate(predicate):
    predicate = predicate.replace(["won't", "can't", "n't "],      // nahradenie skra-
        ["will not", "cannot", " not "])  // tenych negacii za ich rozvite formy
    words = tokenize(predicate)                                    // tokenizacia
    tags = pos(words)                              // detekcia PoS znaciek k slovam
    normalized = None
    for (i = 0; i < words.length(); i++) :
        if i > 0 and (words[i - 1] in ["a", "the"]) :
            continue                // za clenom vzdy nasleduje podstatne meno
        if tags[i] in ['MD', 'VB', 'VBD', 'VBG', 'VBN', 'VBP', 'VBZ'] :
            normalized = normalizeVerb(words[i])          // normalizacia slovesa
    return normalized
```

Algoritmus 2: Pseudokód algoritmu na normalizáciu predikátu.

V uvedenom algoritme sa spomína aj externá funkcia *normalizeVerb()* na normalizáciu jedného slovesa. Anglický jazyk rozlišuje iba medzi pravidelnými slovesami, ktoré vzniknú pripojením prípony *-ed*, a nepravidelnými slovesami, ktorých je konečný počet. Vďaka tomu je možné túto externú funkciu jednoducho realizovať s využitím manuálne zostavenej tabuľky na normalizáciu nepravidelných slovies.

3.7 Analýza extrakcií

Získané n-tice je potrebné ďalej analyzovať za účelom nadobudnutia doplňujúcich znalostí o týchto extrakciách, pretože sprístupnia QA systému pracujúcemu s týmto obsahom uloženým vo vedomostnej databáze dodatočné informácie umožňujúce lepšiu špecifikáciu dotazov, čo je predpoklad aj pre presnejšie odpovede.

3.7.1 Určovanie typu frázy

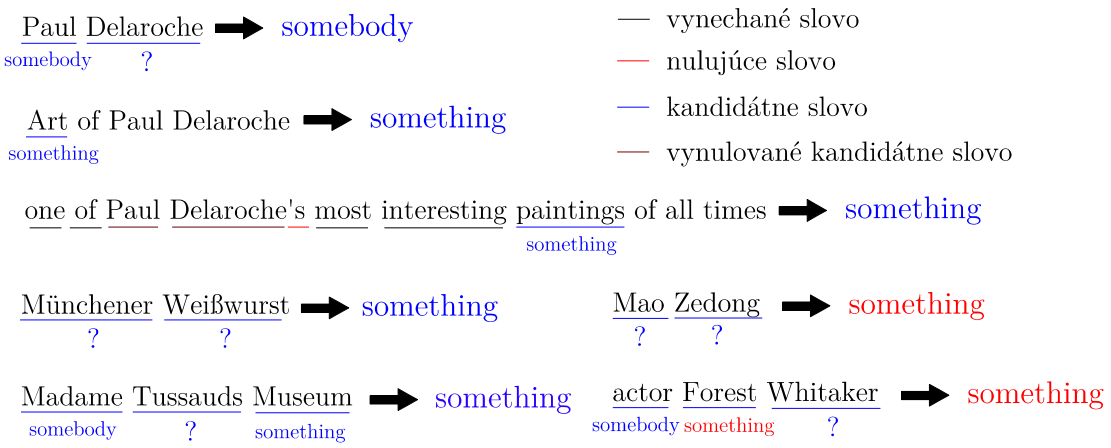
Jednou z takýchto analýz je určovanie typu frázy, ktoré umožní zaradiť frázu na základe toho či pojednáva o osobe, mieste, veci alebo časovom údaji. Tieto informácie môžu byť

ďalej využité na zodpovedanie otázok typu *Kto?*, *Čo?*, *Kde?* alebo *Kedy?* a preto navrhnuté kategórie boli nazvané ako *niekto*, *niečo*, *niekde*, resp. *niekedy*.

Navrhnutý algoritmus 3 sa zameriava na rozlíšenie práve prvých dvoch z týchto kategórií, pričom veľmi podobne je možné realizovať aj odlíšenie kategórie *niekde*. Zároveň však túto kategóriu spolu s kategóriou *niekedy* je možné realizovať aj za použitia regulárnych výrazov, keďže možné spôsoby vyjadrenia miesta a času sú v anglickom jazyku z veľkej časti predvídateľné a vo väčšine prípadov obsahujú opakujúce sa kľúčové predložky alebo ďalšie špecifické slová.

Algoritmus 3 prijíma ako svoj vstup ľubovoľnú frázu, ktorú následne roztokenizuje a pre jednotlivé tokeny určí slovné druhy. Potom cez tieto usporiadané označené tokeny prechádza tak, že sa snaží identifikovať vhodných kandidátov na určenie typu, a to konkrétne podstatné mená. Pri určovaní kandidátov vychádza z predpokladu, že práve podstatné mená sú jadrom frázy obsahujúcej podmet, resp. predmet a nie ojedinele sú tvorené postupnosťou tohoto slovného druhu. Preto sa najprv snaží navrhovaný algoritmus objaviť prvé podstatné meno a následne ukladať všetky kandidátne slová tvorené postupnosťou podstatných mien až do prerušenia tejto postupnosti. Špecifickým prípadom je token `'s` vyjadrujúci privlastňovací vzťah vzhľadom na ďalšiu časť frázy, a preto v prípade jeho objavenia začína proces hľadania kandidátov nanovo, avšak tentokrát sú vynechávané pri hľadaní prvého kandidáta iba slová vyjadrujúce jeho kvalitatívne vlastnosti.

Po zostavení množiny kandidátov, u ktorých je predpoklad, že pomenúvajú alebo rozvíjajú tú istú osobu, resp. *niečo*, je využitá databáza *WordNet*. Nad každým z týchto slov je vytvorený tranzitívny uzáver jeho hyperným, pričom sa snaží nájsť špecifické hypernymum *living_thing* označujúce živú bytosť alebo *object* pomenúvajúce väčšinu členov kategórie *niečo*. Na záver je porovnaný počet kandidátov, ktorý spadajú do každej z týchto kategórií a v prípade, že počet kandidátov označujúcich živú bytosť prevýši druhú kategóriu, je daná fráza prehlásená za typ *niekto*. Po vylúčení, že fráza nie je ani typu *niekde* alebo *niekedy* je možné o tejto fráze ďalej prehlásiť, že s najväčšou pravdepodobnosťou spadá do kategórie *niečo*.



Obr. 3.3: Príklady analýzy problematických fráz algoritmom 3.

Obrázok 3.3 názorne ilustruje priebeh analýzy niektorých problematických fráz, ktoré by mohli byť vstupom do vyššie uvedeného algoritmu 3. Pod slovom každej frázy je čiarou

Vstup : fráza určená na analýzu
Výstup: typ podmetu – *niekto* alebo *niečo*

```
def detectType(phrase):
    words = tokenize(phrase)                                    // tokenizacia
    tags = pos(words)                              // detekcia PoS znaciek k slovam
    nouns = []                                  // zoznam kandidatov na urcovanie typu
    firstpass = True                    // urcuje ci ide o prvy priechod cez cyklus
    i = 0
    // najprv vyhladame vsetkych vhodnych kandidatov na urcovanie typu
    while i < words.length() and (firstpass or (tags[i] == 'POS')) :
        if tags[i] == 'POS' :                                  // 's hladanie vynuluje
            nouns = []
        if firstpass :                  // v prvom priechode hladame prve podst. m.
            while i < words.length() and tags[i] not in ['NN','NNS','NNP','NNPS'] :
                i += 1
        else:      // v druhom priechode preskakujeme iba vlastnosti pods.m.
            while i < words.length() and tags[i] in ['CC', 'JJ', 'JJR', 'JJS', 'PRP',
            'PRP$'] :
                i += 1
        while i < words.length() and tags[i] not in ['NN','NNS','NNP','NNPS'] :
            nouns.append(words[i])
            i += 1
        i += 1
        firstpass = False
    // hodnotenie typov jednotlivych slov
    score = {'something': 0, 'somebody':0}
    for w in nouns.reversed() :
        candidades = WordNet.getNouns(w) // zoznam pods.m. pre w vo WordNet
        for c in candidates :
            hypernyms = WordNet.hypernymClosure(c)
                                // tranzitivny uzaver nad hypernymami kandidata
            if "living_thing" in hypernyms :
                score['somebody'] += 1
            elif "object" in hypernyms or "abstraction" in hypernyms :
                score['something'] += 1
    // typ s vacsim hodnotenim je vysledok
    return (score['somebody'] > score['something']) ? "somebody" : "something"
```

Algoritmus 3: Pseudokód algoritmu na určovanie typu frázy.

označené, ak algoritmus prechádzal cez dané slovo. V prípade, že ho preskočil, je toto slovo podčiarknuté čiernou čiarou. V prípade, že toto slovo bolo vyhodnotené ako kandidátne, teda označené jednou zo značiek patriacej slovnému druhu podstatné meno, je takéto slovo podčiarknuté na modro a pod ním zapísaný typ, ktorý mu bol na základe hypernymového uzáveru pridelený. V prípade, že sa nepodarilo určiť typ podstatného mena, je pod týmto slovom otáznik. Červenou farbou je označený chybne detegovaný typ alebo v prípade červenej čiary ide o miesto vynulovania kandidátneho zoznamu a slová, ktoré boli z tohoto dôvodu zo zoznamu kandidátnych odstránené, sú podčiarknuté hnedou farbou. Za každou frázou je šípka, za ktorou je uvedený výsledný určený typ frázy, modrou farbou ak bol detegovaný správne a červenou nesprávne.

Na prvom príklade francúzskeho akademického maliara z prvej polovice 19. storočia, ktorého meno je Paul Delaroche, vidíme, akým spôsobom je vyhodnocované meno osoby, pri ktorom je bežné krstné meno identifikované ako *niečo* a priezvisko je pre databázu WordNet neznáme. Vďaka správnemu označeniu krstného mena je celá fráza vyhodnotená správne, teda ako *niekto*. Problém môže nastať v prípade nositeľov exotickejších mien, ako napríklad čínskeho diktátora menom Mao Ce-tung (angl. Mao Zedong), kde sú obe časti tohoto mena pre databázu *WordNet* neznáme, čo vedie k určeniu typu frázy ako *niečo*. Vo väčšine prípadov však postupnosť neznámych podstatných mien označuje naozaj práve objekty, čo je aj prípad správne detegovaného názvu istého, pre *WordNet* neznámeho, druhu nemeckej klobásy uvedeného ako jeden z príkladov.

Často sa pred menom osoby nachádza aj povolanie, ktoré je ľahko detegované ako *niekto*, čo vo väčšine prípadov vedie k správnej detekcii fráz ako napríklad leader Zedong, kde je uvedené priezvisko neznáme pre sieť *WordNet* ale slovo leader je správne určené ako osoba. Ojedinelý problém môže nastať pokiaľ má – v tomto prípade – meno, či už krstné alebo priezvisko, význam aj ako bežný objekt. Príkladom môže byť meno známeho herca Foresta Whitakera, ktorého krstné meno v angličtine tiež znamená *les*.

Ďalším fenoménom zachyteným na druhom príklade Art of Paul Delaroche, ktorý môže predstavovať podmet a zároveň začiatok nejakej vety, je, že v prípade dlhších slovných spojení, ktoré obsahujú spojky alebo predložky, sa analýza môže zastaviť už na začiatku tohoto pomenovania a určiť správny typ frázy. Toto sa týka väčšiny názvov inštitúcií, ale často sa tiež vzťahuje na pomenovania významných diel a podobne. Na treťom riadku obrázka 3.3 je uvedený pravdepodobne najkomplikovanejší príklad, ktorý ilustruje fakt, že aj keď je kandidátna množina určená na detekciu už zostavená, v prípade, že po nej nasleduje token `'s`, je potrebné spustiť hľadanie kandidátov nanovo.

3.8 Reprezentácia dát pomocou RDF

Po tom, ako jednotlivé stanice výpočtového klastra dokončia všetky výpočtovo náročné operácie zahrňujúce extrakciu faktov a ich prečistenie, je možné výsledky ich činnosti uložiť do distribuovaného databázového servera, ktorý bude pri ďalšom využívaní vyhodnocovať dotazy mierené do rozsiahlej vedomostnej databázy rýchlejšie ako jednoduchý databázový server na jednej stanici. Keďže spracované extrakcie majú tvar n-tíc, v ktorých vystupujú entity a relácie, jednou z najvýhodnejších známych technológií na ich uloženie a ďalšiu efektívnu prácu s nimi je databáza podporujúca formát RDF a dotazovací jazyk SPARQL, z ktorých bude prvá technológia predstavená v zvyšku tejto sekcie a druhá v podsekcii 4.1.3 nasledujúcej kapitoly týkajúcej sa implementácie.

Ako už bolo naznačené, v súvislosti s problémom extrakcie informácií automaticky vzniká otázka, ako tieto získané informácie vhodne uchovať. Keďže predmetom záujmu je oblasť,

v ktorej prichádza k veľmi dynamickému meneniu požiadaviek na to, aké vzťahy sa chcú modelovať, a vlastnosti, ktoré je potrebné uchovávať vzhľadom na rozličný kontext ich pôvodu, je potrebné pracovať s dostatočne flexibilnými technológiami umožňujúcimi bezproblémovo reagovať na požiadavky výskumníkov v priebehu zdokonaľovania systémov, ktoré vyvíjajú.

RDF (Resource Description Framework) je sada špecifikácií navrhnutých organizáciou W3C zameraná na modelovanie informácií prostredníctvom orientovaných grafov tak, aby boli čitateľné ľudsky a zároveň aj strojovo [24]. K tomu využíva trojice *subjekt – predikát – objekt*, ktoré popisujú orientované vzťahy medzi dvoma entitami prostredníctvom vzťahu určeného predikátom.

Na rozdiel od konvenčných relačných databáz, kde sú dáta v tabuľkách identifikovateľné na základe primárnych kľúčov, v prípade RDF sú entity v subjekte určené pomocou ***URI (Uniform Resource Identifier)***, ktorý predstavuje jedinečný identifikátor označujúci každú vystupujúcu entitu. Niekedy je však vhodnejšie využiť tzv. *anonymné uzly*, ktoré tento explicitne zvolený názov dôležitý pre ďalšiu dohľadateľnosť zámerne zanedbávajú a nechávajú na použitom interpreteri jazyka aby vygeneroval náhodný identifikátor. Ďalšou odlišnosťou od dát uložených v populárnych relačných databázach je možnosť ich heterogenity. V databázach, kde sú informácie organizované v tabuľkách, je nutné, aby mali všetky záznamy identický počet vlastností. Reprezentácia RDF však umožňuje oveľa flexibilnejší popis zdrojov, a to na základe iba tých vlastností, ktoré sa daného zdroja naozaj týkajú.

Na vyjadrenie predikátu sa *URI* využíva taktiež, avšak rozdiel oproti zdroju je ten, že musí mať jasne špecifikovanú sémantiku, ktorá sa za daným vzťahom skrýva. Ako obsah objektu pravej časti trojice je možné jednak využiť *URI* referenciu na iný zdroj, alebo tzv. *literál* reprezentovaný reťazcom. Tento literál môže byť určitého dátového typu, ako napríklad číslo alebo pravdivostná hodnota, je ho však tiež možné označiť voliteľným identifikátorom konkrétneho jazyka, ku ktorému sa vzťahuje.

3.8.1 RDF syntax

Na popis zdrojov v jazyku RDF sa používajú 2 základné typy notácie, a to ako ***XML/RDF***, kde sú dáta serializované vo formáte *XML*, alebo ako varianty ***Notation3*** určeného na kompaktný zápis trojíc. V prípade druhej uvedenej notácie ide o zápis, ktorý navrhol vynálezca webu Sir Tim Bernes-Lee a zahŕňa tiež funkcie z predikátovej logiky prvého rádu. Jej nevýhodou je však pomalé spracovanie, a tak bola vytvorená syntax ***Turtle***, ktorá je podmnožinou funkcionality poskytovanej *Notation3* a podporuje jednoduchý, kompaktný a čitateľný zápis. Ďalšou variantou, ktorá vznikla ako podmnožina vyjadrovacích schopností *Turtle* je ***N-Triples***, ktorá odstraňuje možnosti kompaktného zápisu a vedie k vyjadrovaniu pomocou explicitne zapísaných trojíc umožňujúcich rýchlejšie spracovanie systémovými nástrojmi alebo zabezpečuje vyšší kompresný pomer [25].

Príklad zápisu osoby porovnávajúci syntax *XML/RDF*, *Turtle* a *N-Triples* je uvedený ako kód 3.1. V druhých dvoch prípadoch sa *URI* zapisuje medzi dvojicou symbolov < a > a jednotlivé trojice sú ukončené pomocou bodky. V syntaxi *Turtle* je umožnené použiť symbol ; na oddelenie viacerých vlastností týkajúcich sa rovnakého popisovaného zdroja.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:abc="nul://example/people#"
   xmlns:adama="nul://example/people#adam"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
      <rdf:Description rdf:about="nul://example/people#adam">
            <abc:hasName>Adam Adamcek</abc:hasName>
```

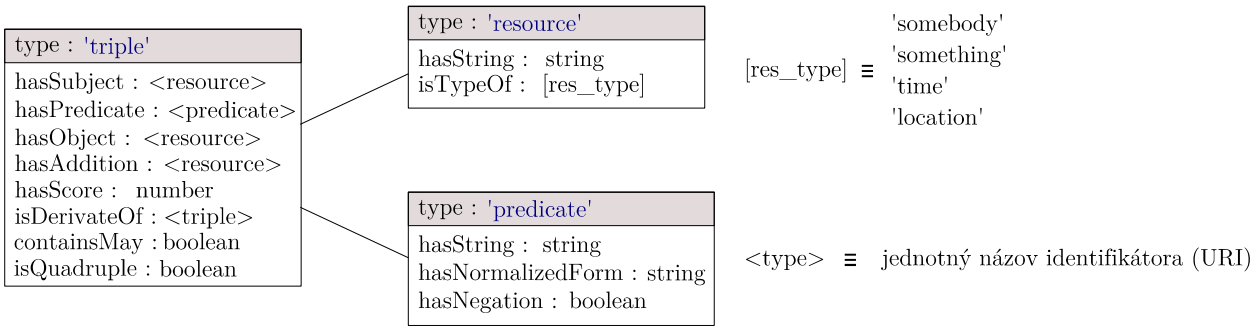
```
        </rdf:Description>
        <rdf:Description rdf:about="nul://example/people#adam">
            <abc:bornIn rdf:resource="nul://example/countries#Slovakia"/>
        </rdf:Description>
        <rdf:Description rdf:about="nul://example/people#adam">
            <rdf:type rdf:resource="nul://example/people#Person"/>
        </rdf:Description>
</rdf:RDF>
--
<nul://example/people#adam> <abc:hasName> "Adam Adamček";
                            <abc:bornIn> <nul://example/countries#Slovakia>;
                            <rdf:type> <nul://example/people#Person>.
--
<nul://example/people#adam> <abc:hasName> "Adam Adamček".
<nul://example/people#adam> <abc:bornIn> <nul://example/countries#Slovakia>.
<nul://example/people#adam> <rdf:type> <nul://example/people#Person>.
```

Kód 3.1: Príklad osoby zapísanej postupne v syntaxi *XML/RDF*, *Turtle* a *N-Triples*.

3.8.2 Návrh modelu databázy

Formát RDF prináša špecifický spôsob uloženia dát, a preto je potrebné vedieť nielen identifikovať správne informácie, ktoré máme záujem uchovávať, avšak aby s nimi bolo možné aj ďalej pracovať a ľahko naprieč nimi vyhľadávať, je v prvom rade potrebné navrhnúť štruktúry, ktorými budú reprezentované.

Obrázok 3.4 zobrazuje objekty, ktoré je potrebné vytvoriť za účelom reprezentovania všetkých častí extrakcií, a ich atribúty, ktorých úlohou je zachytiť vlastnosti týchto extrakcií a každej jej komponenty. V modeli vystupujú tri typy objektov – *trojica*, *zdroj* a *predikát*. Tak ako je uvedené v podsekcii 3.8 predchádzajúcej kapitoly, každý z týchto objektov je reprezentovaný unikátnym identifikátorom *URI*, ku ktorému sú prostredníctvom vopred vyšpecifikovaných atribútov uvedených v danom obrázku pripojené hodnoty týchto atribútov.



Obr. 3.4: Objekty databázy navrhovaného systému a ich vlastnosti, ktoré je potrebné modelovať pomocou jazyka RDF.

Trojica (triple) reprezentuje extrakciu ako kombináciu ďalších objektov, z ktorých sa skladá. Dve rôzne extrakcie by sa mali zobraziť na rovnaký objekt trojice práve keď sa skladajú z rovnakých častí. Prostredníctvom relácie *hasSubject* je pripojený objekt *zdroj* reprezentujúci prvú časť tejto trojice – podmet, rovnaký typ objektu je využitý aj na pripojenie predmetu pomocou relácie *hasObject* a tiež voliteľnej dodatočnej časti extrakcie cez

reláciu *hasAddition*. Prostredníctvom relácie *hasPredicate* je pripojený objekt typu *predicate* reprezentujúci časť s prísudkom. Ohodnotenie extrakcie reprezentované reálnym číslom je pripojené cez vzťah *hasScore*. Ako už bolo načrtnuté, pôvodná extrakcia sa mohla skladať nielen z troch, ale aj viacerých častí, z ktorých v ďalších krokoch postupu je tento počet zredukovaný na najviac štyri časti a táto štvrtá časť je označovaná ako *doplnok*. To, či sa vyskytuje ako súčasť určitej extrakcie je vyjadrené boolovskou hodnotou cez vzťah *isQuadruple*. Ako bolo spomenuté v podsekcii 3.6.1 o čistení získaných extrakcií, n-tice, ktoré obsahujú priraďovacie spojky, je možné rozgenerovať na viacero n-tíc. Takto vzniknuté nové n-tice potom v našom modeli odkazujú na pôvodnú n-ticu prostredníctvom relácie *isDerivateOf*. Ak novovzniknutá n-tica bola vygenerovaná kvôli spojke *or*, je tiež pozitívne nastavený boolovský príznak cez vzťah *containsMay*.

Ďalším typom objektu je **zdroj (resource)**, ktorý reprezentuje akúkoľvek frázu okrem predikátu n-tice. Prostredníctvom relácie *hasString* je k tomuto objektu pripojený samotný textový obsah. Podsekcia 3.7.1 sa venovala analýze fráz extrakcií a určovania ich typov. Tento výsledok je možné pripojiť k objektu reprezentujúcemu *zdroj* prostredníctvom relácie *isTypeOf*. Rovnako je však možné túto reláciu využiť aj na pripojenie informácie o tom, či fráza reprezentuje čas alebo miesto, ak je možné získať aj takýto poznatok zo zdrojovej frázy.

Posledný typ objektu reprezentuje **predikát (predicate)** trojice. Jeho textový obsah je rovnako ako pri type *zdroj* pripojený prostredníctvom vzťahu *hasString*. Ďalšia informácia, ktorú je možné pripojiť prostredníctvom relácie *hasNormalizedForm* je normalizovaná podoba hlavného slovesa obsiahnutého v tomto predikáte. Pomocou vzťahu *hasNegation* je taktiež možné pripojiť prostredníctvom boolovskej hodnoty aj informáciu o tom, či je predikát vyjadruje negáciu alebo nie.

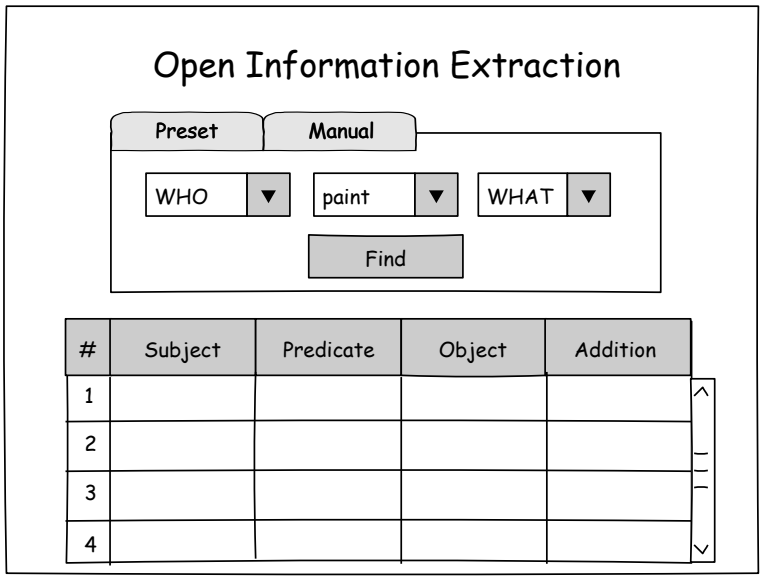
Je vhodné uviesť, že všetky neštandardné typy použité v návrhu je v jazyku RDF možné reprezentovať typom *literál*. Identifikátory objektov je samozrejme nutné reprezentovať RDF typom *URI*, ktorý musí byť zo samotnej podstaty unikátny.

3.9 Prezentácia získaných dát

Na záver je potrebné disponovať rozhraním, ktoré umožní používateľom systému ľahko prehliadať obsah zostavenej databázy so získanými extrakciami a zostavovať nad touto databázou dotazy s cieľom zodpovedania ich otázok. Na tento účel je vhodné navrhnúť intuitívne webové rozhranie – tzv. **front-end**, prístupné cez ľubovoľný moderný webový prehliadač, prostredníctvom ktorého budú mať títo používatelia možnosť špecifikovať svoje dotazy na tzv. **back-end** komunikujúci s databázou extrakcií a výsledky im budú vo vhodnej forme okamžite prezentované.

Jedna z možností je navrhnutie rozhrania fungujúceho na princípe QA systému popisovaného v kapitole 1. Vytvorenie takéhoto rozhrania schopného pochopiť voľne zadanú otázku v prirodzenom jazyku, transformovať ju na správnu štruktúru dotazu, následne získať zoznam výsledkov z databázy a tie vhodne interpretovať a zostaviť do výslednej odpovede však na rozumne použiteľné fungovanie vyžaduje ďalšie značné množstvo krokov a ide o ďalšiu veľmi rozsiahlu oblasť výskumu. Preto na účely tejto práce bude navrhnuté rozhranie, ktoré bude síce oveľa menej automatizované, a tak pravdepodobne nebude úplne vhodné na používanie akýmkoľvek bežným používateľom, ale z hľadiska možností, ktoré poskytne, umožní výskumníkovi pracujúcemu s týmto systémom v konečnom dôsledku oveľa väčšiu kontrolu a prehľad nad dotazmi a ich vyhodnotením.

Na obrázku 3.5 je zobrazený návrh webového používateľského rozhrania, ktoré by malo vykonávať vyššie popisovaný zámer. Ako vidíme, používateľ by mal možnosť voľby medzi zjednodušeným rozhraním, ktoré by plnilo účel QA systému, a manuálnym rozhraním, ktoré by umožnilo oveľa podrobnejšie špecifikovať dotaz smerujúci na back-end. Po stlačení tlačidla určeného na spustenie vyhľadávania by sa mala zobraziť tabuľka, v ktorej budú zobrazené všetky extrakcie, ktoré spĺňajú vyhľadávacie kritériá.



Obr. 3.5: Návrh používateľského rozhrania určeného na prístup k databáze s extrakciami.

Kapitola 4

Implementácia

V tejto kapitole si predstavíme technológie zvolené pre vytvorenie výsledného systému a ukážeme, ako zapadajú do nášho návrhu. Taktiež budú uvedené dôvody, pre ktoré boli jednotlivé technológie zvolené. Úplne nakoniec budú spomenuté problémy, s ktorými som sa v priebehu implementácie stretol a bolo ich potrebné vyriešiť.

4.1 Použité technológie

V tejto sekcii sú predstavené technológie, ktoré boli využité na implementáciu a sprehľadnenie vývoja, uvedené v prípade základných komponent riešenia spolu aj so spôsobom ich použitia. Ku každej technológii sú uvedené jej výhody a prípadne nevýhody, ktoré vzhľadom na riešený problém poskytujú.

4.1.1 Skriptovacie jazyky Bash, Python a systém správy verzií Git

Keďže samotné riešenie pozostáva z viacerých systémov tretích strán, ktoré bolo treba postupne automatizovane využívať, na dosiahnutie nášho zámeru boli zvolené skriptovacie jazyky *Bash* a *Python*.

Skriptovací jazyk ***Bash*** je jazyk interpretovateľný v rovnomennom príkazovom procesore vytvorenom pre interakciu používateľov s operačným systémom Unix ako tzv. *shell*. Bol vyvinutý pre Projekt GNU ako shell rešpektujúci štandard POSIX, ale zároveň podporujúci viacero rozšírení, ktoré zjednodušujú jeho používanie. V rámci implementačnej časti tejto práce bol zvolený najmä na vytváranie skriptov týkajúcich sa automatizovanej práce s výpočtovým klastrom, ako napríklad distribúcia či agregácia súborov, monitorovanie bežiacich procesov a na zjednodušené spúšťanie vlastných skriptov napísaných v jazyku *Python* [26].

Jazyk ***Python*** je vysokoúrovňový interpretovaný programovací jazyk kompilovaný štandardne do medzikódu, ktorý bol vybraný hneď z niekoľkých dôvodov. Podporuje objektovo orientované, funkcionálne programovanie s automatickou správou pamäte a silným ale dynamickým typovaním, a tak na rozdiel od rozšírenejších jazykov, ako napríklad C/C++, je v ňom možný oveľa rýchlejší a pohodlnejší vývoj. Množstvo vysokoúrovňových dátových typov, ktoré poskytuje, je mimoriadne účelné pri práci s textom. Najväčšou výhodou, ktorú však prináša, je nespočetné množstvo už existujúcich riešení vo forme rozšírení a voľne dostupných skriptov z mnohých repozitárov [27]. Navyše, ide o veľmi obľúbenú a častú voľbu v komunite spracovania prirodzeného jazyka, a tak, ak existuje program z oblasti NLP napísaný aj v inom programovacom jazyku, často sa dá nájsť wrapper prinášajúci jeho pythonovské rozhranie.

Na prehľadnú organizáciu súborov a uchovávanie zmien bol v rámci dodržiavania dobrých vývojárskych praktík používaný systém správy verzií ***Git***. Tento systém, ktorý bol vyvinutý tvorcom operačného systému Linux, kladie dôraz na rýchlosť, opatrenia zaisťujúce integritu dát a podporu distribuovaného a nelineárneho vývoja. V dnešnej dobe patrí medzi najpopulárnejšie voľby pri spolupráci na slobodnom softvéri [28]. Jeho používanie umožnilo udržiavať prehľad o zmenách vykonaných v implementovaných súboroch tejto práce, vďaka čomu bolo možné rýchlejšie odhaľovať novovzniknuté chyby počas vývoja a experimentovanie s viacerými spôsobmi implementácie niektorých častí. Vďaka využívaniu cloudového úložiska bolo tiež zabezpečené automatické zálohovanie riešenia v prípade nepredvídateľných porúch hardvéru, na ktorom bola práca vyvíjaná.

4.1.2 Formát JSON

JavaScript Object Notation, skrátene ***JSON***, je otvorený štandard využívajúci čitateľný text na serializáciu dátových objektov pozostávajúcich z vlastností a ich zodpovedajúcich hodnôt.

Medzi jeho najväčšie výhody okrem aj pre človeka ľahko čitateľného zápisu patrí nezávislosť na použitej platforme a v porovnaní so značkovacím jazykom *XML* zvyčajne výrazne menšia veľkosť dát potrebná na popis rovnakého obsahu, ktorý sa často využíva pri asynchrónnej komunikácii klienta a servera. Medzi možné nevýhody patrí absencia názvoslovia a formálneho popisu gramatiky dokumentu či neefektívne kódovanie binárnych dát [29].

Formát *JSON* ignoruje biele znaky a rozlišuje 6 základných dátových typov [30]:

- *číslo* – čísla v desiatkovej sústave s možným znamienkom, desatinnou časťou alebo v exponenciálnej forme
- *reťazec* – ľubovoľne dlhá postupnosť Unicode znakov s podporou tzv. escape sekvencií medzi dvojicou znakov `"`
- *boolovská hodnota* – buď `true` alebo `false`
- *pole* – ľubovoľne dlhá zoradená postupnosť ľubovoľných dátových typov zapísaná medzi dvojicou znakov `[` a `]`
- *objekt* – nezoradená kolekcia dvojíc `atribút:hodnota` zapísaná medzi dvojicou znakov `{` a `}`, kde atribút je reprezentovaný reťazcom a hodnotou môže byť ľubovoľný dátový typ
- *prázdna hodnota* – reprezentovaná slovom `null`

Spôsob zápisu v tomto formáte je demonštrovaný na nižšie uvedenom kóde 4.1, ktorý ukazuje príklad uloženia vety po rozdelení na tokeny a určení ich slovných druhov.

```
{
  'score': 0.73,
  'tokenized': true,
  'words': [
    {'token': 'Eve',   'tag': 'NNP'},
    {'token': 'gave',  'tag': 'VBD'},
    {'token': 'Adam',  'tag': 'NNP'},
    {'token': 'a',     'tag': 'DT' },
    {'token': 'red',   'tag': 'JJ' },
    {'token': 'apple', 'tag': 'NN' }
```

```
  ]
}
```

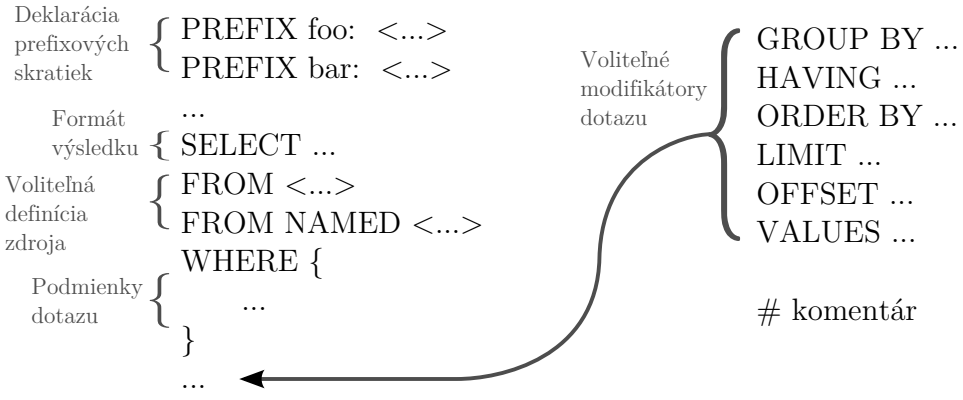
Kód 4.1: Príklad uloženia analyzovanej vety vo formáte JSON.

4.1.3 Dotazovací jazyk SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) [31] je jazyk určený na dotazovanie databázových serverov podporujúcich formát RDF a manipuláciu v nich uložených trojíc. Tento dotazovací jazyk, ktorého najnovšia špecifikácia verzie 1.1 vyšla v marci roku 2013, prináša oproti tradičnému jazyku SQL výhodu v jasnejšej reprezentácii vyjadrovaného dotazu v súvislosti s používateľovým chápaním danej oblasti záujmu. Dotazy v SQL reflektujú špecifickú štruktúru relačnej databázy a ako sú v nej dáta v tabuľkách uložené, pričom dotazy v SPARQL cielené na entity a súvislosti medzi nimi sa dajú vyjadriť v oveľa prirodzenejšej forme pre ľudské chápanie [32].

Formát SPARQL dotazu

Štruktúra bežného SPARQL dotazu je zobrazená na obrázku 4.1. Takýto dotaz obsahuje vo svojom úvode deklaráciu skrátených výrazov pre prefixy, ktoré sa v zvyšku správy opakujú v *URI* identifikátoroch zdrojov. Po nich, za kľúčovým slovom `SELECT` nasleduje vymenovanie premenných, ktorých obsahy budú uvedené vo výslednej odpovedi k tomuto dotazu. Kľúčové slová `FROM` umožňujú uviesť obmedzenia na zdrojové grafy, z ktorých budú dáta pri vyhľadávaní odpovede čerpané.



Obr. 4.1: Štruktúra bežného SPARQL dotazu [33].

Časť `WHERE` obsahuje samotné telo s podmienkami dotazu obsahujúcimi premenné začínajúcimi symbolom `?`. Pri prehľadávaní RDF databázy sa na tieto premenné nadväzujú rôzne dostupné hodnoty tak, aby platilo toto telo s podmienkami ako celok. Pokiaľ sa tak stane, je podľa aktuálne naviazaných hodnôt v premenných zostavený záznam odpovede, ktorý bol špecifikovaný vyššie v časti `SELECT`.

Pri zápise podmienok sa využíva notácia veľmi podobná syntaxi *Turtle* predstavenej v sekcii 3.8.1. Podmienky je možné zoskupovať pomocou symbolov `{` a `}`, na konjunkciu slúži operátor bodka. Pomocou kľúčového slova `OPTIONAL` je možné pokúsiť sa pripojiť podmienku uvedenú na pravej strane k podmienke pripojenej na ľavej strane, avšak tak,

aby to vyhodnotenie podmienky naľavo neovplyvnilo. Operátor `UNION` slúži na zlúčenie výsledkov a `MINUS` na odstránenie výsledkov, ktoré sú získané podmienkou na pravej strane tohoto operátora.

Veľmi užitočná je špeciálna podmienka `FILTER(výraz)`, ktorá umožní redukciu potenciálnych výsledkov iba na tie, pre ktoré sa `výraz` vyhodnotí ako pravdivý. V rámci výrazu je možné využívať rôzne logické operátory a podporné funkcie, ktoré však sú závislé na podpore zo strany samotného databázového systému. Medzi najužitočnejšie štandardne podporované funkcie patria testy uzlov na to, či sú určitého dátového typu, alebo testovanie obsahov literálov na základe kritérií špecifikovanými regulárnymi výrazmi.

Záverečná časť kľúčových slov modifikuje zoznam výsledkov zoradením, obmedzením ich počtu, zoskupovaním a podobne. Za zmienku stojí tiež užitočná varianta dotazu, ktorá namiesto slova `SELECT` využíva kľúčové slovo `ASK`. V tomto prípade je odpoveďou na dotaz `true` ak existuje neprázdny zoznam výsledkov, v opačnom prípade systém odpovie hodnotou `false`.

4.1.4 Decipher NER

Nástroj ***Decipher NER*** je vyvíjaný *Ing. Lubomírom Otrusinom* a kolektívom na *Fakulte informačních technologií VUT* v Brne v rámci projektu *Decipher* [34]. Primárna úloha tohoto nástroja fungujúceho na princípe prefixových stromov a konečných automatov vyhľadávajúcich v jeho znalostnej databáze je rozpoznávanie pomenovaných entít, ktoré bolo už spomínané v podsekcii 2.2.2. Zvláda však tiež jednoduchú rezolúciu zámen, čo je v kombinácii s rýchlosťou, ktorú ponúka, a podporou prúdového spracovania textu oveľa užitočnejšie riešenie ako iné nástroje poskytujúce obdobnú funkcionalitu. Medzi jeho nevýhody však (aspoň momentálne) patrí neschopnosť rezolúcie iných slovných druhov ako zámen a nie príliš ojedinelá chybná identifikácia niektorých entít, ktorá sa dá čiastočne vyriešiť ďalším rozšírením jeho znalostnej databázy. Nástroj je však stále vo vývoji, a tak sa dá predpokladať, že s jeho ďalšími verziami budú mnohé jeho nedostatky odstránené.

Nástroj *Decipher NER* rozpoznáva aktuálne nasledujúcich 15 pomenovaných entít, ku ktorým je schopný poskytnúť rôzne ďalšie informácie zo svojej znalostnej databázy:

- osoba
- umelec
- miesto
- umelecké dielo
- múzeum
- udalosť
- forma vizuálneho umenia
- médium vizuálneho umenia
- žáner vizuálneho umenia
- hnutie z umeleckého obdobia
- národnosť
- mytológia
- rodina
- skupina
- iné

4.1.5 Nástroj na extrakciu faktov Open IE

Na extrakciu faktov z častí korpusu nachádzajúcich sa na jednotlivých serveroch výpočtového klastra bol zvolený nástroj s názvom ***Open IE***, ktorého zdrojové kódy sú voľne dostupné na internete pod licenciou *University of Washington Academic License* [35].

Open IE prijíma vety na spracovanie na štandardnom vstupe, pričom na každom riadku očakáva práve jednu. V prípade, že je program spustený s prepínačom `--split`, je možné

zadávať aj väčšie bloky textu obsahujúce viacero viet, pričom program sa ich pokúsi sám pred spracovaním rozdeliť. Jedna z možností je spustenie tohoto extraktora aj na konkrétny súbor, avšak kvôli tomu, že samotná doba spustenia tohoto programu po úplné načítanie jeho vnútorných závislostí trvá zvyčajne vyše 2 minúty, takáto forma používania na priebežné spracovávanie postupne vytváraných dát nie je príliš praktická.

Po vykonaní extrakcie nad jednotlivými vetami program opäť vypíše na štandardný výstup analyzovanú vetu a po nej samotné získané extrakcie. Pomocou prepínačov `--binary`, `--format simple` a `--format column` je možné ovplyvniť formát výstupu, avšak druhá z možností, ktorú program využíva predvolene, je najčitateľnejšia a na ďalšie spracovávanie aj najvhodnejšia. Ukážka výstupu v tomto štandardnom formáte je uvedená ako kód 4.2.

```
>In paradise, Eve gave Adam a red apple and he was immediately tempted to eat
    it right there.
In paradise, Eve gave Adam a red apple and he was immediately tempted to eat
    it right there.
0,96 (Eve; gave; Adam; a red apple; L:In paradise)
0,34 (he; was tempted; to eat it right there; T:immediately)
0,27 (he; to eat; it; L:right there)
```

Kód 4.2: Ukážka štandardného formátu výstupu programu Open IE.

Ako je možné vidieť na vyššie uvedenom výstupe, formát každej extrakcie má tvar `<score> (podmet; prísudok; predmet[; doplnok`$_{0..n}$`])`, kde `<score>` ohodnocuje danú extrakciu na základe toho, za akú informačne prínosnú považuje extraktor túto extrakciu, a môže nadobúdať reálne hodnoty z intervalu $< 0, 1 >$. Počet doplnkov sa pohybuje najčastejšie v rozmedzí 0 až 2, pričom v týchto doplnkoch sa často objavujú frázy sprevádzané prefixom `T:` alebo `L:`, keďže extraktor je schopný identifikovať výrazy vyjadrujúce čas, resp. miesto a do zodpovedajúcich častí n-tice ich patrične zaznačiť. Je vhodné tiež upozorniť na to, že formát tohoto výpisu môže ojedinele spôsobiť problémy, ak je na vstupe veta, ktorá obsahuje symbol `;`.

Jedným zo zaujímavých prepínačov je tiež `--ignore-errors`, ktorý podľa autorov programu zabezpečí beh extraktora aj po výskyte výnimky. V praxi sa ukázalo, že používanie programu na spracovanie veľkého počtu viet je bez tohoto prepínača je značne nestabilné, a tak je jeho použitie nevyhnutné. Avšak, aj keď pri jeho použití počet neočakávaných ukončení výrazne klesne, nedá sa povedať, že by im úplne zabránil. Preto je potrebné vytvoriť dodatočné mechanizmy na monitorovanie behu procesu tohoto programu a implementovať vlastný spôsob zotavenia.

4.1.6 Syntaktické analyzátory CoreNLP a BLLIP

CoreNLP je sada nástrojov určených na analýzu a spracovanie prirodzeného jazyka vyvíjaných na Stanfordovej univerzite v Spojených štátoch amerických. Obsahuje prostriedky na rozpoznávanie pomenovaných entít, rezolúciu koreferencie, analýzu sentimentu a ďalšie. Táto sada je vyvinutá v programovacom jazyku Java a licencovaná pod *GNU General Public Licence verzie 3* [36]. Medzi výhody tohoto riešenia patria relatívne vysoká presnosť výstupov a rôzne pokročilé možnosti použitia, nevýhodou je však značná pomalosť. Pre účely tejto práce bolo zaujímavé, že tento nástroj poskytuje väčšinu potrebných nástrojov v jednom, a to menovite najmä vyspelú koreferenciu, rozpoznávanie pomenovaných entít, syntaktický analyzátor s výstupom vo forme zložkových derivačných stromov a označovanie

slovných druhov. Avšak po prvotnom experimentovaní sa zistilo, že pre účely tejto práce tento systém nie pre z výkonnostných dôvodov vhodnou voľbou.

Ako alternatíva k potrebnej syntaktickej analýze bol objavený prehodnocovací syntaktický analyzátor ***BLLIP*** šírený pod licenciou *Apache 2.0*, ktorý je schopný oveľa rýchlejšej analýzy, a dokonca podáva aj presnejšie výsledky ako vyššie uvádzaná alternatíva. Bol vyvinutý na Brownovej univerzite a okrem už spomínaných výhod je jeho ďalšou bezspornou výhodou aj fakt, že ide o riešenie dodávané ako rozšírenie pre jazyk *Python* [37].

4.1.7 Lingvistické knižnice NLTK a Pattern

Ako už bolo spomínané, jeden z dôvodov voľby programovacieho jazyka *Python* je popularita v rámci lingvistickej obce. Pre riešenie tejto práce boli využité možnosti, ktoré prinášajú nasledujúce dve lingvistické knižnice.

NLTK (Natural language toolkit) je sada knižníc vyvinutých pre programovací jazyk *Python* a dátových sád určených na štatistické a symbolické spracovanie prirodzeného jazyka. Bola vytvorená za účelom podporiť výskum a výuku týkajúcu sa lingvistiky, kognitívnych vied, strojového učenia, získavania informácií a ďalších podobných oblastí. Obsahuje nástroje na klasifikáciu, tokenizáciu, určovanie slovných druhov, vytváranie stromových reprezentácií a mnohé ďalšie esenciálne pomôcky pri práci s textom. Táto sada je vydávaná pod licenciou *Apache 2.0* [38]. Medzi jej popredné schopnosti, s ohľadom na zameranie tejto práce, patrí jednoduché rozhranie na prácu s databázou *WordNet*, ktorá bola predstavená v sekcii 3.1.

Pattern je knižnica pre jazyk *Python* určená na dolovanie dát z webu, spracovanie prirodzeného jazyka, strojové učenie či vizualizáciu dát šírená pod *BSD licenciou*. Bola vyvinutá belgickým Výskumným Centrom Počítačovej Lingvistiky a Psycholingvistiky a obsahuje množstvo nástrojov obsiahnutých aj v knižnici *NLTK*, avšak s menej pokročilou funkcionalitou. Toto obmedzenie sa pozitívne prejavuje výrazne vyššou rýchlosťou vykonávania niektorých úloh v porovnaní s vopred popisovanou knižnicou. Jej ďalšou výhodou je manuálne zostavená databáza na časovanie a normalizáciu až 8500 najbežnejších anglických slovies. Mimo anglického jazyka tiež podporuje španielčinu, nemčinu, francúzštinu, taliančinu a holandčinu [39].

4.1.8 Škálovateľné RDF úložisko 4store

RDF úložisko ***4store*** bolo vyvinuté firmou *Garlik*, ktorá sa zaoberá ochranou pred krádežami identity na internete, aby podporila sémantické webové aplikácie. Používala ho ako svoju primárnu platformu po dobu troch rokov, kým ju nenahradila novším a lepším riešením *5store*. Následne toto pôvodné riešenie sprístupnila pod licenciou *GNU General Public Licence verzie 3*.

Za hlavné prednosti tohoto úložiska udáva spoločnosť, ktorá tento systém vyvinula, výkon, škálovateľnosť a stabilitu. Podporuje beh na platformách, ktoré vychádzajú z operačného systému Unix, v klastroch pozostávajúcich z 1 až 32 výpočtových staníc, pričom je schopný dosahovať rýchlosť importu 120 tisíc RDF trojíc za sekundu a časy na spracovanie SPARQL dotazov rádovo v milisekundách. Pri vývoji bolo špeciálne dbané na bezpečnostnú stránku systému [40].

Na prácu so systémom je k dispozícii sada nástrojov, ktoré sa dajú používať cez príkazový riadok. Tieto nástroje je možné z používateľského hľadiska rozdeliť do troch základných kategórií:

- **4s-backend-∗** – nástroje s týmto prefixom slúžia vytváranie, spúšťanie, zastavovanie, mazanie, zálohovanie či nastavovanie databázy na aktuálnom stroji, z ktorého sú spúšťané
- **4s-cluster-∗** – nástroje s týmto prefixom je možné používať na automatizované spravovanie celého klastra, pomocou protokolu *ssh* s využitím autentifikácie cez prihlasovacie kľúče vyvolávajú nástroje prvej skupiny na jednotlivých staniciach
- nástroje určené na prácu s obsahom databázy:
  - **4s-import** – slúži na importovanie obsahu RDF súboru do databázy
  - **4s-update** – umožňuje aktualizovať SPARQL dotazom obsah databázy
  - **4s-query** – týmto príkazom je možné vytvoriť SPARQL dotaz na databázu
  - **4s-import** – slúži na importovanie obsahu RDF súboru do databázy
  - **4s-httpd** – umožňuje spustiť aplikačného démona, ktorý bude na danej stanici schopný naslúchať na zadanom porte a prostredníctvom protokolu *http* zodpovedávať SPARQL dotazy

Dôležitou vlastnosťou databázy *4store* je tzv. *soft limit*, ktorého úlohou je zabrániť spotrebovaniu veľkého množstva zdrojov na nájdenie výsledkov a zvýšiť rýchlosť vyhodnotenia dotazu za cenu obmedzenia počtu nájdených výsledkov. Toto však môže mať za následok nevyhľadanie výsledkov, ktoré sa ale v databáze nachádzajú. Hodnota tohoto limitu je štandardne nastavená na 1000, je ju však možné ľubovoľne navýšiť alebo aj úplne vypnúť prostredníctvom hodnoty -1. Takéto konanie ale môže mať za následok neprijateľne vysoké spomalenie rýchlosti spracovávania dotazov nad obsahom databázy.

4.1.9 Flask, Bootstrap a AngularJS

Pri vyberaní technológií určených na vytvorenie webového rozhrania na prácu s vyextrahovanými dátami bolo sledovaných viacero kritérií. Najdôležitejšie bolo vybrať technológie, ktoré umožňujú rýchly vývoj, fungujú spoľahlivo a technologicky bezproblémovo zapadajú do zvyšku riešenia. Taktiež bola zohľadnená dostupnosť dokumentácie, aktívnosť komunity a aktuálne trendy v oblasti webových technológií, aby vytvárané riešenie malo potenciál na ľahké nasadenie a udržiavanie.

Flask je mikroframework napísaný v jazyku *Python* určený na tvorbu webových aplikácií. Vychádza z WSGI knižnice *Werkzeug*, využíva šablónovací systém *Jinja2* a je šírený pod *BSD licenciou* [41]. Jeho minimalistické, ale ľahko rozšíriteľné jadro poskytuje spoľahlivé a univerzálne základy na vytvorenie robustných aplikácií, a tak je voľbou viacerých popredných internetových spoločností ako napríklad *Pinterest* [42].

Bootstrap je voľne dostupná sada nástrojov na vytváranie webových rozhraní a bol pôvodne vyvinutý pre interné účely spoločnosti *Twitter*. Skladá sa z HTML a CSS šablón rôznych tabuľkových, formulárových, tlačidlových, typografických či ďalších komponent, ktoré je možno ľahko kombinovať za účelom rýchleho navrhovania webovej stránky. Pri používaní tohoto frameworku je implicitne vyriešený problém s podporou zobrazenia na malých zariadeniach, keďže responzívny mobilný dizajn je aktuálne vo verzii 3 jedna z jej základných filozofií. Okrem už spomínanej možnosti pohodlného a rýchleho vývoja či responzivity, patrí medzi jeho bezsporné výhody aj kompatibilita s najnovšími verziami všetkých bežne používaných webových prehliadačov, čo odstieňuje používateľa tejto technológie od mnohých problémov a umožňuje mu pri vývoji sústrediť sa na práve na dosahovanie zámeru [43].

AngularJS je voľne dostupný framework napísaný v jazyku *JavaScript* a udržiavaný spoločnosťou *Google* s cieľom zjednodušiť vývoj jednostránkových webových aplikácií. Je postavený na návrhovom vzore *model-pohľad-radič (MVC, Model-View-Controller)*, ktorý rozdeľuje funkčnosť aplikácie medzi 3 rovnomenné komponenty [44, 45]:

- **model** – reprezentuje dáta aplikácie, s ktorými sa pracuje
- **pohľad** – transformuje dáta z modelu do vhodného používateľského rozhrania
- **radič** – reaguje na udalosti a zaisťuje zodpovedné zmeny v zvyšných komponentách

Bezproblémová integrácia posledných dvoch menovaných technológií je možné vďaka komunitnému projektu ***UI Bootstrap***, ktorý prináša šablóny pre interaktívne *Bootstrap* komponenty prepísané tak, aby ich bolo možné intuitívne využívať z prostredia *AngularJS* [46].

Všetky nástroje na vytvorenie front-endu sú šírené pod *MIT licenciou*.

4.2 Prostredie riešenia

Systém, ktorý je predmetom tejto práce, bol navrhovaný v prostredí **výpočtového klastra** poskytnutého *Výskumnou skupinou znalostných technológií FIT VUT v Brne*. K dispozícii je niekoľko desiatok staníc, z ktorých na distribuované výpočty v rámci riešenia tejto práce postačoval len zlomok. Ide o stanice vybavené procesorom *Intel® Xeon® CPU E5-2630* s frekvenciou 2.30GHz a 64GB operačnou pamäťou fungujúcich pod operačným systémom *Ubuntu 14.04.1 LTS*. Samozrejme, ide o stroje zdieľané s inými používateľmi, a tak som nemal k dispozícii ich plný výpočtový výkon.

4.3 Implementácia komponent navrhovaného systému

V tejto sekcii je popis implementácie nášho systému korešpondujúci s návrhom predstaveným v kapitole 3. Pri implementácii boli využité technológie popísané v sekcii 4.1 tejto kapitoly.

4.3.1 Čistenie vstupných dát

Na zvolený korpus spomínaný v sekcii 3.1, ktorý je tvorený obsahom anglickej mutácie encyklopédie *Wikipédia*, bol na odstránenie všetkých nepotrebných značiek použitý ***XML parser*** vyvinutý *Ing. Markom Schmidtom* a *Bc. Davidom Smejkalom*. Keďže bol vytvorený v rokoch 2007 až 2008 na analýzu staršej verzie formátu výpisov poskytovaných organizáciou *Wikimedia*, bolo potrebné niektoré jeho časti aktualizovať a doplniť podporu nových značiek.

Po transformovaní formátu *XML* do čistého textu boli v ďalšom kroku identifikované určité typy článkov, ktoré prinášajú veľmi nízku, resp. žiadnu informačnú hodnotu, a slúžia iba na komunitné účely pre ľudí zaoberajúcich sa vytváraním obsahu *Wikipédie*. Názvy týchto článkov začínajú prefixami *Category:*, *File:*, *Help:*, *Template:*, *Portal:*, *Wikipedia:* a *MediaWiki:*. Tieto články boli z korpusu odstránené.

4.3.2 Rozdelenie do výpočtového klastra a predspracovanie korpusu

Pripravený korpus je možné ďalej rovnomerne rozdeliť a distribuovať na stanice výpočtového klastra pomocou na tento účel vytvorených skriptov.

Cieľom ďalšieho kroku bola snaha o zjednoznačnenie viet pomocou koreferencie, aby po samotnej extrakcii nevznikali vyextrahované n-tice obsahujúce nepriame odkazy na entity, najčastejšie vyjadrené pomocou zámen. Na tieto účely sa experimentovalo s niekoľkými systémami, z ktorých najpresnejšie výsledky priniesol systém *CoreNLP*. Bohužiaľ, tento systém využíva veľmi podrobnú a pokročilú analýzu viet, a tak je nielen veľmi pomalý, ale tiež v ňom nie je možné spracovávať rozsiahle texty. Pri jeho prvotnom používaní v závislosti od špecifickej skladby niektorých viet nebol občas schopný vyhodnotiť krátke paragrafy v čase niekoľkých desiatok sekúnd. Preto táto možnosť pri použití so zamýšľaným systémom, ktorý má byť schopný škálovať na korpus o miliónoch či dokonca až miliardách článkov, neprichádzal do úvahy.

Nakoniec bol na úlohu koreferencie zvolený nástroj *Decipher NER*, ktorý fungoval na všetkých strojoch dostupného výpočtového klastra ako služba, s ktorou je možné komunikovať pomocou aplikačného rozhrania *SEC API* popísaného v [47] prostredníctvom protokolu *JSON*. Každý článok je odoslaný na koreferenciu celý ako súvislý blok textu, aby sa predišlo strate kontextu pri vyhodnocovaní tejto koreferencie v prípadoch, že by bol text rozdelený na menšie časti. Pri implementácii bolo overené, že systém *Decipher NER* zvláda vďaka svojmu jednoprechodovému algoritmu vyhodnotiť aj tie najrozsiahlejšie články, ktoré boli v dátovej sade *Wikipédie* obsiahnuté.

4.3.3 Extrakcia faktov

Na úlohu extrakcie faktov bol zvolený systém *Open IE* popísaný v podsekcii 4.1.5 na začiatku tejto kapitoly. Tento nástroj je však napísaný v jazyku Java, a tak bolo potrebné vyriešiť spôsob prepojenia s ďalšími časťami nášho systému. V rámci vývoja bolo vystriedaných viacero prístupov zahŕňajúcich naprogramovanie servera komunikujúceho vo formáte *JSON*, ktorý prijímal požiadavky s textami na extrakciu a v odpovediach odosielal vyextrahované n-tice, avšak toto riešenie sa neukázalo ako príliš efektívne a zaťažovalo sieť.

Spúšťanie tohoto nástroja na každý text samostatne tiež nebolo najvhodnejšie riešenie, keďže proces jeho inicializácie trvá zhruba 2 minúty a nedokáže efektívne spracovať príliš rozsiahle texty. Po vyriešení niektorých problémov spôsobených nevhodne navrhnutým spôsobom interaktívnej komunikácie tohoto nástroja s používateľom, bolo zrealizované konečné riešenie spočívajúce v jednorazovej invokácii procesu tohoto extraktora prostredníctvom *Pythonu*, pripojeniu štandardného vstupu a výstupu pomocou zreťazení, ktoré poskytuje operačný systém *Linux* a vhodnom komunikovaní prostredníctvom nadviazaných štandardných prúdov pre dosiahnutie želaného zámeru. Na urýchlenie spracovania sú dlhšie články spracovávané po menších častiach.

Keďže sa predpokladalo, že samotná extrakcia bude patriť medzi výpočtovo najnáročnejšie časti nášho systému, toto riešenie umožňuje púšťanie jedného či viacerých extraktorov na ľubovoľnom počte staníc na dosiahnutie rýchlejšieho procesu extrakcie.

4.3.4 Čistenie a analýza vyextrahovaných faktov

Po vyextrahovaní sady n-tíc z textu každého článku je možné jednotlivé n-tice ďalej analyzovať podľa navrhnutých postupov v sekciách 3.6 a 3.7 predchádzajúcej kapitoly s návrhom systému.

Prvým z krokov bezprostredne po extrakcii je však prečistenie získaných extrakcií, pretože ďalšie časti systému počítajú s jednoduchými n-ticami skladajúcimi sa z 3 alebo 4 častí. Niektoré extrakcie pochádzajúce z *Open IE* však túto dĺžku presahujú alebo obsahujú zložitejší tvar rozvíjajúci predchádzajúce extrakcie s referenciami na ne. U n-tíc s dĺžkou väčšou ako 4 časti boli preto nadbytočné časti pričlenené k štvrtej časti a občasné zložité n-tice vyskytujúce sa v neštandardnom tvare boli vynechané úplne.

Ďalej bol implementovaný algoritmus 1 v podsekcii 3.6.1 ako samostatný krok, a to z dôvodu možnosti voliteľného vypnutia kvôli problémom s rýchlosťou syntaktickej analýzy fráz diskutovanej v podsekcii 4.4.2. Úlohou tohoto kroku je rozgenerovanie zložitých fráz na jednoduchšie aby bola možná ľahšia identifikácia súvislostí v databáze. Navrhnutý algoritmus sa pri svojej funkčnosti spolieha na vstup vo forme zložkového derivačného stromu, pričom na túto úlohu bol najprv použitý nástroj *CoreNLP* a neskôr nahradený syntaktickým analyzátorom *BLLIP*.

Ako ďalší krok bola popísaná normalizácia predikátu navrhnutá v algoritme 2 v podsekcii 3.6.2 za použitia knižnice *Pattern* a ďalej určovanie typu frázy popísané v podsekcii 3.7.1. Toto určovanie typu frázy využíva vo svojej implementácii viacero prístupov a technológií. Podporované typy fráz sú *niekto*, *niečo*, *niekedy*, *niekde*, pričom na zaradenie do prvých dvoch je využitý ako základ navrhnutý algoritmus 3. Na jeho realizáciu je na určenie slovných druhov je využitá knižnica *Pattern* a na prácu s databázou *WordNet* knižnica *NLTK*. Keďže niektoré mená nie sú obsiahnuté v databáze *WordNet*, je tiež opätovne využitý nástroj *Decipher NER*, ktorý okrem koreferencie dokáže identifikovať známe entity vystupujúce v texte. Opätovne je využitý preto, že medzi krokom koreferencie a týmto krokom je krok extrakcie, pri ktorom sa nepodarilo vymyslieť spôsob ako získané informácie nadväzujúce na časti textu preniesť cez spracovanie systémom *Open IE*. Celkovo kombinácia *WordNetu* a *Decipher NER* priniesla na rozdeľovanie typov na *niekto* a *niečo* lepšie výsledky ako použitie iba jedného z týchto nástrojov.

Keďže systém *Decipher NER* zvláda aj identifikáciu lokalít prípadne niektorých časových údajov, bolo pôvodne zamýšľané využiť jeho možnosti aj na identifikáciu typov *niekde* a *niekedy*. Avšak ako bolo neskôr zistené, systém *Open IE* dokáže tieto typy identifikovať sám a univerzálnejšie.

Všetky novozískané informácie sú serializované do formátu *JSON* a pripojené na koniec každého riadku s extrakciou.

4.3.5 Uloženie do databázy

Ako databáza na uloženie trojíc bolo zvolené úložisko *4store*, ktoré, ako už bolo predstavené, podporuje jazyk RDF. Preto je potrebné všetky extrakcie s informáciami získanými počas analýzy konvertovať do niektorého z formátov zápisu tohoto jazyka. Na tento účel bola využitá pythonovská knižnica *RDFLib* umožňujúca vytvorenie dočasného RDF úložiska v operačnej pamäti, ktoré je možné naplniť trojicami a následne serializovať do ľubovoľného RDF formátu. V našom prípade bol zvolený formát *Turtle*, keďže ide o najkompaktnejší zápis, ktorý je zároveň ľahko čitateľný v prípade potrebnej manuálnej kontroly výstupu.

Objekty, ktoré sa ukladajú do databázy sú modelované podľa návrhu na obrázku 3.4, pričom najväčší ako problém sa ukázala práca s identifikátormi objektov. Pred každým vytváraním nových objektov je potrebné skontrolovať, či je ich naozaj potrebné vytvoriť a nie je iba možné využiť už existujúci identifikátor. Na riešenie tohoto problému bolo v priebehu vývoja implementovaných viacero prístupov, ktoré sú ďalej spolu s odôvodneniami predstavené ďalej v podsekcii 4.4.3. Konečná implementácia umožňuje zapnutie a použitie

ľubovoľného z menovaných prístupov, pričom na samostatné spracovanie jedného článku je výhodnejšie použitie popisovaného naivného prístupu, ktorý zároveň výsledok automaticky vloží do databázy cez *http* protokol a na spracovanie rozsiahlej sady článkov je nutné použiť posledný prístup využívajúci vyrovnávaciu pamäť s využitím perzistentného úložiska *ZODB*. V druhom prípade sú však iba vytvorené súbory s obsahom vo formáte *Turtle*, ktoré je potrebné manuálne nahrať do databázy *4store* nástrojom *4s-import*. Každý z týchto súborov je závislý na obsahu predchádzajúcich súborov, a tak pri nahrávaní n-tého súboru sa predpokladá, že jemu predchádzajúce súbory sú už v databáze nahraté.

4.3.6 Používateľské rozhranie

Implementácia používateľského rozhrania sa skladá z dvoch častí – *back-end* komunikuje s databázou a *front-end* zabezpečuje interakciu s používateľom a prezentuje získané výsledky. Na implementáciu back-endu bola využitá knižnica *Flask*, ktorá vytvorí server poskytujúci súbory front-endu klientskemu prehliadaču a prijíma jeho dotazy, ktoré preposiela službe *4s-httpd* databázy *4store* prostredníctvom protokolu *http*. Obdržané výsledky posiela naspäť klientskemu front-endu spolu s nameraným časom prehľadávania databázy.

Na implementáciu vzhľadu front-endu boli využité šablóny komponent poskytované projektom *Bootstrap* a implementácia dynamických častí bola realizovaná prostredníctvom technológie *AngularJS*. Konečný návrh umožňuje fulltextové vyhľadávanie v jednotlivých častiach n-tíc, vyhľadávanie podľa typu frázy alebo normalizovaného tvaru predikátu. Používateľ si môže vybrať spomedzi prednastavených filtrov alebo sám zostaviť dotaz prostredníctvom rozhrania umožňujúceho pridať ľubovoľný počet filtrovacích trojíc určujúcich pravidlá vyhľadávania, pričom môže používať špeciálnu syntax na uvedenie premenných, ktoré majú vystupovať naprieč jednotlivými trojicami a predstavujú hľadanú odpoveď. Po spustení vyhľadávania sú nastavené parametre konvertované na SPARQL dotaz, ktorý je odoslaný back-endu a je ho možné zobraziť, prípadne upraviť aj v rozhraní front-endu. Po obdržaní odpovede je zobrazený nielen čas hľadania a objavené riešenia vystupujúcich premenných, ale tiež všetky nájdené kombinácie záznamov, ktoré sa podieľajú na riešení spolu s možnosťami zobraziť ich jednotlivé detaily. Ukážka vzhľadu front-endu je predstavená na obrázku 5.10 uvedeného v ďalšej kapitole.

4.3.7 Skripty na ovládanie

Keďže ide o riešenie vytvorené ako viacero samostatne implementovaných krokov, ktorých skripty sa používajú v špecifickom poradí a na všeobecne ľubovoľnom počte staníc výpočtového klastra, bola veľká časť vývoja venovaná tiež vytvoreniu skriptov automatizujúcich a zjednodušujúcich spúšťanie a prácu s týmto riešením. Tieto skripty boli napísané v jazyku *Bash* a slúžia na spúšťanie a zastavovanie jednotlivých komponent systému alebo systému ako celku. Taktiež umožňujú automatizované vytvorenie novej databázy *4store*, monitorovanie behu nástroja *Open IE* a jeho prípadné opätovné spúšťanie, alebo napríklad aj skomprimovanie priečinkov s článkami pre možnosť ich presunutia medzi stanicami. Vysvetlenie funkčnosti a popis použitia jednotlivých skriptov sú popísané v prílohe A.

4.4 Problémy pri riešení

Ako už bolo naznačené v predchádzajúcich častiach tejto kapitoly, v priebehu implementácie a testovania nášho systému sa objavili viaceré problémy, na ktoré bolo potrebné

hľadať riešenia. V tejto sekcii sú tieto problémy uvedené spolu s diskutovanými zvolenými riešeniami.

4.4.1 Nestabilné komponenty systému

Na záver podsekcie 4.1.5 bol spomínaný jeden z problémov, ktorý sa objavil hneď v úvodných fázach vývoja, a bolo ním zistenie, že systém *Open IE* je nepredvídateľne nestabilný, a to aj v prípade použitia prepínača `--ignore-errors`. Z tohoto dôvodu bola, neplánovane, prevažná časť riešenia kroku extrakcie venovaná najmä implementácii záložných metód monitorujúcich beh extraktora, jeho zotavenie a možnosť ďalšieho nadviazania v procese extrakcie.

V neskorších fázach vývoja sa tiež začali objavovať občasné výpadky služieb na koreferenciu, prípadne boli vykonané priebežné zdokonaľovania nad skriptami, ktorých úlohou bolo napríklad rozgenerovávať frázy alebo určovať ich typ, a tak bolo opakované spúšťanie celého navrhnutého extrakčného systému po každom vyskytnutom probléme alebo modifikácii neprijateľné. Rovnako nebolo možné jednoducho premiestňovať medzi stanicami výpočtového klastra úlohy určené na spracovanie podľa aktuálnej vyťaženosti danej stanice, prípadne dočasne prerušiť určitý krok spracovania a následne ho jednoduchým spôsobom obnoviť, pretože sa pri pôvodnom návrhu pracovalo so sekvenčným spúšťaním krokov a v každom kroku s jedným vstupným súborom obsahujúcim skonkatenované vstupy predchádzajúceho kroku a jedným výstupným súborom, do ktorého daný krok zapisoval.

Vyššie uvedené dôvody viedli k návrhu nového unifikovaného prístupu k spracovávaniu inšpirovaného dielenským spôsobom vývoja. Bola vytvorená organizovaná štruktúra priečinkov, kde každý priečinok prislúcha jednému kroku extrakčného systému. V prvom priečinku sú na počiatku uložené jednotlivé články v samostatných textových súboroch. Následne každý krok extrakcie pristupuje k priečinku predchádzajúceho kroku, po jednom spracúva v ňom obsiahnuté súbory a zapisuje do samostatných textových súborov v priečinku, ktorý mu prislúcha.

Ďalej bol navrhnutý systém, ktorý za pomoci rozšírenia ***Watchdog*** [48] pre jazyk *Python* dokáže monitorovať zmeny v priečinkoch a v prípade, že bol vytvorený nový súbor, pridá ho na zoznam súborov čakajúcich na spracovanie v danom kroku. Skript každého kroku bol následne prepísaný tak, aby fungoval ako služba, čakajúca na pozadí a monitorujúca priečinok predchádzajúceho kroku. V prípade, že dôjde k neplánovanému ukončeniu skriptu v nejakom kroku extrakčného systému, sú samozrejme informácie o súboroch, ktoré treba spracovať, stratené.

Preto bol tiež vytvorený skript, ktorý dokáže modifikovať súbory obsiahnuté v nejakom priečinku aby upozornil monitorovaciu službu na ich existenciu, a dokáže tak zabezpečiť správnu inicializáciu zoznamu súborov určených na spracovanie, ak boli vytvorené ešte pred spustením monitorovacej služby. Tento skript však modifikuje iba potrebné súbory, a to na základe viacerých faktorov vzhľadom na obsah priečinkov v krokoch $n-1$ a n:

- ak daný súbor v priečinku kroku $n-1$ existuje, ale v priečinku kroku n zodpovedajúci súbor neexistuje, bude v prvom uvedenom priečinku modifikovaný
- ak daný súbor v priečinku kroku $n-1$ má novší dátum poslednej modifikácie ako zodpovedajúci súbor v priečinku kroku n, bude v prvom uvedenom priečinku opätovne modifikovaný

Ako výsledok riešenia tohoto problému tak vznikol zjednotený systém spracovávania, ktorý je nielen robustnejší proti neočakávaným poruchám jeho jednotlivých častí, ale zároveň

umožňuje ľahko zálohovať výsledky jednotlivých medzikrokov, opätovne ich prepočítať alebo ich presúvať medzi jednotlivými stanicami v prípade potreby.

4.4.2 Pomalá syntaktická analýza

Ďalším z problémov, ktoré sa v priebehu vývoja objavili, bola pomalá syntaktická analýza fráz. Aj keď po zmene nástroja *CoreNLP* syntaktickým analyzátorom *BLLIP* nastalo významné urýchlenie spracovávania, pri testovaní na veľkých sadách dát sa ukázalo, že zapnutá podpora rozgenerovávania priraďovacích fráz nevhodne obmedzuje rýchlosť spracovania od prijatia vstupného textu až po uloženie extrakcií do databázy. Keďže úlohou tohoto samotného kroku je iba vytvorenie ďalších extrakcií, ktorých účelom je zlepšiť presnosť poslednej časti implementovaného riešenia – QA systému, ale nie je pre fungovanie nutný, bola pridaná možnosť tento krok vypnúť. Prakticky je toto vypnutie realizované iba skopírovaním obsahu vstupu na výstup, teda ako by skript postupoval aj pri frázach, ktoré žiadne priraďovacie spojky neobsahujú.

4.4.3 Efektívne vkladanie dát do RDF databázy

Aj keď boli počas vývoja odladené mnohé problémy, ktoré sa v priebehu vytvárania systému objavili, samotné testovanie funkčnosti poväčšine prebiehalo na malej vzorke článkov, a tak sa až do záverečných pokusov o spustenie spracovania na veľké množstvo článkov nedal predpokladať problém, ktorý sa eventuálne objavil. Vzniknutým problémom bola rýchlosť vkladania dát do RDF databázy, ktorá bola spôsobená faktom, že pri spracovávaní ľubovoľnej extrakcie je potrebné pre každú časť jej n-tice overiť, či sa daná fráza alebo dokonca celá n-tica sa už v databáze náhodou nenachádza.

Naivný prístup

V rámci pôvodnej implementácie bol použitý prístup, pri ktorom pred vytváraním každého nového objektu v databáze bolo pomocou jednoduchých dotazov kontrolované, či sa daný objekt sa v databáze už náhodou nenachádza, a za predpokladu, že ešte neexistoval, bol vytvorený nový objekt s unikátnym identifikátorom, ktorý bol následne do databázy vložený. Ak však takýto objekt už v databáze existoval, bol pri ďalšej práci využívaný jeho identifikátor a nebolo potrebné generovať duplicitný objekt.

Pri malom objeme testovacích dát v úložisku *4store* sa tento problém neprejavoval, avšak po vložení prvého milióna trojíc pochádzajúceho zo zhruba dvoch tisícov článkov bola doba spracovania každej požiadavky pri paralelnom dotazovaní piatich staníc neprimerane vysoká. Každý dotaz, ktorého úlohou bolo overiť existenciu celej trojice alebo nejakej jej časti, sa spracovával v rádoch stoviek milisekúnd, pričom na vyhodnotenie jednej extrakcie boli potrebné minimálne 4 dotazy, a tak celkový čas potrebný na spracovanie každého ďalšieho článku rapídne stúpal na úroveň desiatok sekúnd a pri rozsiahlejších článkov až minút.

Vyrovnávacia pamäť pri paralelnom vkladaní

Prvým pokusom riešenie vzniknutej situácie bola implementácia lokálnej vyrovnávacej pamäte pre každú stanicu. V prípade, že stanica vytvorila nový objekt, uložila si do tejto pamäte potrebné informácie, pričom pri ďalšom hľadaní odpovede na existenciu určitého objektu najprv kontrolovala obsah tejto pomocnej pamäte a až následne zasielala požiadavku

na samotnú RDF databázu. V prípade pozitívnej odpovede na existenciu objektu v databáze si samozrejme túto informáciu taktiež uložila.

V praxi sa ukázalo, že účinok tohoto riešenia bol zanedbateľný, keďže pri obsahu databázy na úrovni iba pár tisícov článkov bola väčšina extrakcií, ktoré sa spracovávali, čiastočne alebo úplne unikátnych, a tak táto vyrovnávacia pamäť neplnila želaný účel.

Ďalším pokusom o zlepšenie tohoto nápadu bolo vytvorenie mechanizmu, ktorý pomocou špeciálne zostrojeného SPARQL dotazu vedel získať všetky potrebné údaje na vytvorenie vyrovnávacej pamäte reflektujúcej aktuálny obsah celej databázy. Na jednej zo staníc výpočtového klastra následne bežala služba, ktorej úlohou bolo v hodinových intervaloch vytvárať na zdieľanom úložisku súbory s globálnou vyrovnávacou pamäťou, ktorá reprezentovala aktuálny stav databázy pre jednotlivé typy objektov, a všetky stanice boli následne schopné tieto informácie okamžite načítať. Toto riešenie vychádzalo z hypotézy, že pri paralelnom napĺňaní databázy mohol do tejto databázy vložiť požadovaný objekt iný zo serverov výpočtového klastra, ale aj tak z aktuálnej stanice bolo potrebné vykonať jednu kontrolu aby sa mohla naplniť jej vyrovnávacia pamäť. Vďaka priebežnej aktualizácii obsahu vyrovnávacej pamäte však bolo potrebné kontrolovať existenciu objektov, ktoré mohli byť pridané iba v uplynulej hodine.

Aj keď použitie tejto metódy už malo citeľnejší vplyv na rýchlosť vkladania, stále išlo o veľmi malé zlepšenie, ktoré neumožňovalo naplnenie databázy väčším množstvom článkov v rozumnej dobe.

Perzistentná vyrovnávacia pamäť na dedikovanom stroji

Po preskúmavaní ďalších možností bola vytvorená nová hypotéza, ktorá predpokladala, že konverzia extrakcií na RDF trojice spustená iba na jednom stroji by mohla v konečnom dôsledku fungovať oveľa efektívnejšie ako pokusy o paralelné spracovávanie tohoto kroku vo výpočtovom klastri. Výhodou behu konverzie na jedinej stanici je, že táto stanica by mala istotu, že v databáze neexistuje žiadny objekt obdobný tomu, ktorý má sama záujem vytvárať, takže by sa predišlo zbytočnému zasielaniu dotazov na existenciu. Veľmi rýchlo bolo potvrdené, že táto hypotéza bola správna a priniesla rapídne urýchlenie napĺňania databázy novo spracovanými článkami.

Avšak, keďže bolo stále potrebné využívať vyrovnávaciu pamäť a tá bola riešená pomocou pythonovského dátového typu slovník, ktorého obsah bol udržiavaný v operačnej pamäti stanice, počet spracovaných článkov bol v priamej korelácii s množstvom spotrebovanej operačnej pamäte. Bolo preto potrebné nájsť spôsob na udržiavanie tejto vyrovnávacej pamäti priamo na disku aby nebolo možné vyčerpať množstvo operačnej pamäte, ale zároveň muselo hľadané riešenie vedieť rýchlo prehľadávať uložené dáta.

Finálne riešenie pre túto prácu nakoniec prinieslo objavenie natívnej objektovej databázy ***ZODB (Objektová Databáza Zope)*** vo forme knižnice vytvorenej pre jazyk *Python*, ktorá spĺňa vyššie požadované vlastnosti a poskytuje transparentne používateľnú perzistentnú databázu vo forme súborov na disku. Je vyvíjaná spoločnosťou *Zope* a šírená pod licenciou *Zope Public License* [49]. Jej najväčšou výhodou bolo, že požadovala minimálne úpravy v existujúcom kóde na jej úspešnú integráciu do už existujúceho riešenia. Dáta však ukladá do perzistentnej pamäte na báze potvrdzovania transakcií, pričom si vytvára zálohy medzikrokov, čo sa ukázalo ako ďalší obmedzujúci faktor. Napriek tomu umožnila skonvertovanie 50-násobne väčšieho množstva článkov do RDF formátu ako pôvodný naivný prístup popísaný ako prvý v tejto podsekcii.

Kapitola 5

Experimentovanie na dátach anglickej Wikipédie

Ako už bolo uvádzané v predchádzajúcich kapitolách, návrh a implementácia extrakčného systému tejto práce počíta primárne s podporou fungovania na obsahu poskytnutom anglickou mutáciou internetovej encyklopédie *Wikipédia*. V tejto kapitole sú preto uvedené výsledky a zistenia, ktoré prinieslo použitie systému na už spomínanú dátovú sadu.

Informácie uvedené v tejto kapitole popisujú výsledky získané pri použití výpisu anglickej *Wikipédie* stiahnutého dňa 20. apríla 2015 z oficiálnych stránok projektu *Wikimedia*. Tento výpis bol poskytnutý vo formáte *XML*, má veľkosť približne 52GB a obsahuje takmer 5 miliónov článkov.

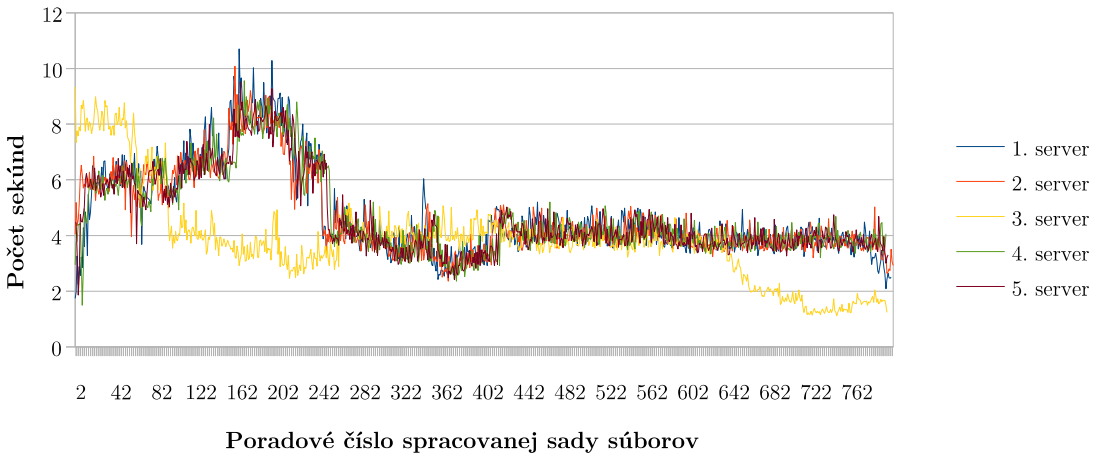
5.1 Štatistiky extrakcie

Prvým krokom, ktorý bolo potrebné vykonať bola konverzia získaného výpisu na čistý text obsahujúci iba užitočné články tak, ako je popisované podsekcii 4.3.1 implementačnej kapitoly. Predpokladalo sa, že tento krok patrí medzi výpočtovo nenáročné, ale ukázalo sa, že kompletné vyextrahovanie čistých textov článkov trvalo jednej výpočtovej stanici takmer 24 hodín. Ako výsledok čistenia vznikol súbor obsahujúci očakávané texty článkov bez formátovacích značiek, ktorého veľkosť bola 12GB. Dá sa predpokladať, že vcelku nadmerná dĺžka čistenia bola spôsobená neoptimalizovanými regulárnymi výrazmi, ktoré skript *XML parser* využíva, ale keďže išlo o jednorazovú akciu, čas strávený touto redukciou veľkosti zdrojových dát o približne 77% neprekážal.

Ďalším krokom bolo vyčlenenie prvých 200-tisíc článkov, ktoré boli rozdistribuované rovnomerne medzi 5 serverov. V ďalších popisovaných krokoch teda každý zo serverov spracovával presne 40-tisíc článkov. Keďže v procese extrakcie sa občas objavili problémy, ktoré bolo potrebné riešiť na zvýšenie stability systému, bolo zopárkrát nutné systém zastavovať a neskôr spúšťať od bodov prerušenia. Prípadne niektoré z využívaných externých systémov neboli občas schopné spracovať niektoré zo vstupov, čo spôsobilo, že ojedinelé články boli v niektorých krokoch vynechané. Celkový počet vynechaných článkov, ktoré neprešli celým zreťazením procesov z rôznych príčin, je pár stoviek a konkrétne hodnoty je možné vidieť v prílohe B.

5.1.1 Analýza rýchlosti jednotlivých krokov systému

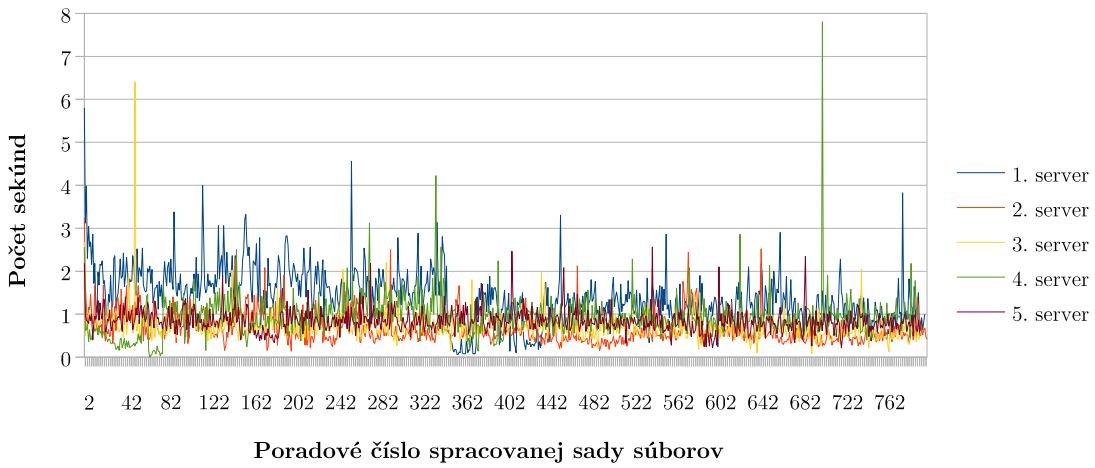
Ako bolo uvedené v podsekcii 4.4.3 kapitoly 4, za jeden z najzávažnejších problémov, ktorým bolo potrebné čeliť počas záverečného testovania bola rýchlosť konverzie n-tíc do RDF formátu, prostredníctvom ktorého bolo možné tieto dáta ukladať do databázy *4store*. Vytvorený systém založený na sledovaní zmien v priečinkoch poskytol podrobné záznamy o rýchlosti spracovania jednotlivých súborov, ktoré slúžili ako vstup pre vytvorenie grafov rýchlosti spracovania uvedených ďalej v tejto podsekcii. Časové razítka na záznamoch však uvádzajú ako najmenšiu časovú jednotku sekundu a medzi dĺžkou spracovania jednotlivých článkov nastávajú v závislosti na ich dĺžke občas priepastné rozdiely. Preto bol celkový počet článkov rozdelený na sady po 50-tich kusoch, u ktorých bola vždy vypočítaná priemerná dĺžka spracovania jedného článku vzhľadom na celkový čas spracovávania danej sady. Toto umožnilo nielen odstrániť problém určovania rýchlosti súborov, ktorých spracovanie bolo na úrovni milisekúnd, ale tiež pomohlo k sprehľadneniu grafov a vyplynutiu určitých konštantných trendov. Podrobnosti k spracovaniu rýchlostí na jednotlivých serveroch sú uvedené v tabuľke B.1 v prílohe B.



Obr. 5.1: Graf zobrazujúci priemerné trvanie koreferencie jedného článku vypočítané na sade tvorenej vždy päťdesiatimi článkami.

Prvý z grafov na obrázku 5.1 zachytáva rýchlosť spracovania koreferencie koreferenčnej služby spustenej na jedinej stanici a predstavuje celkovo 53 hodín spracovávania. Vidíme, že štyri z piatich grafov majú prakticky identický priebeh a iba priebeh rýchlosti jedného z nich sa výrazne odlišuje. V skutočnosti bol 3. server spustený o pol dňa neskôr ako zvyšné štyri servery, a po pozornejšom preskúmaní je možné vidieť, že jeho priebeh tiež kopíruje priebeh zvyšku skupiny. Priemerná doba na koreferenciu jedného článku pri paralelnom spracovávaní dvojnásobných požiadaviek z 5 serverov je zhruba 4 sekundy. Dve súčasné požiadavky sú z dôvodu využívania tejto služby aj na paralelne bežiaci krok analýzy fráz. Nárast dĺžky spracovania nastal v priebehu dňa okolo 16-tej hodiny poobede, keď bola stanica poskytujúca koreferenciu pravdepodobne vyťažená aj inou úlohou, a to na priemerných 8 až 10 sekúnd na článok. Po ukončení koreferencie na štyroch staniciach vidíme zrýchlenie koreferencie na dobiehajúcej stanici na úrovni priemerne 1,5 sekundy.

Na grafe na obrázku 5.2 môžeme vidieť rýchlosť extrakcie n-tíc faktov z jednotlivých článkov. Je zrejmé, že jeden z pôvodných predpokladov, že tento krok bude patriť k výpoč-



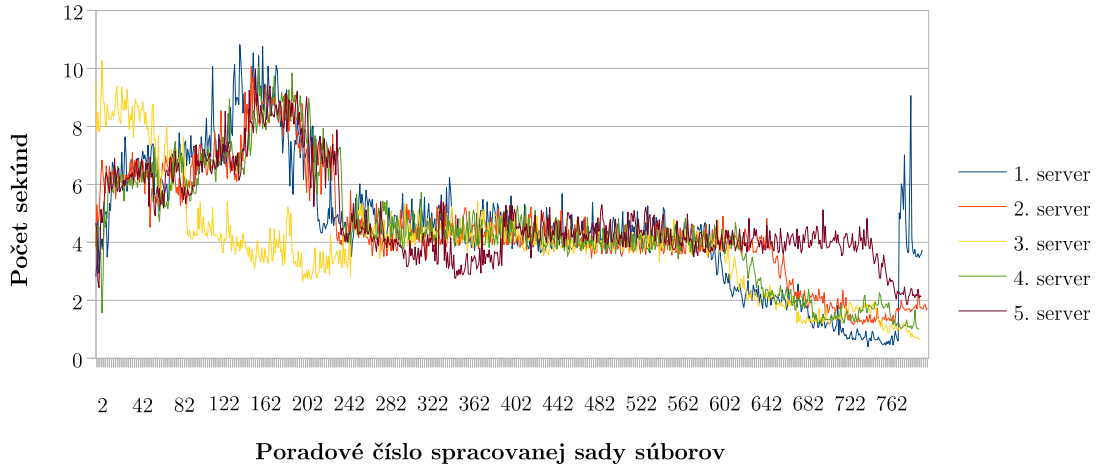
Obr. 5.2: Graf zobrazujúci priemerné trvanie extrakcie jedného článku vypočítané na sade tvorenej vždy päťdesiatimi článkami.

tovo najnáročnejším, sa nepotvrdil. Priemerná dĺžka extrakcie jedného článku bola až na občasné výchylky približne 1 sekunda, čo sa dá považovať, v porovnaní s ostatnými krokmi, za veľmi rýchle spracovávanie.

Krok rozgenerovávania nadrozmerných fráz s priraďovacími spojkami bol z tohoto spracovávania vynechaný, keďže ako už bolo zdôvodnené v podsekcii 4.4.2, syntaktické analyzátory využívané v tomto kroku boli niekoľko desiatok násobne pomalšie ako služby využité v iných krokoch. Ďalší krok zachytáva graf čistenia a analýzy n-tíc, ktorý kopíruje trend grafu 5.1, a to z dôvodu, že využíval na svoju funkciu rovnakú službu ako koreferencia, avšak tentokrát na rozpoznávanie pomenovaných entít. Je však posunutý o približne pol sekundu nahor z dôvodu dodatočnej analýzy spracovaných fráz pomocou ďalších lingvistických nástrojov na zaradenie v sieti *WordNet* a normalizáciu predikátov.

Posledným a najkomplikovanejším krokom sa nakoniec ukázalo vytváranie RDF reprezentácie jednotlivých n-tíc, a to z dôvodu kontroly už existujúcich databázových objektov reprezentujúcich aktuálne spracovávané časti n-tice. Tento problém je popísaný v podsekcii 4.4.3 implementačnej kapitoly, a grafy reprezentujúce časovú náročnosť konečného riešenia vo forme perzistentnej vyrovnávacej pamäte pomocou *ZODB* na dedikovanom stroji sú uvedené na obrázkoch 5.4 a 5.5.

Ako je možné vidieť na prvom grafe na obrázku 5.4, až do zhruba 50-tisíc spracovaných článkov zvládala konverzia na RDF reprezentáciu spracovávať články na úrovni stoviek milisekúnd. Postupne sa však databáza vyrovnávacej pamäte *ZODB* zapĺňala identifikátormi už existujúcich objektov a od zhruba tisícej sady začala priemerná dĺžka spracovania jedného článku exponenciálne rásť. Pílový tvar je spôsobený pravidelným spúšťaním kompresií nad *ZODB*, ktorej veľkosť rástla čím ďalej rýchlejšie. Celkovo po takmer piatich dňoch trvala analýza obsahu databázy 4,5 hodiny a kompresia 6 hodín, takže po spracovaní presne 112050 článkov bola konverzia počas ďalšej kompresie násilne ukončená. Graf na obrázku 5.5 zobrazuje vývoj časovej náročnosti analýzy a kompresie obsahu *ZODB* databázy. Celkovo bolo vytvorených 11719899 objektov trojíc, 13384761 objektov zdrojov využívaných ako podmet, predmet alebo doplnok a 903686 predikátov.



Obr. 5.3: Graf zobrazujúci priemerné trvanie analýzy fráz jedného článku vypočítané na sade tvorenej vždy päťdesiatimi článkami.

Každý z 50-tich článkov danej sady bol po konverzii do RDF reprezentácie vkladaný do virtuálneho RDF úložiska v operačnej pamäti realizovaného knižnicou *RDFLib*. Ten bol po každej sade skonvertovaný do formátu *Turtle*, zapísaný do súboru a obsah grafu bol vyprázdnený. Dĺžku tejto konverzie zobrazuje graf 5.6, kde je možné vidieť, že v jeho koncových fázach prišlo ku krátkemu manuálnemu testovaniu v súvislosti s pokusmi o optimalizáciu iných častí na menšom počte článkov, a tieto výsledky boli tiež zaznamenané. Celkovo bolo vygenerovaných 394 nadväzujúcich súborov s RDF reprezentáciou spracovaných n-tíc, pričom priemerná doba na vygenerovanie jedného súboru bola 37,8223 sekúnd.

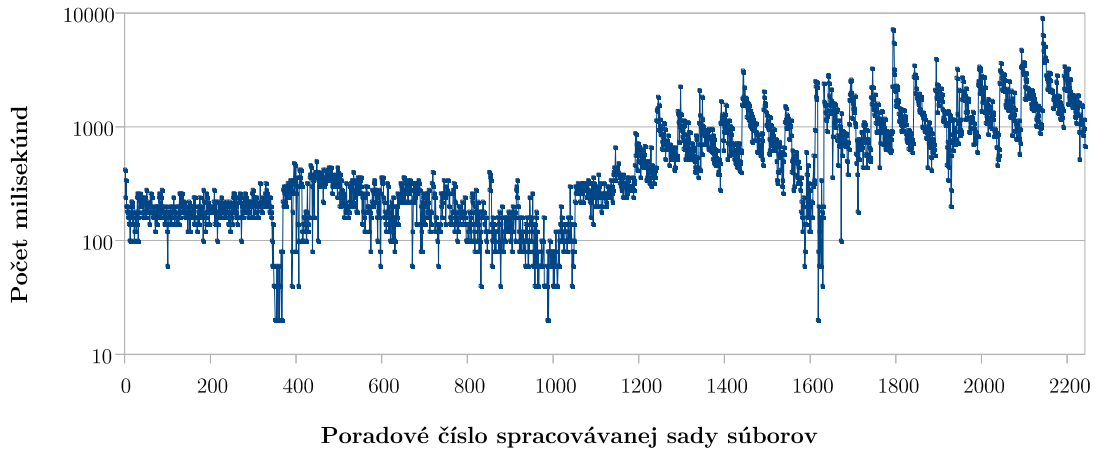
5.1.2 Ďalšie štatistiky

Počas prvotného zoznamovania sa s extrakčným systémom *Open IE* bola vykonaná na staršom výpise extrakcia všetkých článkov bez akýchkoľvek ďalších krokov, ktorej úlohou bolo zistiť, koľko extrakcií je vytvorených z jedného článku. Graf na obrázku 5.7 zobrazuje túto štatistiku.

Ako už bolo spomínané v predchádzajúcej podsekcii, pri každej kompresii bola spúšťaná aj analýza obsahu vyrovnávacej pamäte, aby bolo možné sledovať vývoj zmien a počet novovytvorených objektov RDF reprezentácie. Graf na obrázku 5.8 zobrazuje priebežné prírastky počtu nových objektov trojíc, zdrojov a predikátov v databáze po spracovaní vždy 2500 článkov. Je zaujímavé vidieť, že počet novovytvorených trojíc a zdrojov je v úzkej korelácii, aj keď je z každej trojice získaná zvyčajne dvojica alebo trojica zdrojov. V prípade zapnutého rozgenerovávania výrazov obsahujúcich priraďovacie spojky sa dá predpokladať, že by sa vo výsledku objavovalo ešte významne menšie množstvo unikátne pôsobiacich fráz.

5.1.3 Pamäťové nároky

Zdrojové články slúžiace na spracovanie mali na prvej stanici spolu veľkosť 311MB, na druhej 288MB, na tretej 344MB, štvrtej 418MB a piatej 380MB. Tieto články boli uložené v kódovaní UTF-8 a celkovo ich bolo na každej stanici presne 40-tisíc. To, ako sa



Obr. 5.4: Graf zobrazujúci priemerné trvanie vytvárania RDF reprezentácie jedného článku vypočítané sade tvorenej vždy päťdesiatimi článkami.

v jednotlivých krokoch menila celková veľkosť spracovaných dát uvádza tabuľka B.1 v prílohe B. Priemerne v kroku koreferencie narástla celková veľkosť dát o približne 3,5%, o 11,2% pri extrakcii a o 77% pri analýze fráz. Tieto zmeny sú udávané vzhľadom na veľkosť pôvodných zdrojových dát a neberú do úvahy nespracované súbory.

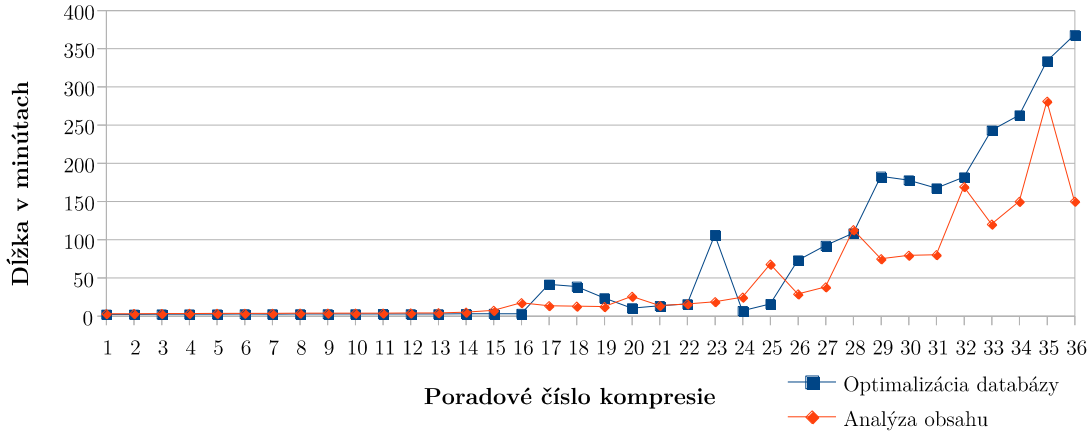
Po vzatí do úvahy, že bolo nakoniec do RDF reprezentácie skonvertovaných len 56,03% z pôvodného počtu článkov bolo vypočítané, že po vygenerovaní RDF dát vo formáte *Turtle* sa zväčší objem vzhľadom na pomernú veľkosť vstupných dát o 600,1%. Skomprimovaná vyrovnávacia pamäť databázy ZODB obsahujúca existujúce identifikátory objektov k týmto dátam predstavovala 8GB. Po vložení týchto dát do distribuovanej databázy *4store* vo výpočtovom klastri o desiatich staniciach zaberá na prvých dvoch staniciach 2,5GB a na zvyšných ôsmych 1,9GB.

5.2 Kvalita koreferencie

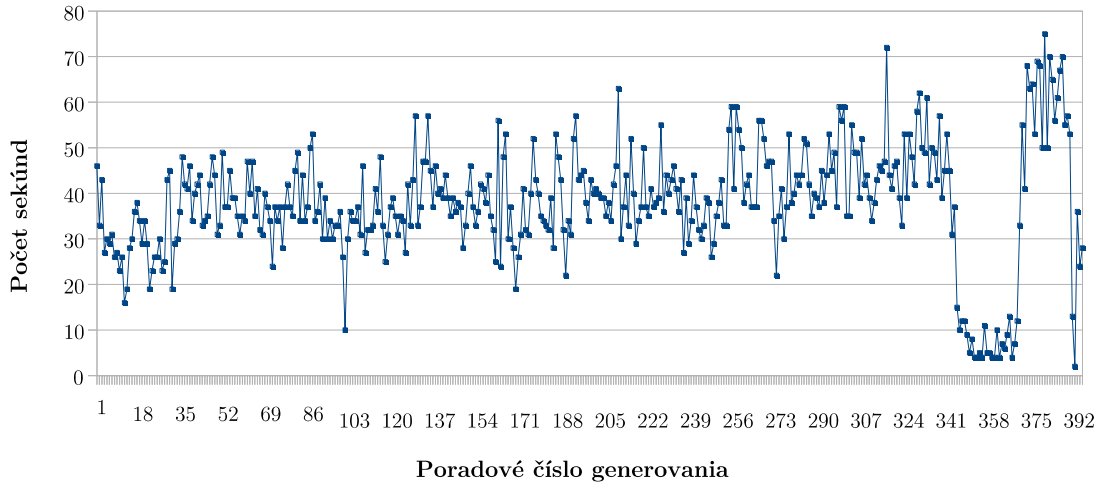
Na vyhodnotenie kvality koreferencie bolo použitých prvých 30 viet z článku o Pablovi Picassovi. Na tomto úryvku boli manuálne určené všetky postupnosti podstatných mien a zámen. Ďalej boli ohodnotené podľa toho, či boli vhodné alebo nepotrebné na použitie koreferencie. Následne bola na týchto vetách vykonaná koreferencia, pričom bolo hodnotené či systém vykonal koreferenciu na očakávaných slovách a zároveň či bola vykonaná správne. Výsledky boli zaznačené do kontingenčnej tabuľky 5.1. Z tejto tabuľky bola vypočítaná miera pokrytia rovná 46,05% a miera presnosti rovná 63,64%.

Systém zvládol koreferenciu jednoduchých zámen ako *his*, *her* a podobne. Tiež dokázal vykonať koreferenciu, ak bolo uvedené iba priezvisko, ktoré expandoval na celé meno. Avšak v prípade zámen koreferenciu občas nevykonal vôbec, prípadne ju vykonal chybne a určil vzhľadom na inú osobu uvedenú vo vete. V prípade komplikovaných viet, v ktorých vystupovali napríklad viacerí členovia rodiny, občas určil koreferenciu vzhľadom na iného člena rodiny ako bolo očakávané.

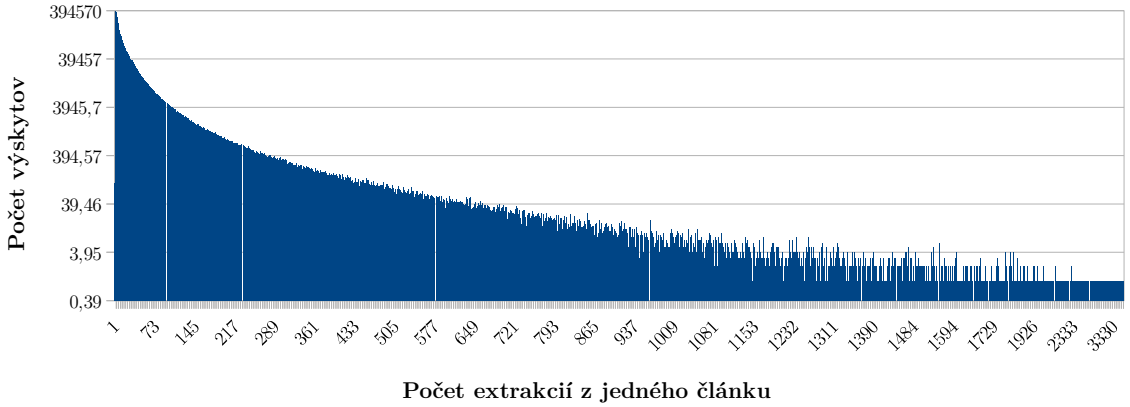
Systém nezvládol vôbec koreferenciu, resp. niekedy vykonal len jej časť, ak išlo o spo-



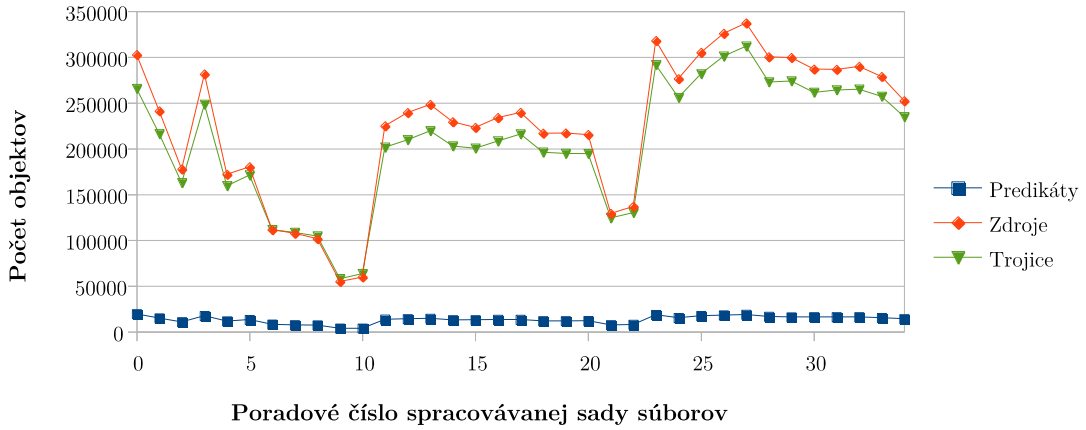
Obr. 5.5: Graf zobrazujúci dĺžku trvania analýzy obsahu ZODB databázy a jej optimalizácie po spracovaní sád tvorených vždy päťdesiatimi článkami.



Obr. 5.6: Graf zobrazujúci dĺžku trvania generovania RDF popisu novovytvorených objektov databázy vo formáte *Turtle* po spracovaní sád tvorených vždy päťdesiatimi článkami.



Obr. 5.7: Štatistika počtu vyextrahovaných faktov z jedného článku.



Obr. 5.8: Graf zobrazujúci počet novovytvorených objektov v databáze po spracovaní 50 sád tvorených vždy päťdesiatimi článkami.

jenie zámena a osoby pomenovanej nepriamo, z ktorého spojenia sa dalo vyvodiť, o ktorú osobu v článku sa jedná. Príkladom takýchto fráz, niekedy nevykonaných vôbec a niekedy vykonaných len vzhľadom na zámeno, sú napríklad *his father* alebo *his younger sister*. Ďalej v jednom prípade systém koreferencoval Picassovu sestru menom *Conchita* na súčasnú americkú popovú speváčku Selenu Gomez a v inom prípade označil slávne dielo *Guernica* ako španielske mesto *Gernika-Lumo*. V druhom prípade sa síce dá nájsť súvislosť medzi Picassovým dielom a uvedeným mestom, avšak z pohľadu účelu použitia koreferencie v našom systéme, pri ktorej sa nahrádzajú nejednoznačné slová ich jednoznačnejšími variantami, ide o nevhodné nahradenie.

5.3 Úspešnosť normalizácie

Na zdrojovej databáze, ktorej obsah bol vytvorený z rozsiahleho článku o vedcovi Albertovi Einsteinovi bolo spomedzi dostupných 10904 predikátov náhodne vybraných 500

		Človek	
		Potrebná	**Nepotrebná**
Systém	**Správne**	35	20
	Nesprávne	41	92

Tabuľka 5.1: Kontingenčná tabuľka pre hodnotenie úspešnosti koreferencie.

rozdielnych a tieto boli manuálne kontrolované na správnosť normalizácie. Z tohoto počtu bolo identifikovaných práve 7 predikátov, ktoré neboli gramaticky správne normalizované, čo predstavuje približne 1,4% chybu, resp. 98,6% presnosť normalizácie. Treba však podotknúť, že za správne boli považované aj predikáty, kde sa frázové slovesá normalizovali na ich základný tvar bez zachovania predložiek, ktoré sa bežne považujú za ich súčasť, pretože išlo o očakávané správanie algoritmu navrhovaného v podsekcii 3.6.2 kapitoly 3.

Medzi chybne normalizované predikáty patrilo *sketches* normalizované na *sketches*, *taught* na *taught*, *joined* na *joined*, *was married* na *marrie*, *is a favorite model for* na *favorite*, *was the first university in* na *first* a *was Personal life Supporter of* na *personal*.

Medzi komplikovanejšie predikáty, ktorých normalizácia bola považovaná za správnu, boli zaradené napríklad aj *were nearly universally rejected* na *reject*, *is a better teacher than* na *be*, *co-invented* na *co-invent*, *[is] star [of]* na *be*, *have never felt* na *feel*, *can split apart* na *split*, *was positively charged* na *charge* alebo *didn't* na *do*. Ako už bolo spomínané, počítala sa sem aj normalizácia frázových slov, ako napríklad *went on* na *go* alebo *had not given up* na *give*.

Súčasne s normalizáciou bola tiež analyzovaná úspešnosť detegovania negácií v predikátoch. Predikáty ako *didn't*, *were not well understood* alebo *cannot be* a im podobné boli všetky správne detegované ako negatívne. Jediné objavené predikáty, ktoré mali byť označené ako záporné a neboli, boli spojenia, kde negáciu spôsobovalo slovo *never* ako napríklad v *have never felt*.

5.4 Úspešnosť určovania typu fráz

Taktiež na databáze zostavenej z extrakcií článku o Albertovi Einsteinovi bolo analyzované, ktoré frázy sa podarilo označiť ako osoby, *niečo*, čas alebo miesto. V tomto prípade však boli do vyhodnocovania zahrnuté aj extrakcie vzniknuté krokom rozgenerovania súvetí obsahujúcich priraďovacie spojky, aby bola vyhodnotená úspešnosť navrhovaného systému predpokladajúceho aj prítomnosť takýchto dát.

Na vyhodnotenie úspešnosti systému zaradiť správne určitú frázu do jednej z týchto kategórií bolo zvolených náhodne 250 unikátnych fráz, ktoré boli manuálne zaradené do aspoň jednej z týchto kategórií, pričom systém zaraďoval do práve jednej z týchto kategórií. Výsledky boli zaznačené do kontingenčných tabuliek 5.2 až 5.5.

Pre kontingenčnú tabuľku 5.2 vyšlo pokrytie 74,68% a presnosť 84,29%, pre tabuľku 5.3 92,11% a 83,83%, pre tabuľku 5.4 43,43% a 81,25% a pre tabuľku 5.5 73,68% a 82,35%. Priemerné pokrytie pri detekcii typov teda vychádza na 70,98% a priemerná presnosť na 82,93%.

Medzi chybné frázy identifikované ako osoba boli zaradené napríklad *quantum mechanics*, *Einstein's remains*, *about 3,500 pages of private correspondence*, *Einstein's political view*,

		Človek	
		Áno	**Nie**
Systém	**Áno**	59	11
	Nie	20	160

Tabuľka 5.2: Kontingenčná tabuľka pre hodnotenie úspešnosti detekcie fráz typu *niekto*.

		Človek	
		Áno	**Nie**
Systém	**Áno**	140	27
	Nie	12	71

Tabuľka 5.3: Kontingenčná tabuľka pre hodnotenie úspešnosti detekcie fráz typu *niečo*.

		Človek	
		Áno	**Nie**
Systém	**Áno**	13	3
	Nie	17	217

Tabuľka 5.4: Kontingenčná tabuľka pre hodnotenie úspešnosti detekcie fráz typu *niekde*.

		Človek	
		Áno	**Nie**
Systém	**Áno**	14	3
	Nie	5	228

Tabuľka 5.5: Kontingenčná tabuľka pre hodnotenie úspešnosti detekcie fráz typu *niekedy*.

Einstein's intellectual achievements, *Albert Einstein and Mileva Marić's friendship*, *Albert Einstein's expressive face and distinctive hairstyle* alebo *Albert Einstein's celebrated 1905 papers*.

Naopak, medzi komplikovanejšie, ale za to správne detekcie osoby patrili *Barbara Wolff of The Hebrew University's Albert Einstein Archives*, *British historian Martin Gilbert*, *Albert Einstein's former student Leó Szilárd*, *Jewish invitees to Republic of Turkey* alebo *Time magazine's Frederic Golden*.

V prípade zaraďovania do kategórie *niečo* sem bola zaradená väčšina fráz, z ktorých medzi zaujímavejšie patria napríklad *Other principles postulated by Albert Einstein*, *In his lecture at Einstein's memorial*, *The following publications by Albert Einstein* alebo univerzita *Lincoln*.

Podarilo sa sem chybne zaradiť aj niektoré osoby ako námorný dôstojník menom *Locker-Lampson*, holandský fyzik *De Haas* alebo Einsteinova sestra *Maja*.

Medzi *niečo* boli tiež zaradené aj *the Albert Einstein family* alebo sláčikové kvarteto *the young Juilliard Quartet*, ktoré sa dajú považovať za nejednoznačné.

Do kategórie časového obdobia bola zaradená väčšina fráz, ktoré obsahovali iba informáciu o časovom období, ako napríklad *over the next few years*, *During Albert Einstein's Prague stay*, *in May 1904*, *that same year*, *for two weeks* a podobné.

Do kategórie miesto boli zaradené napríklad *in republic of Austria*, *in Zürich*, *among others*, *in their letters*, *to Repubblica Italiana*, *at King's College*. Za chybné zaradenie by sa dalo považovať *as now in my whole life* alebo *in infancy*.

5.5 Kvalita extrakcií vo vedomostnej databáze

Na vyhodnotenie kvality extrakcií bol využitý opäť článok o Albertovi Einsteinovi. Náhodne bolo vybraných 290 unikátnych fráz, ktoré vznikli extrakciou pomocou nástroja *Open*

IE. Tieto frázy boli manuálne hodnotené ako zmysluplné alebo nie, a keďže tieto frázy zahŕňali aj tie, ktoré vznikli rozgenerovaním fráz obsahujúcich priraďovacie spojky, bol počítaný nielen pomer kvality všetkých fráz, ale aj týchto novovzniknutých.

Z 290 hodnotených fráz bolo určených 251 ako správnych a 39 ako nesprávnych, čo predstavuje 86,55% správnosť, resp. 13,45% chybovosť vyextrahovaných fráz. Spomedzi týchto fráz bolo 78 vytvorených rozgenerovaním fráz obsahujúcich priraďovacie spojky. Z týchto novovytvorených fráz bolo 69 označených ako správne a 9 ako nesprávne, čo predstavuje 88,46% správnosť, resp. 11,54% chybovosť.

Rozgenerovávanie však zohráva dôležitú úlohu najmä pri prepájaní súvislostí medzi extrakciami, čo je využiteľné najmä na zodpovedávanie bežných otázok popísaných v ďalšej sekcii, keďže bez jeho použitia pôsobí väčšina fráz unikátne, aj keď sa v nich opakovane spomínajú tie isté osoby, udalosti či veci. Väčšina problémov súvisiacich s rozgenerovávaním bola spôsobená spojením *between X and Y*, ktoré bolo rozdelené na dve samostatné frázy *between X* a *between Y*, čo je aj ukázané na obrázku 5.9.

5.6 Odpovedanie na bežné otázky

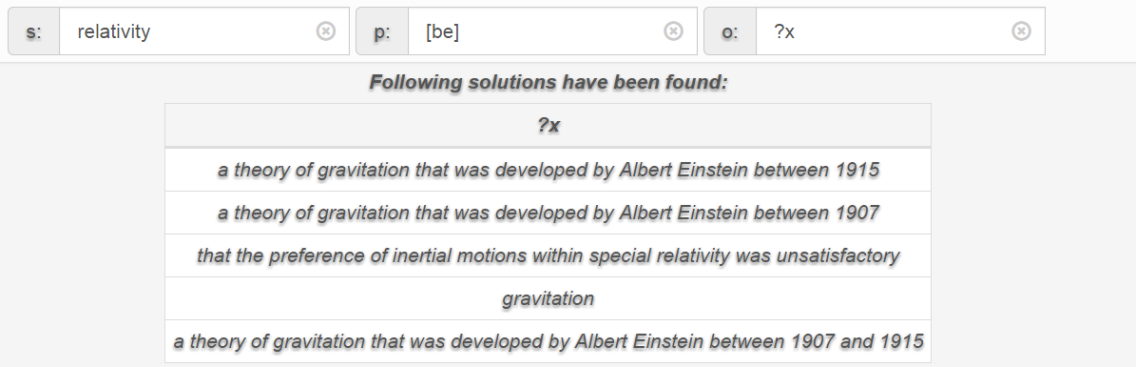
Pri praktickom používaní sa ako najväčší problém ukázal *soft limit* spomínaný v podsekcii 4.1.8 predchádzajúcej kapitoly. Toto nastavenie obmedzuje počet záznamov spracovaných počas vyhľadávania odpovede za cenu rýchlosti vyhodnotenia. Ak bolo toto obmedzenie vypnuté alebo nastavené na príliš vysokú hodnotu, pri použití na databázu získanú vyextrahovaním vyše 110-tisíc článkov trvalo spracovanie jednoduchých dotazov na úrovni desiatok sekúnd a použitie dotazov zadaných kombináciou viacerých prepojených n-tíc bolo spracovávané minúty, pričom občas nebolo vôbec ukončené v prijateľnom čase. Pri zapnutí rozumne nastaveného obmedzenia trvalo vyhľadávanie iba pár sekúnd, avšak za cenu veľmi obmedzeného počtu prinášaných výsledkov a prevažne dokonca žiadnych.

Vyhľadávanie v tomto systéme však má potenciál, ktorý až do vyriešenia spomínaných problémov je možné demonštrovať na malej databáze, vytvorenej napríklad z rozsiahleho článku o Albertovi Einsteinovi. Po použití kroku zabezpečujúceho rozgenerovanie fráz obsahujúcich priraďovacie spojky vznikne 15620 RDF trojíc, z ktorých je 1159 trojíc predstavujúcich extrakcie, 1627 zdrojov, 479 predikátov a zvyšok sú RDF trojice modelujúce ďalšie vlastnosti spomínaných objektov a prepojenia v tejto databáze.

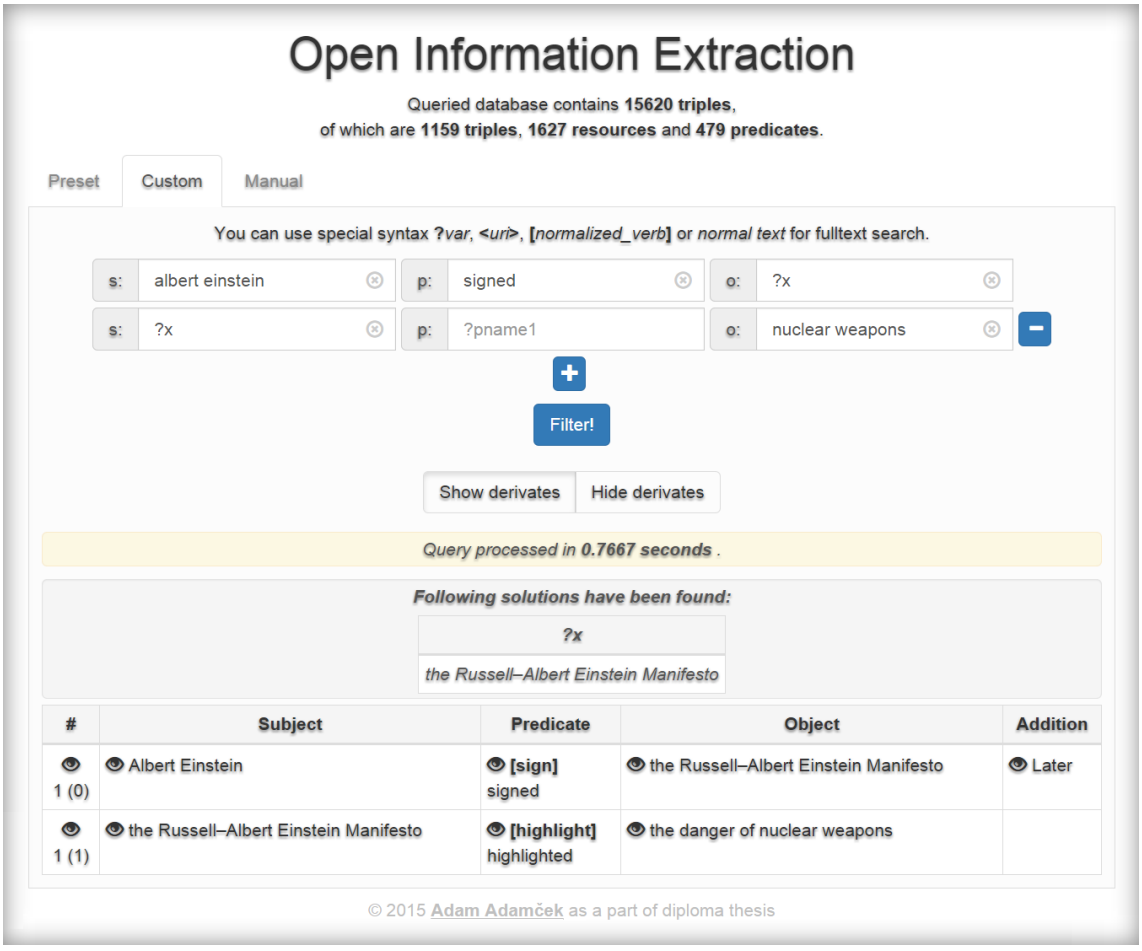
Pomocou navrhnutého rozhrania je možné hľadať odpovede na jednoduché otázky ako napríklad *Čo je relativita?*. Príklad zápisu takejto otázky do vhodného tvaru v našom rozhraní, z ktorého je následne vytvorený dotaz smerujúci na RDF databázu, je ukázaný na obrázku 5.9. Spomedzi vyhľadaných výsledkov sú však relevantné iba posledné dva, čo ukazuje, že pri nasadzovaní do praxe je potrebné vyriešiť aj problém určovania relevantných výsledkov.

Keďže systém podporuje rozhranie, pri ktorom je možné zadať aj viacero n-tíc, v ktorých sa môžu vyskytovať premenné navzájom prepájajúce tieto n-tice, bolo toto rozhranie použité na simulovanie zložitejších dotazov, ktoré by boli po vhodnej transformácii do takejto formy pretvorené. Systém je následne schopný vyhľadať takú odpoveď, ktorá vyhovuje všetkým miestam, na ktoré má byť v zadaní naviazaná. Príklad je uvedený na obrázku 5.10, kde je využitá premenná `?x`, ktorá má vo výsledku vystupovať ako odpoveď na otázku používateľa voľne preloženú ako *Čo týkajúce sa nukleárnych zbraní podpísal Albert Einstein?*. Je potrebné dodať, že táto otázka je z dôvodu prehľadnosti zapísaná tak, že v skutočnosti za premennú `?x` môže byť v zobrazenom prípade naviazaný ľubovoľný typ frázy, avšak systém umožňuje aj obmedzenie typu frázy na *niečo* v aktuálnej realizácii za cenu zníženej prehľadnosti zápisu.

Zaujímavosťou je, že uvedený výsledok bol ovplyvnený krokom koreferencie, keďže pôvodný názov vyhľadaného manifestu je v origináli *Russell-Einstein Manifesto*.



Obr. 5.9: Ukážka jednoduchého dotazu zodpovedajúceho otázku *„What is relativity?"* a zoznam nájdených riešení.



Obr. 5.10: Ukážka zložitejšieho dotazu zodpovedajúceho otázku *„What has Albert Einstein signed about nuclear weapons?"*.

Kapitola 6

Zhodnotenie riešenia a možnosti ďalšieho napredovania

Na konci predchádzajúcej kapitoly bolo ukázané, ako je možné využiť vytvorený systém na zodpovedanie bežných otázok používateľa tohoto systému. Samozrejme, k univerzálnemu fungovaniu na otázky položené v prirodzenom jazyku vedie ešte dlhá cesta, pričom absolútny základ by bola správna transformácia takejto otázky na správny SPARQL dotaz podporovaný našou databázou. Keďže prehľadávanie podľa zadaného textu sa momentálne spolieha na fulltextové vyhľadávanie v jednotlivých reťazcoch, nemusia byť vyhľadané všetky fakty relevantné pre našu otázku. Ďalšia potrebná vec na vyriešenie je preto taktiež prehľadávanie synoným a branie kontextu do úvahy, keďže v prípade použitia všeobecnejších slov môže byť vyhľadané naozaj veľké množstvo výsledkov.

Medzi ďalšie možné spôsoby ako zlepšiť úspešnosť vyhľadávania patria vylepšenia úspešnosti fungovania jednotlivých komponent, ktoré systém využíva, a to menovite služby na koreferenciu a určovanie pomenovaných entít, extraktora faktov s cieľom zvýšiť kvalitu extrakcií a zrevidovať navrhnuté algoritmy s cieľom odstrániť slabé miesta objavené počas testovania. Veľmi dôležité je však nahradenie aktuálneho relatívne pomalého syntaktického analyzátora oveľa rýchlejším, prípadne nájdenie alternatívneho výpočtovo oveľa efektívnejšieho riešenia rozgenerovávajúceho zložité frázy obsahujúce priraďovacie spojky.

Ukladanie ďalších informácií viazaných na obsiahnuté fakty vedie k ďalšiemu problému, ktorým je výkon systému. Počas testovania bolo ako najvýznamnejší problém, ktorému je potrebné čeliť, identifikované nutné vyriešenie efektívnej konverzie extrakcií do RDF formátu. Po vyskúšaní viacerých prístupov a objavení ich hraníc možného použitia sa ako ďalšie možné riešenie naskytá využitie štandardnej relačnej databázy ako vyrovnávacej pamäte pre predvídateľne rýchle vyhľadávanie identifikátorov už existujúcich RDF objektov. Ďalším výkonnostným problémom sa ukázala rýchlosť vyhľadávania odpovedí. Systém je možné nastaviť tak, aby vyhľadával rýchlo, avšak prinášal veľmi obmedzené riešenia, alebo podával v rozumnom čase presné riešenia z veľmi limitovanej vedomostnej databázy. Je žiadané nájsť iné riešenie ako len možnosť navyšovať počet staníc výpočtového klastra prevádzkujúceho distribuovanú vedomostnú databázu. Možné riešenie je v skúmaní dostupných optimalizácií vytváraných SPARQL dotazov alebo tiež odskúšanie iných dostupných RDF úložísk, z ktorých je však väčšina platená.

Po odladení vyššie uvedených problémov je možné čeliť ďalšej veľkej výzve – podpore iných korpusov, najmä ďalších voľne dostupných dát zhromaždených projektom *Common Crawl* [50]. Tento korpus obsahuje nielen gramaticky správne napísané články pokúšajúce

sa zachytiť iba faktické informácie, ale zmes rozličných typov obsahov získaných z rôznych častí internetu ako napríklad aj konverzácie na internetových fórach alebo ponuky v internetových obchodoch. Pre dosiahnutie porovnateľnej kvality vyextrahovaných informácií ako z *Wikipédie* je v tomto prípade potrebné sa vysporiadať sa mnohými ďalšími problémami. Je potrebné ošetriť používanie nespisovného jazyka, neaktuálnosť textu a existujú tiež aj dodatočné komplikácie ako informácie organizované v tabuľkách a odrážkach v oveľa vyššej miere ako na *Wikipédii*, multimédiá vystupujúce ako súčasť kontextu a iné.

Záver

Úlohou tejto práce bolo zaoberať sa problémom extrakcie informácií a navrhnúť systém zabezpečujúci získanie faktov zo zvolenej dátovej sady, ktorú v našom projekte predstavoval voľne dostupný výpis článkov anglickej mutácie internetovej encyklopédie *Wikipédia*, a ich následné uloženie do RDF databázy. Ďalej vytvoriť rozhranie umožňujúce dotazovanie do tejto databázy za účelom hľadania odpovedí na otázky používateľa.

Prvým krokom pri riešení tejto práce bolo identifikovať vhodnú literatúru a oboznámiť sa so základnými pojmami a prístupmi súvisiacimi s oblasťou spracovania prirodzeného jazyka. Taktiež bolo potrebné naštudovať odborné články týkajúce sa jednotlivých extrakčných systémov, a získať tak predstavu o procese extrakcie či princípoch, na akých jednotlivé extraktory fungujú.

Ďalším krokom bolo vytvorenie návrhu systému, ktorý mal byť schopný vykonať extrakciu a jeho postupná implementácia. Samotný návrh bol niekoľkokrát revidovaný a prispôsobovaný, a to najmä z dôvodov postupného objavovania a voľby technológií využitých pri implementácii a následnom zisťovaní ich možností, resp. nedostatkov. Počas vývoja sa objavilo množstvo nepredpokladaných problémov, na ktoré bolo potrebné hľadať riešenia, čo v konečnom dôsledku viedlo aj už spomínanému prispôsobovaniu pôvodného návrhu.

Taktiež, nezanedbateľnú časť práce zahŕňalo zoznamovanie sa prostredím výpočtového klastra a najmä jeho skupinovým ovládaním. Keďže paralelná práca na viacerých strojoch prináša mnohé úskalia, najčastejšie z dôvodu nie úplne identických prostredí systémov či ich zdieľania s viacerými používateľmi vykonávajúcimi ich vlastné rozmanité úkony, vykonávanie rovnakých krokov často neprináša okamžite očakávané výsledky a je mnohokrát potrebné riešiť a ošetrovať špecifické problémy pre vzniknuté situácie. Aj keď podpora výpočtového klastra nie je primárnym predmetom tejto práce ale iba jednou z implicitných vlastností navrhovaného systému, považujem ju za dôležité pripomenúť aspoň v závere tejto práce, keďže veľká časť času pri riešení bola strávená práve na odstraňovaní problémov, ktoré podpora mnohovýpočtového prostredia spôsobovala.

Tento systém bol následne otestovaný na článkoch už spomínanej anglickej verzii internetovej encyklopédie *Wikipédia*, kde bola analyzovaná rýchlosť spracovania jednotlivých komponent vytvoreného systému a úspešnosť ich fungovania. Taktiež bolo prezentované použitie vytvoreného rozhrania za účelom zodpovedania bežných otázok vyplývajúcich z obsahu databázy, ktoré by mohli používateľa nášho systému zaujímať.

Posledná kapitola pokúša nielen zhodnotiť dosiahnuté výsledky, ale keďže počas testovania boli identifikované určité slabé stránky vytvoreného systému, vychádza z týchto poznatkov pri návrhu možného napredovania tohoto projektu.

Výsledkom tejto práce je prehľad nadobudnutý v oblasti spracovania prirodzeného jazyka, najmä problému extrakcie informácií, a vyvinutý systém skladajúci sa zo sady skriptov zabezpečujúcich extrakciu dát z článkov dátovej sady, ich uloženie do RDF databázy v prostredí výpočtového klastra poskytnutého *Výskumnou skupinou znalostných technológií FIT VUT v Brne* a rozhrania umožňujúceho vykonávať dotazy nad týmito extrakciami.

Literatúra

[1] Wu, F.; Weld, D. S.: Open information extraction using Wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2010, s. 118–127.

[2] Pala, K.: *Počítačové zpracování přirozeného jazyka*. Brno: FI MU, neoficiální vydání, 2000, 128 s.

[3] Garabík, R.; Giantisová, L.; Horák, A.; aj.: Tokenizácia, lematizácia a morfologická anotácia Slovenského národného korpusu. *Interný materiál – stále platný východiskový manuál na ručnú morfologickú anotáciu*, 2004, nepublikované.

[4] WebLicht: Tools in Detail [online]. `http://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/Tools_in_Detail`, 2013, [rev. 2015-01-07].

[5] Český národní korpus: Příručka ČNK: Pojmy/korpus [online]. `http://wiki.korpus.cz/doku.php/pojmy:korpus`, 2014, [rev. 2015-05-23].

[6] Institute of Formal and Applied Linguistics: Prague Dependency Treebank 3.0 [online]. `https://ufal.mff.cuni.cz/pdt3.0`, 2015, [rev. 2015-05-23].

[7] The Trustees of Princeton University: WordNet: A lexical database for English [online]. `https://wordnet.princeton.edu/`, 2015, [rev. 2015-05-19].

[8] Marcus, M. P.; Marcinkiewicz, M. A.; Santorini, B.: Building a large annotated corpus of English: The Penn Treebank. *Computational linguistics*, ročník 19, č. 2, 1993: s. 313–330.

[9] Grishman, R.: MUC-6 [online]. `http://www.cs.nyu.edu/cs/faculty/grishman/muc6.html`, 2001, [rev. 2014-12-20].

[10] The National Institute of Standards and Technology: Automatic Content Extraction (ACE) Evaluation [online]. `http://www.itl.nist.gov/iad/mig/tests/ace/`, 2009, [rev. 2014-12-20].

[11] Chang, C. H.; Kayed, M.; Girgis, M. R.; aj.: A survey of web information extraction systems. *Knowledge and Data Engineering, IEEE Transactions on*, ročník 18, č. 10, 2006: s. 1411–1428.

[12] Sarawagi, S.: Information extraction. *Foundations and trends in databases*, ročník 1, č. 3, 2008: s. 261–377.

[13] Cunningham, H.: Information extraction, automatic. *Encyclopedia of language and linguistics,*, 2005: s. 665–677.

[14] Banko, M.; Cafarella, M. J.; Soderland, S.; aj.: Open information extraction for the web. In *IJCAI*, ročník 7, 2007, s. 2670–2676.

[15] Etzioni, O.; Cafarella, M.; Downey, D.; aj.: Web-scale information extraction in knowitall:(preliminary results). In *Proceedings of the 13th international conference on World Wide Web*, ACM, 2004, s. 100–110.

[16] Etzioni, O.: Moving up the information food chain: Deploying softbots on the world wide web. *AI magazine*, ročník 18, č. 2, 1997: str. 11.

[17] Cafarella, M. J.; Downey, D.; Soderland, S.; aj.: KnowItNow: Fast, scalable information extraction from the web. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2005, s. 563–570.

[18] Yates, A.; Cafarella, M.; Banko, M.; aj.: Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, Association for Computational Linguistics, 2007, s. 25–26.

[19] Fader, A.; Soderland, S.; Etzioni, O.: Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2011, s. 1535–1545.

[20] Schmitz, M.; Bart, R.; Soderland, S.; aj.: Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Association for Computational Linguistics, 2012, s. 523–534.

[21] Schmitz, M.; Bart, R.; Soderland, S.; aj.: SRLIE [online]. `https://github.com/knowitall/srlie`, 2014, [rev. 2015-01-07].

[22] Wikimedia Foundation: Wikimedia Downloads [online]. `http://dumps.wikimedia.org/`, 2015, [rev. 2015-01-04].

[23] König, E.; van der Auwera, J.: *The Germanic Languages*. Taylor & Francis, 1994, 532,562 s.

[24] RDF Working Group: Resource Description Framework (RDF) [online]. `http://www.w3.org/RDF/`, 2014, [rev. 2015-01-04].

[25] Beckett, D.: A line-based syntax for an RDF graph [online]. `http://www.w3.org/TR/n-triples/`, 2014, [rev. 2015-04-17].

[26] Free Software Foundation: GNU Project: Bash [online]. `http://www.gnu.org/software/bash/`, 2014, [rev. 2015-03-11].

[27] Python Software Foundation: Python [online]. `https://www.python.org/`, 2014, [rev. 2015-01-04].

[28] Chacon, S.; Straub, B.: *Pro Git*. Berkely, CA, USA: Apress, první vydání, 2009, ISBN 978-1-43-021833-3.

[29] Ananiev, A.: JSON Pros and Cons [online]. `https://myarch.com/json-pros-and-cons/`, 2007, [rev. 2015-03-29].

[30] Bray, T.: The JavaScript Object Notation (JSON) Data Interchange Format [online]. `https://tools.ietf.org/html/rfc7158`, 2014, [rev. 2015-03-29].

[31] The W3C SPARQL Working Group: SPARQL 1.1 Overview [online]. `http://www.w3.org/TR/sparql11-overview/`, 2013, [rev. 2015-01-06].

[32] Polikoff, I.: Comparing SPARQL with SQL [online]. `http://www.topquadrant.com/2014/05/05/comparing-sparql-with-sql/`, 2014, [rev. 2015-01-06].

[33] Feigenbaum, L.: SPARQL By Example: The Cheat Sheet [online]. `http://www.cambridgesemantics.com/2008/09/sparql-by-example/`, 2009, [rev. 2015-04-15].

[34] European Community's Seventh Framework Programme: DECIPHER [online]. `http://decipher-research.eu/`, 2015, [rev. 2015-04-21].

[35] Cafarella, M.; Etzioni, O.; Banko, M.; aj.: Open IE [online]. `http://knowitall.github.io/openie/`, 2014, [rev. 2015-01-04].

[36] The Stanford Natural Language Processing Group: Stanford CoreNLP: A Suite of Core NLP Tools [online]. `http://nlp.stanford.edu/software/corenlp.shtml`, 2015, [rev. 2015-04-12].

[37] Johnson, M.; Charniak, E.; McClosky, D.: Python bindings for the BLLIP natural language parser [online]. `https://pypi.python.org/pypi/bllipparser/`, 2014, [rev. 2015-01-04].

[38] NLTK Project: Natural Language Toolkit [online]. `http://www.nltk.org/`, 2015, [rev. 2015-04-10].

[39] CLiPS Research Center: Pattern [online]. `http://www.clips.ua.ac.be/pages/pattern`, 2015, [rev. 2015-04-16].

[40] Garlik: 4store: Scalable RDF storage [online]. `http://4store.org/`, 2015, [rev. 2015-05-12].

[41] Ronacher, A.: Flask (A Python Microframework) [online]. `http://flask.pocoo.org/`, 2015, [rev. 2015-03-15].

[42] Cohen, S.: Quora: What challenges has Pinterest encountered with Flask? [online]. `http://www.quora.com/What-challenges-has-Pinterest-encountered-with-Flask`, 2015, [rev. 2015-05-03].

[43] Otto, M.; Thornton, J.: Bootstrap: The world's most popular mobile-first and responsive front-end framework [online]. `http://getbootstrap.com/`, 2015, [rev. 2015-05-03].

[44] Google: AngularJS: Superheroic JavaScript MVW Framework [online]. https://angularjs.org/, 2015, [rev. 2015-05-03].

[45] Buschmann, F.; Henney, K.; Schmidt, D. C.: *Pattern-Oriented Software Architecture: On Patterns and Pattern Languages*. Wiley series in software design patterns, Wiley, 2007, ISBN 978-0-47-148648-0.

[46] Angular-UI team: Angular directives for Bootstrap [online]. https://angular-ui.github.io/bootstrap/, 2015, [rev. 2015-05-04].

[47] Knowledge Technology Research Group FIT BUT: Documentation of the project for the KNOT group SEC API [online]. http://sec.fit.vutbr.cz/sec_api.pdf, 2015, [rev. 2015-05-16].

[48] Watchdog: Python API library and shell utilities to monitor file system events [online]. https://pythonhosted.org/watchdog/, 2015, [rev. 2015-04-16].

[49] Zope Foundation: ZODB - a native object database for Python [online]. http://www.zodb.org/, 2011, [rev. 2015-05-18].

[50] Common Crawl [online]. https://commoncrawl.org/, 2015, [rev. 2015-05-21].

Zoznam príloh

Príloha A: Popis realizácie extrakčného systému

Príloha B: Podrobnosti k nameraným hodnotám pri experimentovaní na dátach anglickej Wikipédie

Obsah CD

- Zdrojové kódy implementovaného systému v adresári `/src/`
- Vyextrahované dáta vo formáte *Turtle* v adresári `/data/`
- Záznamy jednotlivých krokov získané pri finálnej extrakcii v adresári `/data/logs/`
- Táto práca vo formáte PDF v adresári `/thesis/`
- Zdrojové kódy tejto práce vo formáte systému LaTeX v adresári `/thesis/latex/`

Príloha A

Popis realizácie extrakčného systému

Implementovaný systém, ktorý je predmetom tejto práce, je umiestnený v priečinku `/src/`. Jednotlivé súbory, ktoré sú v ňom obsiahnuté, sú popísané v nasledujúcej kapitole. Na záver kapitoly je uvedený príklad použitia tohoto systému.

A.1 Implementácia extrakčného systému

Skripty implementovaného extrakčného systému v jazyku *Python* sa nachádzajú v priečinku `/src/python_scripts/`. Každý zo skriptov po spustení bez argumentov vypíše podporované spôsoby použitia. Význam jednotlivých súborov zoradených v tomto adresári abecedne je nasledovný:

- `bumpfiles.py` – skript slúžiaci na modifikáciu času vytvorenia u súborov predchádzajúceho kroku, ktoré ešte neboli v aktuálnom kroku spracované, aby bol upozornený skript sledujúci zmeny v priečinku predchádzajúceho kroku; je možné použitie na konkrétny súbor alebo na celý priečinok vzhľadom na iný priečinok
- `create_articles.py` – vytvorí pre každý článok zo zdrojového súboru obsahujúceho prečistené texty článkov samostatný súbor; je možné špecifikovať počet článkov a udať poradové číslo článku, od ktorého začne vytváranie
- `createrdf.py` – skript obsahujúci pomocné funkcie na vytváranie RDF grafu v operačnej pamäti pomocou knižnice *RDFLib*
- `fix_detecttype_in_sparql.py` – skript umožňujúci prepočítanie detegovaných typov fráz v prípade zmeny zodpovedného algoritmu priamo v RDF súboroch bez nutnosti vykonávať celý proces extrakcie
- `openie.py` – skript obsahujúci algoritmy na detekciu typov fráz, normalizáciu predikátov, načítanie konfigurácie, komunikáciu s konfiguračnou službou a ďalšie funkcie spoločné pre väčšinu ostatných skriptov
- `openie_coreference.py` – implementácia kroku koreferencie; umožňuje vykonať koreferenciu jedného súboru alebo celého priečinka vzhľadom na priečinok so súbormi predstavujúcimi jednotlivé články obsahujúce prečistený text v *UTF-8* kódovaní
- `openie_derivation.py` – implementácia kroku rozgenerovávania fráz obsahujúcich priraďovacie spojky; umožňuje vykonať toto rozgenerovávanie nad jedným súborom alebo celým priečinkom, ktorého obsah vznikol procesom extrakcie

- `openie_extraction.py` – implementácia kroku extrakcie; umožňuje vykonať extrakciu jedného súboru alebo celého priečinka, ktorého obsah vznikol procesom koreferencie
- `openie_ner.py` – implementácia kroku čistenia a analýzy extrakcií; umožňuje vykonať obsah tohoto kroku nad jedným súborom alebo celým priečinkom, ktorého obsah vznikol krokom rozgenerovávania fáz obsahujúcich priraďovacie spojky
- `openie_sparql.py` – implementácia kroku konverzie extrakcií do RDF a napĺňanie RDF databázy pomocou naivnej metódy, prípadne s využitím vyrovnávacej pamäte získavanej periodicky ako výpis z RDF databázy; umožňuje vykonať konverziu jedného súboru alebo celého priečinka vzhľadom na priečinok, ktorého obsah vznikol v kroku čistenia a analýzy extrakcií
- `sparql_cache.py` – skript zabezpečujúci periodické vytváranie súboru s vyrovnávacou pamäťou získanej ako obsah RDF databázy a určenej na využitie predchádzajúcim skriptom
- `sparql_create_all.py` – skript konvertujúci extrakcie na súbory vo formáte *Turtle* s využitím implementácie vyrovnávacej pamäte pomocou perzistentného úložiska *ZODB*
- `watch.py` – pomocný skript obsahujúci funkcie na vytvorenie služby sledujúcej zmeny v priečinkoch, ktorý využívajú jednotlivé kroky implementovaného systému

A.2 Implementácia používateľského rozhrania pre prístup k extrakciám

Skripty implementovaného používateľského rozhrania sa nachádzajú v priečinku `/src/frontend/`. Back-end je implementovaný v súbore `/src/frontend/openie_server.py`, pričom jeho úloha je prevádzkovať službu serveru zasielajúceho súbory front-endu klientovi pristupujúcemu cez prehliadač a komunikácia s RDF databázou. Súbory front-endu sa nachádzajú v podpriečinku `/src/frontend/templates/`:

- `index.html` – hlavný súbor front-endu, ktorý využíva ďalšie súbory v rovnakom priečinku
- `results.html` – šablóna, ktorá sa využije pri zobrazovaní prijatých výsledkov z back-endu
- `popover-html-unsafe.html` – šablóna bubliny obsahujúcej náhľad detailov jednotlivých častí extrakcií
- `script.js` – implementácia logiky front-endu zabezpečujúcej jeho dynamiku a komunikáciu s back-endom
- `style.css` – súbor s kaskádovými šablónami upravujúcimi výsledný vzhľad front-endu

A.3 Externé nástroje využívané systémom

V priečinku `/src/tools/` sa nachádzajú externé nástroje alebo ich časti, ktoré implementovaný systém využíva:

- `WSJ/` – v tomto priečinku sa nachádzajú natrénované modely pre syntaktický analyzátor *BLLIP*
- `obtain_wikiplaintext.html` – skript slúžiaci na získanie najnovšieho výpisu anglickej mutácie internetovej encyklopédie *Wikipédia*
- `openie-assembly-4.1.1-SNAPSHOT.jar` – skompilovaná verzia nástroja *Open IE* slúžiaceho na extrakciu faktov
- `wikiparser.v4.py` – skript slúžiaci na prečistenie výpisu *Wikipédie* vo formáte *XML* na čistý text

A.4 Skripty na ovládanie systému

V priečinku `/src/bash_scripts/` sa nachádzajú skripty v jazyku *Bash*, ktorých úlohou je automatizované púšťanie jednotlivých častí systému. Pre svoj chod využívajú konfiguračný súbor, ktorého predvolené nastavenia sú v súbore `/src/bash_scripts/default.conf`, avšak jednotlivé nastavenia sú prepísané obsahom súboru `/src/openie.conf`, ktorý sa odporúča využívať pri vlastnom prispôsobovaní nastavení. Vysvetlenie jednotlivých nastavení je uvedené ďalej v sekcii A.5. Súbory s funkcionalitou na ovládanie sú nasledujúce:

- `common.cfg` – súbor obsahujúci pomocné funkcie na načítanie nastavení a prehľadný výpis na obrazovku, ktorý využívajú ostatné skripty
- `4s-start.sh/4s-stop.sh` – skripty slúžiace na spustenie alebo zastavenie služby *4s-httpd*; v prípade použitia prepínača `--redeploy` je zmazaný a nanovo vytvorený celý databázový *4store* klaster a v prípade použitia `--cluster` je tento klaster reštartovaný
- `cache-start.sh/cache-stop.sh` – skripty spúšťajúce a zastavujúce beh služby zabezpečujúcej periodické vytváranie vyrovnávacej pamäte z databázy RDF využiteľnej pri procese konverzie extrakcií do RDF reprezentácie
- `coreference-start.sh/coreference-stop.sh` – skripty slúžiace na spustenie alebo zastavenie služby koreferencie
- `derivation-start.sh/derivation-stop.sh` – skripty slúžiace na spustenie alebo zastavenie služby určenej na rozgenerovávanie fráz obsahujúcich priraďovacie spojky
- `extraction-start.sh/extraction-stop.sh` – skripty slúžiace na spustenie alebo zastavenie služby extrakcie
- `frontend-start.sh/frontend-stop.sh` – skripty slúžiace na spustenie alebo zastavenie služby používateľského rozhrania
- `get-ner-tar.sh` – skripty slúžiace na vytvorenie *tar* archívu zo súborov kroku čistenia a analýzy extrakcií a umiestnenie tohoto archívu do priečinka `/tmp/`; tento skript je možné využiť pri migrovaní súborov na jeden stroj v prípade konverzie extrakcií kvôli použitiu prístupu s perzistentnou vyrovnávacou pamäťou *ZODB*
- `is-running.sh` – pomocný skript určený na kontrolu behu procesu špecifikovaného v argumente

- `java-defunct-monitor.sh` – služba monitorujúca beh extrakčného nástroja *Open IE* a v prípade nečakaného ukončenia ho znova spustí
- `ner-start.sh/ner-stop.sh` – skripty slúžiace na spustenie alebo zastavenie služby kroku čistenia a analýzy extrakcií
- `openie-status.sh` – skript vypisujúci aktuálny stav jednotlivých častí systému
- `sparql-start.sh/sparql-stop.sh` – skripty slúžiace na spustenie alebo zastavenie služby konverzie extrakcií do RDF za použitia naivného prístupu

Systém tiež obsahuje skripty `/src/openie-start.sh` a `/src/openie-stop.sh` na paralelné spustenie, resp. zastavenie všetkých častí realizujúcich jednotlivé kroky extrakčného systému.

A.5 Nastavenia systému

Nastavenia systému je možné meniť v súbore `/src/openie.conf`. V týchto nastaveniach je možné využívať aj premenné prostredia ako napríklad `$HOSTNAME` umožňujúce flexibilné spravovanie vo výpočtovom klastri. Význam jednotlivých nastavení je nasledujúci:

- `FrontendPort` – číslo portu, na ktorom je možné pristupovať k front-endu, adresa je daná samotným názvom stanice, na ktorej bol front-end spustený
- `DatabaseHost` – adresa stanice, na ktorej bola spustená služba *4s-httpd*
- `DatabasePort` – port, na ktorom bola spustená služba *4s-httpd*
- `DatabaseGraph` – názov *4store* klastra, do ktorého budú ukladané extrakcie
- `OpeniePath` – cesta k extraktoru *Open IE*
- `BllipModelPath` – cesta k natrénovaným modelom pre syntaktický analyzátor *BLLIP*
- `NERhost` – adresa, na ktorej je prístupná služba *Decipher NER*
- `NERport` – port, na ktorom je prístupná služba *Decipher NER*
- `SkipDerivation` – hodnota `1` pre zapnuté rozgenerovávanie fráz obsahujúcich priraďovacie spojky, hodnota `0` pre preskočenie tohoto kroku
- `SparqlPrefix` – názvoslovie využívané pri generovaní RDF reprezentácie
- `SparqlGraph` – názov RDF grafu, do ktorého budú ukladané extrakcie
- `ArticlesDirectory` – cesta k priečinku, ktorý obsahuje výsledky jednotlivých krokov spracovania
- `LogsDirectory` – cesta k priečinku, v ktorom budú vytvárané záznamy o priebehu jednotlivých krokov spracovania

A.6 Použitie systému

Príklad použitia implementovaného extrakčného systému je uvedený v nasledujúcej časti. Postup, ktorý je popísaný, prestavuje vykonanie extrakcie na výpočtovom klastri a konverziu do formátu *Turtle* s využitím perzistentnej vyrovnávacej pamäte *ZODB*. Jednotlivé kroky postupu testované na referenčných strojoch výpočtového klastra, na ktorom bol systém vyvíjaný, sú:

1. nainštalovanie potrebných súčastí uvedených v `/src/README.txt`
2. získanie najnovšieho výpisu *Wikipédie* napríklad pomocou skriptu `/src/tools/obtain_wikiplaintext.sh`
3. konverzia *XML* formátu výpisu *Wikipédie* na čistý text pomocou skriptu `/src/tools/wikiparser.v4.py`
4. nastavenie konfigurácie systému v `/src/openie.conf`
5. spustenie extrakčného systému pomocou skriptu `/src/openie-start.sh`
6. po spracovaní všetkých článkov ukončenie extrakčného systému pomocou skriptu `/src/openie-stop.sh`
7. získanie extrakcií z výpočtového klastra na jednu stanicu pomocou skriptu `/src/bash_scripts/get-ner-tar.sh`
8. skonvertovanie týchto extrakcií do formátu *Turtle* pomocou skriptu `/src/python_scripts/sparql_create_all.py`
9. vytvorenie nového *4store* klastra a spustenie služby *4s-httpd* pomocou skriptu `/src/bash_scripts/4s-start.py --redeploy`
10. vloženie získaných *Turtle* súborov do *4store* databázy pomocou nástroja `4s-import`
11. spustenie front-endu pomocou `/src/bash_scripts/frontend-start.sh`
12. prístup k front-endu prostredníctvom bežného webového prehliadača

Príloha B

Podrobnosti k nameraným hodnotám pri experimentovaní na dátach anglickej Wikipédie

	Server	Počet súborov	Priemerný čas	Max. čas	Veľkosť
Koreferen.	1.	39893	4,7478s	85s	322MB
	2.	39965	4,6766s	84s	293MB
	3.	39699	3,8483s	63s	354MB
	4.	39638	4,6874s	84s	431MB
	5.	39733	4,6778s	85s	402MB
Extrakcia	1.	39895	1,3972s	114s	352MB
	2.	39969	0,6646s	108s	300MB
	3.	39702	0,7149s	290s	381MB
	4.	39641	0,9388s	311s	466MB
	5.	39893	4,7478s	73s	437MB
Analýza	1.	39654	4,5800s	109s	556MB
	2.	39942	4,5436s	85s	513MB
	3.	39597	3,9206s	68s	587MB
	4.	39537	4,5216s	84s	739MB
	5.	39635	4,8356s	85s	687MB

Tabuľka B.1: Dosiahnuté priemerné časy a maximálny nameraný čas spracovania jedného článku. Počet súborov je uvádzaný vzhľadom na predchádzajúci krok, pričom pôvodný počet bol presne 40000 súborov. Veľkosť predstavuje celkovú veľkosť súborov vytvorených v danom kroku.