Vysoká škola báňská - Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Extrakce zpráv z webových stránek

News Extraction from Web Pages

2014 Bc. Štefan Blanár

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Štefan Blanár**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Extrakce zpráv z webových stránek
News Extraction from Web Pages

Zásady pro vypracování:

Cílem práce je provedení průzkumu existujících přístupů v oblasti automatické extrakce zpráv z webových stránek, návrh a implementace vybrané nebo vlastní metody a aplikačního prostředí pro experimenty.

1. Průzkum a popis existujících přístupů.
2. Návrh a implementace vybrané nebo vlastní metody.
3. Návrh a implementace počítačové aplikace pro provádění experimentů.
4. Návrh, realizace a hodnocení experimentů.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Miloš Kudělka, Ph.D.**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Chcel by som sa týmto poďakovať môjmu vedúcemu diplomovej práce
Mgr. Milošovi Kudělkovi, Ph.D.za jeho trpezlivosť, čas, úsilie, ochotu, pomoc a dôležité rady, ktorými mi pomohol pri vypracovaní tejto diplomovej práce.

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 06.05.2014 .....

Abstrakt

V této diplomové práci jsem se zabýval problematikou extrakce zpráv z webu, což je problém tématicky spadající pod dobře známý problém dolování strukturovaných dat z HTML dokumentů na Internetu. Vypracoval jsem průzkum různých stávajících přístupů k tomuto problému, který je shrnut na začátku této práce. Dále jsem se věnoval zkoumání stávajících wrapperů a jejich možných uplatnění při řešení problému extrakce zpráv z webu. Také jsem vypracoval rozsáhlé pozorování nejznámějších zpravodajských portálů a zpráv na nich. Poté jsem získané poznatky aplikoval při tvorbě vlastních řešení tohoto problému. Definoval jsem, co je zpráva a jak se liší od informace. Vlastní řešení jsem následně otestoval v reálných podmínkách na skutečných, dobře známých zpravodajských webových portálech. Výsledky tohoto testování jsou prezentované v závěru práce.

Klíčová slova

text mining, regulérní výraz, extrakce, zpráva, Internet, web, URL, metoda, algoritmus, ReLIE, ONTEA, DOM, XML, HTML, XPath, MDF, TPC, NCSCA, TTR, wrapper, crawler, automatický wrapper, poloautomatický wrapper, klíčová slova, schéma

Abstract

The main goal of this diploma thesis is to perform large – scale research about text mining methods especially text mining of structured data from web, concrete from HTML documents, what is well-known problem. Results of this research will be summarized in fist part of this document. Next I probe a few web wrapper's, especially I'll try to find some existing wrapper, which could be used as solution for extraction news from web. I also perform an extensive observation of the most famous news portals and news on them. Finally acquired knowledge will be used for developing my own solution of problem extraction news from web pages. I'll define what web news is and how they differs from information. Then I test my solution in real conditions on real well known news portals. All results of this testing will be presented in last chapter of this thesis.

Keywords

text mining, regular expression, extraction, news, Internet, Web, URL, method, algorithm, ReLIE, ONTEA, DOM, XML, HTML, XPath, MDF, TPC, NCSCA, TTR, wrapper, crawler, automatic wrapper, semi-automatic wrapper, keywords, scheme

Slovník použitých pojmov, skratiek a symbolov

W3C – *World Wide Web Consortium* – konzorcium pre dohľad a špecifikáciu štandardov a odporučení pre tvorbu webu
TM – *text mining* – dolovanie textu. Odvetvie výskumu v informatike zaoberajúce sa ziskom a spracovaním textových dát z webu
DP – diplomová práca
reg. – regulárny / regulárna / regulárne
regex – regulárny výraz
ReLIE – ***R****egular* ***e****xpression* ***L****earning for* ***I****nformation* ***E****xtraction* – metóda samoučenia regulárnych výrazov
URL – ***U****niform* ***R****esource* ***L****ocator* **–** adresa zdroja na webe, je súčasťou URI
URI – ***U****niform* ***R****esource* ***I****dentifier* **–** kompletný identifikátor zdroja(dokumentu) v prostredí Internetu
atď. – a tak ďalej
urč. – určitý, určitá, určité
ONTEA – ***ON****tology based* ***TE****xt* ***A****nnotation* – extrakčná metóda zaoberajúca sa štruktúrou, funguje na princípe sémantického webu, pridáva sémantiku, tam kde nie je na základe regulárnych výrazov
tzv. – takzvaný / takzvaná / takzvané
t.j. – to jest, to je
DOM – ***D****ocument* ***O****bject* ***M****odel* – objektovo orientovaná reprezentácia štruktúry dokumentu
XML – ***E****xtensible* ***M****arkup* ***L****anguage* – je špeciálny jazyk využívajúci párových značiek k uchovávaniu dát. V užšom slova zmysle pod týmto pojmom rozumieme súbor pravidiel pre tvorbu takýchto dokumentov špecifikovaný v W3C.
HTML – ***H****yperText* ***M****arkup* ***L****anguage* – hypertextový značkovací jazyk určený na vytváranie webových stránok
XPath – ***X****ML* ***Path*** *Language* – je jazyk slúžiaci na výber uzlov v XML dokumentoch
MDF – ***M****ining* ***D****ata* ***R****ecords* – extrakčná metóda zaoberajúca sa extrakciou záznamov z webu
cluster, **clustre**, **clustrov** – *cluster, clusters* – informácie zhlukované, zhromažďované do skupín na základe určitých podobností
TPC – ***T****ag* ***P****ath* ***C****lustering* – extrakčná metóda založená na zhlukovaní elementov do „clustrov“ na základe analýzy absolútnych ciest k nim(XPath)
NCSCA – ***N****ormalized* ***C****ut* ***S****pectral* ***C****lustering* ***A****lgorithm* – algoritmus zabezpečujúci klasifikáciu vizuálneho signálu a vytváranie „clustrov“
TTR – ***T****ext to****T****ag* ***R****atio* – metóda extrakcie textov na základe dĺžky textu v elementoch
DoC – ***D****egree* ***of*** ***C****oherence* – stupeň spojitosti vizuálne oddelených blokov
PDoC – ***P****ermitted* ***D****egree* ***of*** ***C****oherence* – preddefinovaný stupeň členitosti
YQL – ***Y****ahoo!* ***Q****uery* ***L****anguage* – dotazovací jazyk na dolovanie textu [1]
MSDN – ***M****icro****s****oft* ***D****eveloper* ***N****etwork* – informáčné fórum firmy Microsoft pre developerov využívajúcich platformu .NET
TTL – ***T****ime* ***t****o* ***l****ive* – počet krokov do ukončenia
wrapper – z anglického slova obálka, program vyhľadávajúci špecifikované bloky HTML dokumentu a extrahujúci ich obsah
crawler – z anglického slova plaziť sa, preliezať, program / časť programu zabezpečujúca prechod a navigáciu medzi dokumentmi na Internete
crawlovanie – prechod dokumentmi prostredníctvom hypertextu
semi – automatický – *semi*(polovične), poloautomatický

CSS – skratka z anglického ***C****ascading* ***S****tyle* ***S****heets*, spôsob definovania vzhľadu stránok v oddelenom dokumente
pagelet – časť HTML dokumentu, ktorá je nositeľom obsahu, mení sa a nie je uniformná v jednotlivých dokumentoch
tag – párová / nepárová značka v jazyku HTML, označuje jeden element jazyka (* *, *</br>*, atď.)
klastrovanie, klastracia, clustering – zhlukovanie dát do celkov na základe určitých podobností
metadáta – tzv. dáta o dátach, slúžia najčastejšie na strojové spracovanie, obsahujú dodatočnú informáciu o význame – sémantike dát
NLP – ***N****ature* ***L****anguage* ***P****rocesing*, problematika spracovania a analýzy prirodzeného jazyka
host – časť URL odpovedajúca hostiteľskému serveru
ban – zákaz, znemožnenie činnosti
sub. – pod, na nižšej úrovni, pod úroveň
kap. – kapitola

Obsah

1. Úvod

S rozvojom Internetu je stále viac textových informácií obsiahnutých na tomto médiu a klesá tak potreba tlačených médií či už sa jedná o knihy, magazíny alebo denníky. Každoročne značne upadá percento predajnosti tlačených masových médií a zároveň vzrastá popularita ich elektronickej podoby. Na Internete sa nachádza množstvo rôznych textových informácií, či už to sú odborné štúdie, elektronické publikácie rôzneho druhu, príspevky ľudí, novinové články a pod. To kladie nároky na spracovanie, triedenie a využívanie takýchto informácií. Získavaním a spracovaním textových informácií z Internetu sa zaoberá metóda dolovania textových informácií z webu(tzv. text mining ďalej len TM). TM je dobre známym a často študovaným problémom Informatiky.

V tejto diplomovej práci sa budem zaoberať problémom extrakcie správ z webových stránok. Na základe získaných vedomostí štúdiom problému TM navrhnem optimálnu metódu na zisk a spracovanie správ z webu a danú metódu implementujem. Výslednú aplikáciu podrobím testovaniu. V druhej kapitole tejto diplomovej práce(ďalej len DP.) som sa zapodieval problémom TM, študoval som výsledky rôznych verejne dostupných výskumov v tejto oblasti a popísal som stručnú rekapituláciu toho, čo som sa dozvedel. V tretej kapitole sa venujem aplikácií zistených poznatkov z kapitoly jedna a dva na problém extrakcie správ z webu. Definujem, čo je to správa a ako sa odlišuje od informácie. Taktiež vykonávam empirické skúmanie správ a spravodajských serverov na Internete. V štvrtej a piatej kapitole sa zaoberám implementáciou vybraných metód a ich testovaním v reálnych podmienkach na najznámejších českých a slovenských spravodajských serveroch.

1.1 Motivácia

Na Internete existuje množstvo spravodajských serverov. Každý z nich prináša rozličné informácie či už z dôvodu zamerania serveru, jeho ovplyvnenia(napr. vlastníkom alebo sponzorskými skupinami) alebo z dôvodu kapacity. Pokiaľ chce užívateľ alebo urč. skupina ľudí skutočne objektívne správy potrebuje sledovať správy na viacerých serveroch. Z toho dôvodu vznikajú nároky na extrakciu správ z webu. Ďalším dôvodom je marketingové využitie. Správy z webu sú voľne dostupné a je ich možné extrahovať a ďalej využívať. Záleží však na účele ich využívania(autorský zákon - č. 618/2003). Je možné si napríklad založiť vlastný spravodajský server bez nutnosti zamestnávania redaktorov. Správy budú extrahované z webu a na svoj server sa umiestni len ich prehľad s odkazmi na správy. Najznámejším takýmto serverom je napr. https://news.google.com/.

1.2 Ciele DP

1) Získanie poznatkov štúdiom problému TM, ich zosumarizovanie a prezentácia
2) Využitie získaných poznatkov a ich aplikácia na problém extrakcie správ z webových dokumentov
3) Uplatnenie daných poznatkov pri výbere metód pre extrakciu správ z webu
4) Implementácia vybraných metódy z bodu 3)
5) Testovanie implementovaných metód v reálnych podmienkach na najznámejších českých a slovenských spravodajských serveroch, následná prezentácia výsledkov

1.3 Extrakcia správ z webu a extrakcia textových dát z webu

Problém extrakcie správ z webových stránok veľmi úzko súvisí s problémom extrakcie textových dát z webu(text extraction from web). Týmto problémom sa zapodievalo veľmi mnoho výskumov a napriek tomu je neustále predmetom skúmania. Zapríčiňuje to nesmierna rozmanitosť webu, rôzne štýly tvorby webu a v neposlednej rade nesmierna obsiahlosť Internetu a komplikovanosť prirodzeného jazyka v dnešnej dobe. Aj napriek snahe W3C[1] existuje mnoho zle formátovaných webových prezentácií(kríženie tagov, nedodržiavanie štandardov, neaktuálnosť a iné), ktoré túto úlohu ešte viac komplikujú.

[1] http://www.w3.org/

2. Text mining – extrakcia textových dát z webu

Text mining je analýza dát obsiahnutých v textoch v prirodzenom jazyku. Využitie TM techník v praxi v aplikáciách na riešenie rôznych problémov sa nazýva **textová analýza.** TM slúži na zisk potencionálne cenných a využiteľných informácií z akýchkoľvek spracovateľných textových zdrojov ako sú napr. rôzne textové dokumenty, emaily, HTML stránky alebo prípadne len ich časti (posty na soc. sieťach a pod.). Zisk a spracovanie textových informácií z webu nazývame „*information extraction from web*“. Hlavným cieľom tejto disciplíny je extrahovať štruktúrované textové informácie z neštruktúrovaných alebo čiastočne štruktúrovaných zdrojov(HTML).V súčasnosti sa drvivá väčšina metód využívaných pri získavaní textových informácií z webu sústredí na špecifické významové domény alebo štruktúry. To je spôsobené tým, že nie je možné navrhnúť postup, ktorý by vedel vykonať plne automatickú, univerzálnu a zároveň efektívnu extrakciu dát z rôzne štruktúrovaných zdrojov.

2.1 Prehľad niektorých extrakčných metód a algoritmov

2.1.1 Extrakcia informácií pomocou regulárnych výrazov

Veľa extrakčných úloh môže byť úspešne vyriešených použitím regulárnych výrazov. Dobrým príkladom entít, ktoré sa dajú touto metódou ľahko získať sú napr. emailové adresy, čísla kreditných kariet, čísla účtov, telefónne čísla, rôzne geografické údaje(napr. zemepisná šírka, dĺžka), letopočty, URL a mnoho ďalších. Tieto entity majú niekoľko spoločných kľúčových prvkov, ktoré sa dajú vyjadriť ako štandardný regulárny výraz. Použitie regulárnych výrazov na zisk textových informácií nie je vždy jednoduché a na vytvorenie žiadaného regulárneho výrazu je nutné mať dôkladne naštudovanú cieľovú oblasť.

Pár príkladov regulárnych výrazov [2]:

Email: `^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$`
Zjednodušený regex pre IPv4: `\b(?:\d{1,3}\.){3}\d{1,3}\b`
URL: `href\\s*=\\s*(?:\"(?<1>[^\"]*)\"|(?<1>\\S+)`

Regulárne výrazy je možné vopred manuálne definovať, v takom prípade hovoríme o manuálnej alebo semi – automatickej extrakcii, alebo tiež využiť automatické samoučiace[3] metódy, ktoré zvládnu tvorbu regulárnych výrazov za nás. Zo samoučiacich algoritmov za zmienku stojí napr. ReLIE [2] (Regular expression Learning for Information Extraction), ktorému sa bližšie budem venovať v kapitole 2.1.1.1. Využívanie regulárnych výrazov pri extrakcii textových dát je v súčasnosti veľmi rozšírené a populárne. Existuje mnoho metód, ktoré s nimi pracujú a takmer všetky metódy TM aspoň čiastočne na rôzne menšie úlohy využívajú mechaniku regulárnych výrazov.

2.1.1.1 ReLIE(Regular expression Learning for Information Extraction)[4]

Je samoučiaca metóda, slúžiaca k transformácií vstupného regulárneho výrazu. Daný regulárny výraz bude transformovaný resp. „vylepšovaný“ pomocou pevne daných transformačných

[2] Všetko o regulárnych výrazoch nájdete na [20]
[3] Viac o samoučiacich algoritmoch a základných metódach TM [21] a [22]
[4] Informácie v tejto kapitole čerpajú z [2]

pravidiel[5] definovaných v tejto metóde na základe množiny validačných dát, ktoré manuálne zadáme a budú ukazovať, čo má daný regex považovať za pozitívnu zhodu. K tomu aby mohlo k tejto transformácií vôbec dôjsť budeme potrebovať tréningovú množinu dát na ktorej sa bude algoritmus „učiť".

Zjednodušený popis ReLIE

1. Rozlišujú sa pozitívne a negatívne výstupy extrakcie pre daný regex.
2. Aplikujú sa disjunktívne pravidlá na spresnenie daného regexu.
3. Použijú sa slovníky s negatívnymi výrazmi, ktoré sú napĺňané „*greedy*" algoritmami.
4. Aplikujú sa obmedzenia pre triedy znakov a kvantifikátory.
5. Vyhľadávanie (na princípe „*Hill climbing*[6]" algoritmu): v každej iterácií sa aplikuje niekoľko transformácií pre získanie kandidátskej množiny regexov.
6. Z kandidátskej množiny je vybraný regex s max. hodnotou objektívnej funkcie na tréningovej množine

Pseudokód 2.1 Vyhľadávanie v ReLIE

Procedure `ReLIE` ($M_{tr}, M_{val}, R_0, \mathcal{T}$)
// M_{tr} : $set\ of\ labeled\ matches\ used\ as\ training\ data$
// M_{val} : $set\ of\ labeled\ matches\ used\ as\ validation\ data$
// R_0 : $user-provided\ regular\ expression$
// $\mathcal{T}$: set of transformations
begin
1. $R_{new} = R_0$
2. **do {**
3. **for** each transformation $t_i \in \mathcal{T}$
4. Candidate $_i$ = `ApplyTransformations(`R_{new}, t_i`)`
5. **let** Candidates $= \bigcup_i$ Candidate $_i$
6. **let** $R' = argmax_{R \in \text{Candidates}}\ \mathcal{F}(R, M_{tr})$
7. **if** ($\mathcal{F}(R', M_{tr}) <= \mathcal{F}(R_{new}, M_{tr})$) **return** R_{new}
8. **if** ($\mathcal{F}(R', M_{val}) < \mathcal{F}(R_{new}, M_{val})$) **return** R_{new}
9. $R_{new} = R'$
10. **} while (true)**

end

Príklad: Nájdi optimálny regulárny výraz zachytávajúci tel. čísla
Vstup: tréningová množina dát, validačná množina obsahujúca tel. čísla rôznych formátov{0910-425-255,0910 425 255, 0910.425.255, ...}, najjednoduchší regex schopný nájsť aspoň niektoré tel. čísla : "`(\d+\-)+\d+`"
Aplikuje sa postup uvedený vyššie.
Výstup: transformovaný regex: „`(\d{3}[-.\ ()]){1,2}[\dA-Z]{4}.`"

ReLIE slúži k automatickej transformácií a optimalizácií vstupného nedokonalého regulárneho výrazu na jeho dokonalejšiu verziu. Môže byť použitý v prípade, že užívateľ nie je znalý problematiky regulárnych výrazov a potrebný regulárny výraz vytvorí algoritmus. Táto metóda nemusí byť použitá len k optimalizácií jednoduchých regulárnych výrazov popisujúcich

[5] Jedná sa o disjunktívne transformácie na spresnenie regexu detailne [2]
[6] Viac o vyhľadávaní na princípe „*Hill Climbingu*" sa nachádza na [3].

tel. čísla, URL, atď., ale môže byť využitá aj na vytvorenie regulárneho výrazu schopného extrahovať z cieľového zdroja štruktúrované dáta(napr. obsah niektorých „*tagov*“ v HTML dokumente, všetky mená v dokumente a tak vytvoriť určitú kartotéku, atď.). Použitie tejto metódy je **výhodné** v prípade, že užívateľ nie je znalý regulárnych výrazov a dáta, ktoré chce extrahovať sú pomerne jednoducho extrahovateľné a ich štruktúra je opísateľná zloženým regulárnym výrazom. Dobrým príkladom, kedy je výhodné túto metódu použiť je napr. extrakcia všetkých mien, dátumov, mailov, URL a rôzne iné z dokumentu.

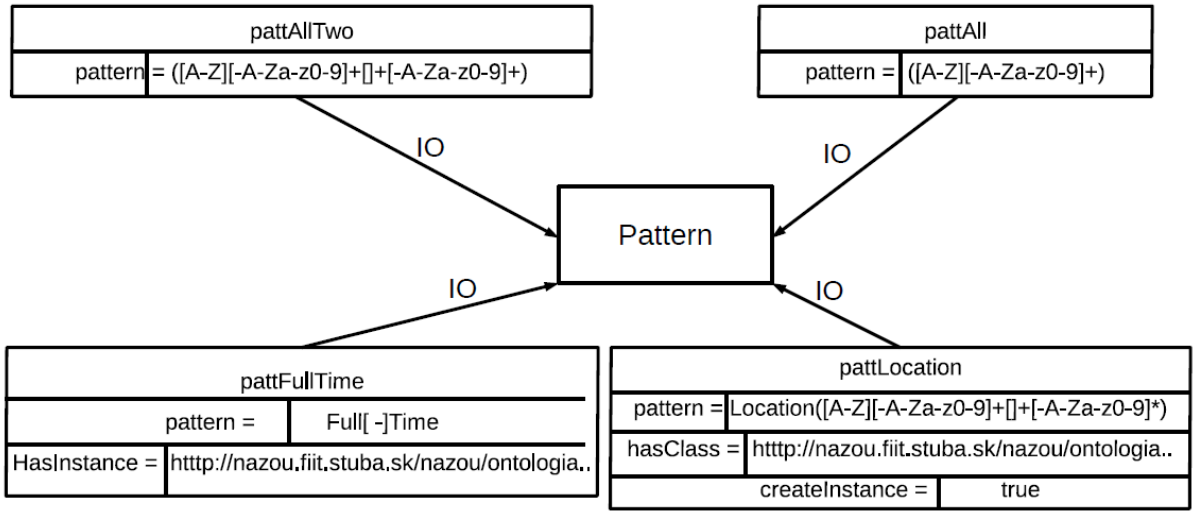
Značnými **nevýhodami** sú:

- potrebnosť obsiahlej tréningovej množiny dát
- množina validačných dát(čím zložitejšia štruktúra tým musí byť rozsiahlejšia, v prípade zložených regexov dokonca $n_1 \, x \, \ldots x \, n_m$, kde n je počet rôznych formátov daných dát a m je počet zložiek výsledného reg. výrazu.)
- nemožnosť extrakcie zložito štruktúrovaných dát, plynie z predchádzajúcej nevýhody

2.1.1.2 ONTEA (ONtology based TExt Annotation)

Je platforma, ktorá umožňuje generovať sémantické metadáta na základe vzorov [4]. Pracuje nad textom príslušným k nejakej aplikačnej doméne, analyzuje dokument alebo text použitím regulárnych výrazových vzorov a detekuje rovnaké sémantické elementy vzhľadom na definovanú doménovú ontológiu[7]. Výrazový vzor, popisujúci určité dáta, ktoré chceme spracovávať sa skladá z dielčích regulárnych výrazov, ktoré popisujú jednotlivé časti resp. dáta.

Obrázok 2.1 Výrazový vzor("pattern") príklad



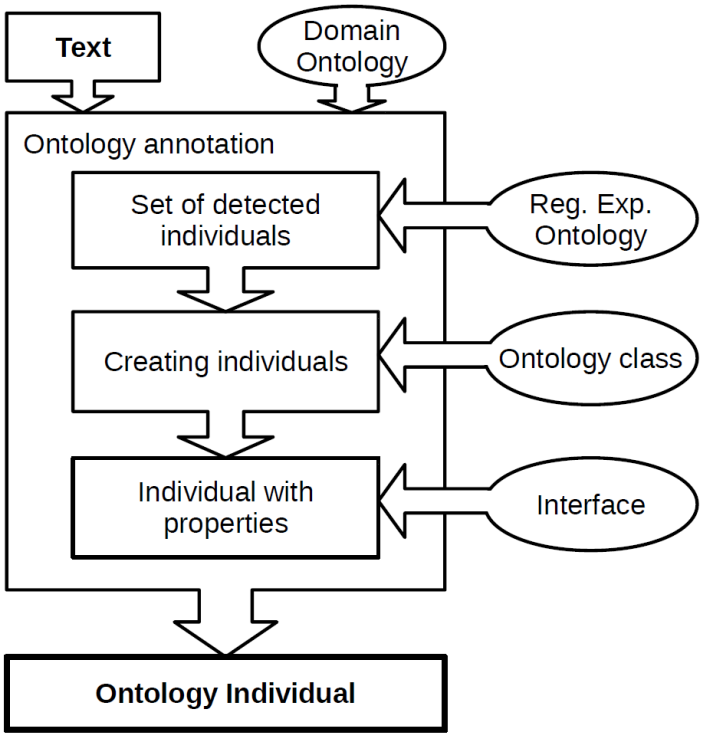
Zásadný zjednodušený princíp Ontea algoritmu

1. Text dokumentu sa načíta.
2. Text je spracovaný definovanými regulárnymi výrazmi a ak našli korešpondujúcu ontológiu jedinca podľa zvyšku vlastností vzorov tak je pridaný do množiny nájdených jedincov ontológie.
3. Ak žiadny jedinec nebol nájdený pre rovnaký vzor a vlastnosť je vytvorený nový jedinec iba s vlastnosťou „*label*“ obsahujúci zhodný text.

[7] V informatike je chápaná ako sémantika, resp. štruktúra, prípadne má aj význam ako popisovanie

4. Takýto proces sa opakuje pre všetky regulárne výrazy a výsledkom je množina nájdených jedincov.
5. Prázdny jedinec triedy reprezentujúci spracovávaný text je vytvorený a všetky možné vlastnosti takejto triedy ontológie boli objavené z definície triedy.
6. Detekovaný jedinec je porovnávaný s vlastnosťami typu a ak vlastnosť typu je ten istý ako jedinec typu (triedy), tak taký jedinec je priradený ako vlastnosť.
7. Také porovnávanie sa robí pre všetky vlastnosti nového jedinca korešpondujúceho s textom/dokumentom, rovnako pre všetkých detegovaných jedincov.

Obrázok 2.2 Základný princíp popisujúci algoritmus Ontea graficky



Príklad[8]:
Vstup: text, vzory(patters) zložené z reg. výrazov a texty určené na analýzu

Tabuľka 2.1 Vstup pre algoritmus ONTEA, texty a skupina reg. výrazov reprezentujúca vzor

Príklad	Text	Vzory – regulárne výrazy
1	Michal Laclavik Works for Slovak Academy Of Sciences located in Bratislava, the capital of Slovakia	`\\b(\\p{Lu}[a-z]+ +\\p{Lu}[a-z]+|\\p{Lu}[a-z]+)\\b` *Person*
2	Automobilka KIA sa rozhodla investovať pri Žiline, kde vybudovala svoju prvú továreň V Európe	`\\b([\\p{Lu}][-&\\p{L}]+[]*[-&\\p{L}]*[][-&\\p{L}]*)[,]+s\\.r\\.o\\.` *Organization*
	Kia Motors Slovakia, s.r.o. P.O.Box 2, 01301 Teplička nad Váhom Slovakia	`\\b[0-9]{3}[]*[0-9]{2} +(\\p{Lu}[^\\s,.]+[]*[^0-9\\s,.]*[]*[0-9\\n,]+)` *Settlement* `(v|pri)+(\\p{Lu}\\p{L}+)` *Location*

[8] Uvedený príklad je z [23] a slúži na lepšie pochopenie funkcionality algoritmu nástroja Ontea

Výstup: extrahované konkrétne informácie na základe zadaných reg. výrazov s priradenou dodatočnou informáciou o sémantike daných dát.

Tabuľka 2.2 Príklad výstupu algoritmu Ontea s pridanou lematizáciou

Príklad	Metóda	Vzory – regulárne výrazy
1	Ontea	Michal Laclavik *Person* Slovak Academy *Organization* Bratislava *Settlement* Slovakia *Country*
2	Ontea create	Žiline *Location* Európe *Location* Kia Motors Slovakia *Organization* Teplička nad Váhom *Settlement*
2	Ontea create, lematizácia	Žilina *Location* Európa *Location* Kia Motors Slovakia *Organization* Teplička nad Váh *Settlement*

Ontea je extrakčná metóda založená na vzoroch zložených z jednotlivých regulárnych výrazov. Prehľadáva text a hľadá inštancie jednotlivých reg. výrazov. V prípade zhody priradí sémantický význam nájdeným dátam. Táto metóda je založená a má mnoho spoločného s metódami a prístupmi k spracovaniu dokumentov ako sú CREAM [5] , Magpie [6], Annotea[9], Rubby[10] alebo RDF anotácia[11] Podobne ako pri ReLIE je vhodné túto metódu používať na extrakciu jednoduchých nie príliš štruktúrovaných dát, ktoré sa dobre dajú popísať regulárnym výrazom. Na rozdiel od ReLIE je nutná pomerne nadštandardná znalosť problematiky reg. výrazov a ich tvorby pre rozličné účely. Taktiež nepotrebuje tréningovú a validačnú množinu dát, tak ako tomu bolo u ReLIE, keďže vzory sú zadávané manuálne.

Najvýhodnejšie je použiť túto metódu na často sa opakujúce dáta charakteristického tvaru. Štandardne sa využíva na vyhodnocovanie rozličných ponúk v rámci určitej domény (pracovné ponuky a pod.). Taktiež sa dá použiť na transformáciu nesémantického webu na sémantický, s tým rozdielom, že dodatočná informácia zistená týmto algoritmom bude vpísaná do „*tagov*“ ako metadáta, namiesto výstupných sémantických textových dát tak vznikne sémantický web.

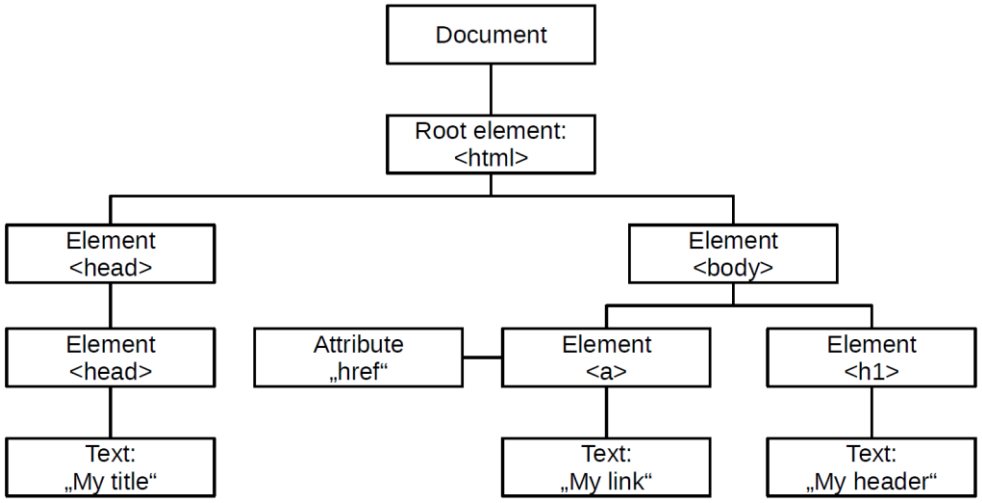
Nevýhody:
- Manuálna tvorba reg. výrazov
- Obmedzenia plynúce z použitia reg. výrazov(štruktúrované dáta, atď.)
- Veľmi problematické spracovanie opakujúcich sa entít a mnohonásobných entít[i]

[9] http://www.w3.org/2001/Annotea/
[10] http://www.w3.org/TR/ruby/
[11] http://ilrt.org/discovery/2001/04/annotations/

2.1.2 Extrakcia informácií na základe porovnávania štruktúry dokumentu

Tzv. porovnávacia metóda, je metóda extrakcie dát z HTML dokumentov s rovnakou štruktúrou(šablónou) a meniacim sa obsahom, ktorý je nositeľom informačnej hodnoty daného dokumentu. Hlavným princípom tejto metódy je porovnávanie DOM[12](Document Object Model) stromovej štruktúry jednotlivých dokumentov(HTML stránok). Tento algoritmus rekurzívne alebo iteratívne prechádza stromy dvoch alebo viacerých dokumentov a porovnáva absolútne cesty k jednotlivým elementom dokumentu(najčastejšie *XPath*). V prípade, že sa v rovnakom elemente resp. elemente s rovnakou cestou nachádza rôzny obsah, tak si tento obsah zapamätá.

Obrázok 2.3 Príklad DOM jednoduchého HTML dokumentu



Zjednodušený popis algoritmu porovnania DOM a následnej extrakcie

1. Na vstupe máme dva alebo viac HTML alebo XML dokumentov
2. Rekurzívne (rekurzívny prechod stromom) alebo iteratívne (element po elemente) porovnávame vnútorné obsahy jednotlivých elementov príslušných dokumentov
3. V prípade, že sme našli dva alebo viac elementov s odlišným obsahom, považujeme ich za nositeľov informácie
4. Uložíme si buď obsah daných elementov alebo cestu k danému elementu
5. Na konci prejdem nájdenými elementmi a vrátim ich obsah

Príklad: Porovnaj dva telefóny z katalógu
Vstup: dva HTML dokumenty s údajmi o daných telefónoch
Výstup: množina rozdielov v daných vnútorných textoch dokumentov zvýraznená na obrázku[13] uvedenom nižšie.

[12] http://www.w3.org/DOM/
[13] porovnanie vykonané na https://www.telekom.sk na nekomerčné využitie

Obrázok 2.4 Príklad výstupu algoritmu porovnania DOM štruktúry

	Sony Xperia™ SP	Samsung Galaxy S4 mini Black
		
	Do košíka	Do košíka
Základné informácie		
Celková hmotnosť	155 g	107 g
Rozmery (výška x šírka x hrúbka)	130,6 x 67,1 x 9,98 mm	124,6 x 61,3 x 8,94 mm
Displej		
Uhlopriečka	4.6"	4.3"
Rozlíšenie	1280 x 720 pixelov	540 x 960 pixelov
Typ displeja	dotykový	dotykový, qHD sAMOLED
Farba	čierna	čierna
Zhotovenie krytov telefónu	plast	plast
SAR (vyžarovanie zariadenia)	1,01 W/kg (hlava); 0,90 W/kg (telo)	--
Typ SIM karty	micro SIM (3FF)	micro SIM (3FF)
Typ	dotykový	dotykový
Technické parametre		
Procesor	1,7 GHz (Qualcomm MSM8960Pro); Grafický procesor Adreno 320	--
Operačná pamäť	1 GB	--
Celková zdieľaná interná pamäť	8 GB (5,05 GB pre užívateľa)	8 GB
Podporovaná pamäťová karta	MicroSD, max. 32 GB	Micro SD, max. 64 GB
Operačný systém	Android 4.1	Android 4.2

Z príkladu je jasné, že túto metódu je možné využívať len v prípade úplnej zhodnosti štruktúry vstupných dokumentov meniacich sa len v obsahu. Ak sa budú dokumenty líšiť v štruktúre, záleží na implementácií či dáta v nezhodných elementoch budú ignorované, budú pridané do výsledku alebo zapríčinia chybový stav programu. Táto metóda extrahuje len rozdielne informácie v daných dokumentoch. Z toho vyplýva, že **najvýhodnejšie** je túto metódu využívať na porovnávanie položiek rôznych katalógov.

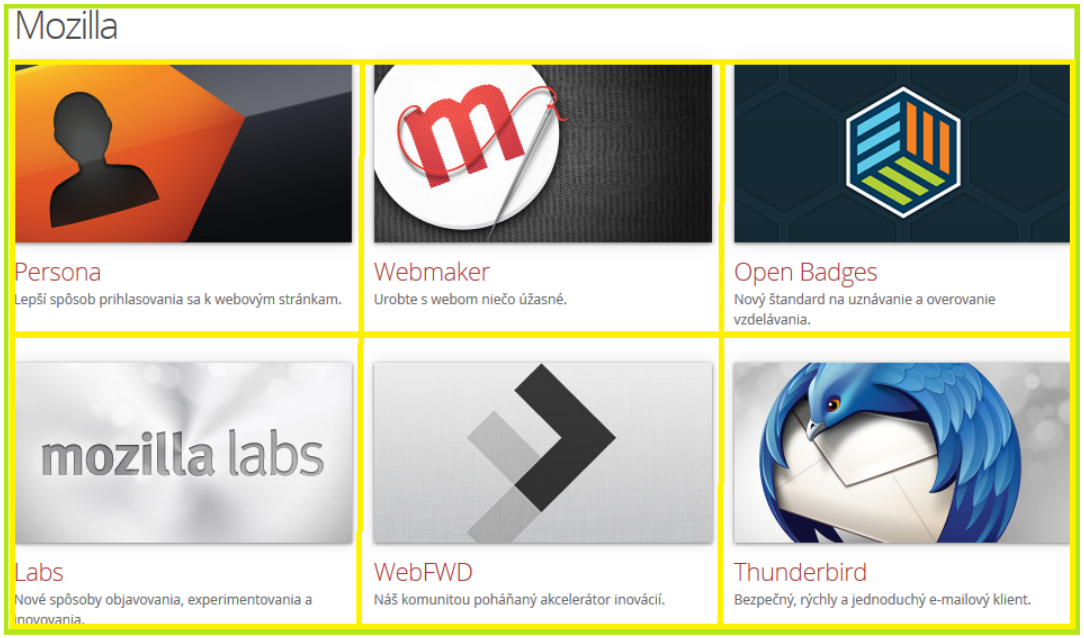
Nevýhody:

- Zachytáva len rozdielne časti dokumentu
- Potrebuje pomerne obsiahlu množinu dokumentov, aby dodala presné výsledky, to vyplýva z predchádzajúcej nevýhody
- Je silne závislá na štruktúre dokumentu(množstvo chýb a nepresností napr. pri krížení „*tagov*“, ktoré je veľmi časté)

2.1.3 Extrakcia podobných záznamov na základe cesty k nim

Existuje množstvo stránok, ktoré obsahujú záznamy rovnakého typu. Najčastejšie sú to elektronické obchody prezentujúce svoju ponuku ako obsiahle „product listy“ alebo sú to rozličné katalógy či porovnávacie servery, ktoré prezentujú rôzne ponuky vzťahujúce sa na určitú doménu. Taktiež existuje pár prístupov a algoritmov, ktoré sa zaoberajú extrakciou dát z dokumentov obsahujúcich takéto záznamy.

Obrázok 2.5 Príklad tzv. "product listu", t.j. niekoľko záznamov rovnakého typu v tesnej blízkosti u seba

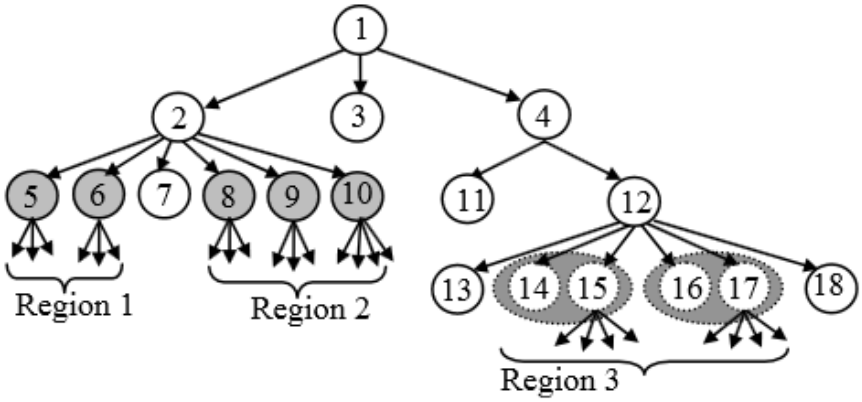


Na obrázku element rodič(div zelený obrys) a rôzne záznamy, jeho listy („*product list*“ žltý obrys)

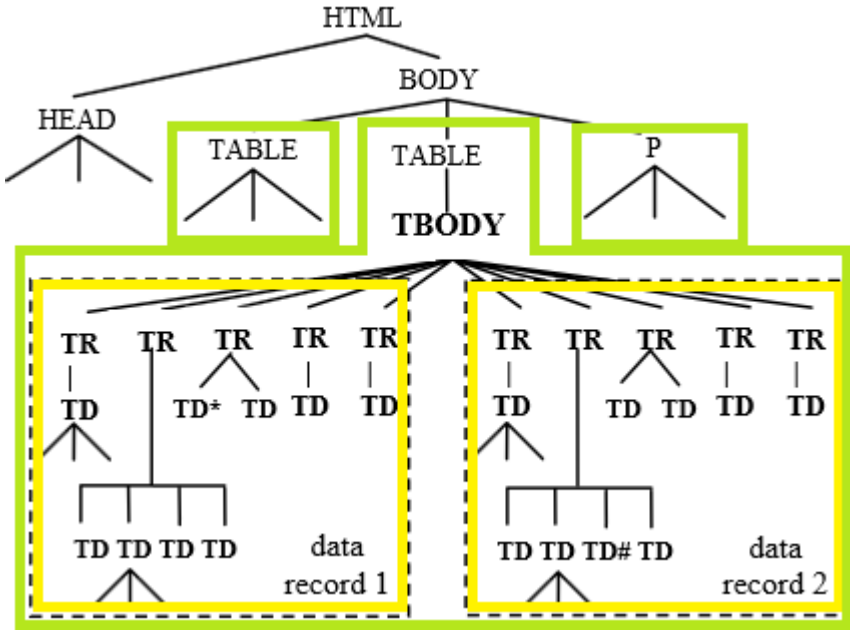
2.1.3.1 Mining Data Records (MDR)

MDR [7] je algoritmus, ktorý hľadá rovnako formátované informácie – dátové záznamy vo väčšej štruktúre dokumentu, ktorú nazývame dátová oblasť. Algoritmus pracuje s DOM daného dokumentu ako so stromovou štruktúrou a jej postupnými prechodmi až k listom určuje dátové oblasti. Nájsť dátovú oblasť resp. oblasť v ktorej sa nachádzajú samotné záznamy, je pomerne jednoduché na základe porovnávania reťazcov vytvoreného stromu. Ďalej sú analyzované konkrétne vyselektované dátové oblasti a v nich identifikované záznamy.

Obrázok 2.6 Nájdené dátové oblasti v stromovej štruktúre



Obrázok 2.7 Nájdené dátové záznamy v stromovej štruktúre HTML: zelené sú dátové oblasti – data regions, žlté sú dátové záznamy – data records



K samotnej identifikácií dátových oblastí a dátových záznamov používa MDR tzv. generalizované uzly. Generalizované uzly sú skupiny uzlov, ktorých počet je väčší ako 1 a platia pre ne dve pravidlá:
1) Všetky generalizované uzly danej skupiny majú rovnakého rodiča
2) Všetky generalizované uzly sú „príbuzné“

Dátová oblasť v závislosti na generalizovaných uzloch musí spĺňať nasledovné pravidlá:
1) je zložená zo skupín normalizovaných uzlov s rovnakým rodičom a dĺžkou
2) všetky skupiny normalizovaných uzlov sú príbuzné
3) normalizovaná editačná vzdialenosť14 neprekročí určitú manuálne stanovenú hranicu

Samotný algoritmus je pomerne zložitý a zdĺhavý, z toho dôvodu ho tu neuvádzam, v prípade záujmu ho je možné nájsť na [7] strany 3 a 4. Hlavnou **výhodou** tohto prístupu je možnosť extrakcie množstva záznamov z dokumentu bez znalosti štruktúry jednotlivých záznamov. Táto metóda je veľmi efektívna pokiaľ sa jedná o záznamy v tabuľkách formulároch a pod. Presnosť a časovú náročnosť nastavuje užívateľ nastavením normalizovanej editačnej vzdialenosti.

Nevýhody:
- Zložitosť implementácie
- Neefektívnosť v prípade množstva nájdených dátových oblastí
- Neefektívnosť v prípade malého množstva záznamov v dátových oblastiach

[14] Rozdielnosť textového obsahu, podľa urč. algoritmu napr. Levenshteinova vzdialenosť [24]

2.1.3.2 Tag Path Clustering (TPC)

Iným prístupom k extrakcií dátových záznamov je Tag Path Clustering [8]. Je to metóda založená na analýze absolútnych ciest k jednotlivým prvkom dokumentu. Pri TPC dochádza k „*preorder*“ prechodu DOM štruktúrou dokumentu a následnej analýze daných ciest. Pre dátové záznamy, ktoré patria tomu istému regiónu platí, že budú mať rovnakú postupnosť ciest k elementom.

Tabuľka 2.3 Príklad absolútnych ciest k dvom dátovým záznamom v jednom regióne:

/html/body/div/div/div/h4	**/html/body/div/div/div/h4**
/html/body/div/div/div/p	**/html/body/div/div/div/p**
/html/body/div/div/div/p/strong	**/html/body/div/div/div/p/strong**
/html/body/div/div/div/p	**/html/body/div/div/div/p**
/html/body/div/div/div/p/i	**/html/body/div/div/div/p/i**
/html/body/div/div/div/a	**/html/body/div/div/div/a**

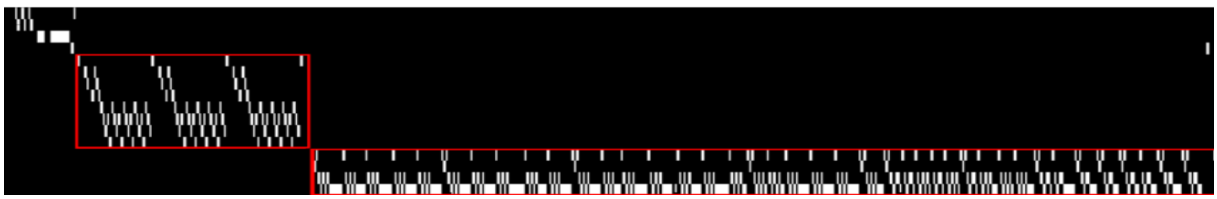
Tieto opakovania sa v TPC identifikujú prostredníctvom tvorby „clusterov“. TPC rieši problém tvorby „clusterov“, vytváraním vektorov z daných ciest, ktoré budú dokopy tvoriť vizuálny signál. Formálne vizuálny signál S_i je usporiadaná trojica $\langle p_i, S_i, O_i \rangle$, kde $p_i$je cesta k elementu, S_i je vektor vizuálneho signálu reprezentujúci výskyt pozícií p_i v danom dokumente a O_i je usporiadaný zoznam, ktorý reprezentuje jednotlivé výskyty. S_i je binárny vektor, kde $S_i(j) = 1$ ak sa p_i v DOM štruktúre dokumentu vyskytuje na pozícií (j), ak sa nevyskytuje tak $S_i(j) = 0$.

Tabuľka 2.4 Príklad určenia vektoru vizuálneho signálu z absolútnych ciest v dokumente:

Unique tag path	Pos	Visual signal vector
html	1	[1,0,0,0,0,0,0,0]
html/body	2	[0,1,0,0,0,0,0,0]
html/body/h1	3	[0,0,1,0,0,0,0,0]
html/body/table	4	[0,0,0,1,0,0,0,0]
html/body/table/tr	5,7	[0,0,0,0,1,0,1,0]
html/body/table/tr/td	6,8	[0,0,0,0,0,1,0,1]

Samotné vyhľadanie „clusterov“ vo vizuálnej reprezentácií signálu zabezpečuje algoritmus „*normalized cut spectral clustering algorithm(NCSCA)“* [9]. Tento algoritmus vytvára skupiny vizuálneho signálu rovnakého vzoru(štruktúry). Navyše sú do tohto algoritmu dodané pravidla ako napr., že sa dátový záznam nemôže skladať z menej ako troch elementov a mnohé ďalšie [8].

Obrázok 2.8 Príklad výstupu po NCSCA



Každý riadok je reprezentácia vektoru vizuálneho signálu. Svetlé pixely sú 1tky tmavé 0ly

Hlavnou **výhodou** tohto prístupu je pomerne vysoká presnosť identifikácie dátových záznamov a ich následná extrakcia je bezproblémová, keďže máme konkrétne absolútne cesty k daným elementom.

Nevýhody:
- Zložitosť implementácie
- Závislosť na výkone(vyžaduje vysoký výkon pre optimálny beh algoritmu)
- Neefektívnosť v prípade malého množstva záznamov v jednotlivých „clustroch".

2.1.4 Extrakcia na základe dĺžky textov v jednotlivých „*tagoch*"

Hlavným princípom takejto extrakcie je odstránenie všetkých „*tagov*", obrázkov, atď. Analyzovaný bude iba vnútorný obsah jednotlivých elementov resp. tzv. „*inner text*" jednotlivých elementov. Najznámejším zástupcom využívajúci tento princíp je metóda ***Text-To-Tag Ratio* (TTR)** [10], ale patria sem aj mnohé iné zvyčajne veľmi jednoduché ba až primitívne metódy extrakcie textov ako napr. extrakcia celého „*plain textu*" z dokumentu po odstránení obrázkov a skriptovania v dokumente a mnohé iné.

2.1.4.1 Text-ToTag Ratio(TTR)

Vznik TTR [10] umožnil rozvoj používania CSS, ktorý odlúčil vzhľad od obsahu a tak výrazne zredukoval počet „*tagov*" v HTML dokumentoch. Hlavným princípom metódy je výpočet hodnoty Text-To-Tag, ktorá nám určuje rozsiahlosť textov v určitých elementoch dokumentu. Tento algoritmus pracuje s jednotlivými riadkami vstupného dokumentu a identifikuje „*tagy*" v jednotlivých riadkoch(najčastejšie prostredníctvom reg. výrazov). Aby nedošlo k nepresným výsledkom musí byť vstupný dokument predspracovaný. Pri predspracovaní dokumentu sa v prvom rade odstraňuje skriptovanie, ďalej komentovaný text a napokon prázdne riadky a formátovanie textu.

Hneď pri prvom zamyslení sa nad daným princípom pozorujeme problém s krátkymi a pritom významnými textami, ktoré sú vo vlastných elementoch. Hodnota TTR, takýchto textov bude pomerne nízka. Z pravidla sa ale jedná o nadpisy, podnadpisy, dátumy, autora a pod. Na odstránenie tohto problému sú výsledky metódy TTR vložené do „clustrov", v ktorých sú k textom s vysokým TTR taktiež pridelené texty z okolitých riadkov, pokiaľ sa tam nejaké nachádzajú. Veľkosť okolia sa manuálne nastavuje na základe predpokladanej štruktúry extrahovaného textu. Štandardná veľkosť okolia sú dva riadky(nadpis a podnadpis vpredu dátum a autor vzadu). V prípade, že sa v okolí text nenachádza alebo je minimálny, t.j. len znak alebo krátka skupina znakov je ignorovaný. Vytváranie „clusterov" na základe okolia elementu s vysokým TTR nie je jedinou metódou. „Clustre" môžu byť vytvorené aj na základe rôznych metód ako napr. „*K-Mean*" algoritmus [11], „*EM*" algoritmus [12], „*Farthest-First*" algoritmus [13], a „*Longest Common Subsequence*" algoritmus [14].

Existuje aj zjednodušená verzia tohto algoritmu. V prípade, že nemáme rôznu štruktúru webových dokumentov, nemusíme pracovať jednotlivými riadkami dokumentu, ale s jednotlivými elementmi na základe DOM objektu. Vďaka tomu nie je nutné vykonávať klastrovanie(„*clustering*"). Touto zjednodušenou metódou sa ešte budem zaoberať v kapitole 4, pretože bude využitá pri implementácií riešenia problému extrakcie správ.

Pseudokód 2.2 výpočet hodnoty TTR

```
input
  h← HTML source code
begin
  Remove all script, remark tags and empty lines
  for each line k to numLines ( h ) do
    x ← number of non-tag ASCII characters in h [ k ]
    y ← number of tags in h [ k ]
    if  y = 0 then
        TTRArray [ i ] ← x
    else
        TTRArray [ i ] ← x / y
    end if
  end for
  return TTRArray
end
```

Najvýhodnejšie je použiť metódu TTR na weby, ktoré majú rozdielnu štruktúru, keďže je táto metóda nezávislá na podobnosti vstupných dokumentov. Táto výhoda je, ale aj nevýhodou, keďže TTR spracováva každý jeden dokument od začiatku až po koniec a nerieši podobnosti jednotlivých dokumentov.

Nevýhody:
- Spracovanie každého dokumentu jednotlivo od začiatku až po koniec
- Slabá detekcia malých textov
- V závislosti na definovanom okolí môže do výsledku zahŕňať nesúvisiace informácie(tzv. informačné presahy)
- Reklamy a nechcený komerčný obsah, v ktorom sa vyskytuje text budú zahrnuté do výstupu.

2.1.5 Extrakcia dát na základe vizuálnej reprezentácie HTML dokumentov

Je spôsob extrakcie využívajúci vizuálne členenie webovej stránky. Táto metóda imituje vnímanie a správanie človeka pri prezeraní dokumentu. Človek nevníma webovú stránku ako celok ale ako množinu usporiadaných blokov(ako puzzle alebo lego s tým rozdielom, že jednotlivými dielmi budú jednotlivé div – y, span – y , atď.). Jednotlivé bloky sa od seba odlišujú rozdielnymi až kontrastnými vlastnosťami, napríklad farba pozadia, typ fontu, prípadne sú hranice jednotlivých blokov oddelené tzv. oddeľovačmi najčastejšie horizontálne alebo vertikálne čiary, okraje elementov(„*border*“) a iné.

2.1.5.1 VIPS algoritmus

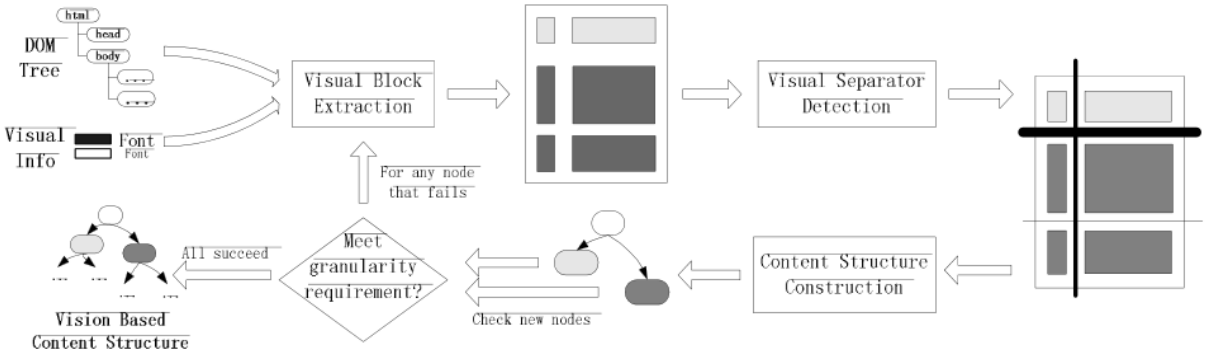
VIPS [15] je algoritmus detekujúci štruktúru HTML stránky na základe jej vizuálnej reprezentácie. Jeho výsledkom nemusí nutne byť štruktúra odpovedajúca štruktúre DOM daného dokumentu. Tento algoritmus definuje webovú stránku ako usporiadanú trojicu $\Omega = (O, \Phi, \delta)$, kde $O = \{\Omega_1, \dots, \Omega_n\}$ čo je konečná množina objektov alebo podstránok danej stránky, pričom platí, že každý objekt danej stránky musí byť zobraziteľný ako samostatná podstránka. $\Phi = \{\Phi_1, \dots, \Phi_n\}$ je konečná množina oddeľovačov vrátane horizontálnych a vertikálnych oddeľovačov. Každý oddeľovač má svoju dôležitosť(váhu), ktorá indikuje jeho dôležitosť

a všetky oddeľovače patriace do Φ ju majú rovnakú, t.j. sú rovnocenné.δ reprezentuje vzťah medzi dvoma prvkami patriacimi do O a môže byť vyjadrená ako $\delta = O\ x\ O \rightarrow \Phi \cup \{NULL\}$. Pre lepšie pochopenie: predstavme si, že $\Omega_1, \Omega_2 \in O$, dva objekty množiny O. Potom $\delta(\Omega_1, \Omega_2) \neq NULL$ znamená, že Ω_1 a Ω_2 sú dva objekty vzájomne oddelené oddeľovačom resp. sa jedná o susedné objekty.
$\delta(\Omega_1, \Omega_2) = NULL$ znamená, že Ω_1 a Ω_2 sa nachádza minimálne jeden objekt.
Pre každý vizuálne oddelený blok je definovaný „*stupeň spojitosti*“ tzv. DoC[15], ten je definovaný na meranie spojitosti jednotlivých blokov. Má nasledujúce vlastnosti:

- Rozsah $\langle 0, 1 \rangle$
- S rastúcim DoC rastie konzistencia bloku
- V hierarchií potomok urč. bloku má DoC vždy väčšie ako jeho predok

Môžeme preddefinovať „Povolený“ DoC (PDoC[16]) na nastavenie „zrnitosti“ výstupnej štruktúry pre rôzne aplikácie. Čím menší PDoC bude, tým bude výsledná štruktúra menej členitá.

Obrázok 2.9 Grafický popis VIPS



Na počiatku VIPS algoritmu sú detekované jednotlivé bloky vstupného dokumentu. Tie sú určené z DOM štruktúry dokumentu a informácií o vizuálnej reprezentácií dokumentu získané z prehliadača. Pravidiel na rozlíšenie blokov je veľmi mnoho, sú definované na základe ľudského vnímania. Patria sem napr. odlišná farba pozadia, rozdielne fonty a štýlovanie alebo aj druh elementu t.j. číslované zoznamy, tabuľa a mnohé iné. Na detekciu oddeľovačov používa VIPS algoritmus „*Visual Separator Detection*“.

Tento algoritmus by sa dal zjednodušene zapísať takto:

1) Inicializáciu zoznamu oddeľovačov. Na začiatku zoznam obsahuje len dva oddeľovače a to počiatočný a koncový pixel daného objektu
2) Pre každý blok je ohodnotený jeho vzťah ku každému oddeľovaču a :
   - Ak je blok obsiahnutý v urč. oddeľovači, rozdeľ oddeľovač
   - Ak sa blok kríži s nejakým oddeľovačom modifikuj parametre oddeľovača
   - Ak blok prekrýva oddeľovač, odstráň oddeľovač
3) Odstráň 4 oddeľovače, ktoré odpovedajú hraniciam daného bloku

Výstupom VIPS algoritmu bude vizuálne členený dokument, v ktorom budeme môcť pristupovať k jednotlivým zložkám a ďalej s nimi pracovať, napr. z nich stiahnuť určité dáta. Najväčšou **výhodou** takéhoto prístupu je pre užívateľa jednoduchosť vstupu, kde užívateľ len

[15] Skratka z anglického názvu Degree of Coherence
[16] Skratka z ang. slovného spojenia Permitted Degree of Coherence – preddefinovaný stupeň členitosti

manuálne označí vizuálne členenú časť dokumentu, ktorej spracovanie chce dosiahnuť. Ďalšou nespornou výhodou je eliminácia informačných presahov pri extrakcií.

Nevýhody:
- Komplikovaná implementácia
- Závislosť na informáciách o grafickej podobe dokumentu(spolupráca s prehliadačom)
- Výkonnostná náročnosť
- Nemožnosť extrakcie dát z vizuálne neodčlenených elementov

2.2 Extrakcia dát z webu pomocou dostupných aplikácií – „wrapperov“

2.2.1 Čo je „wrapper“

„Wrapper[17]“ alebo aj „extractor[18]“ je špeciálny typ aplikácie slúžiaci na zisk informácií z konkrétneho informačného zdroja. Extrakcia informácií je vykonávaná na základe určitých definovaných extrakčných pravidiel v danej aplikácií.
Vstupom takéhoto programu je dokument alebo množina dokumentov, prípadne odkaz na dokument. Výstupom bude vyselektovaná množina dát na základe určitých extrakčných pravidiel. Tie sa definujú pri spustení programu, sú v ňom naprogramované, alebo sa ich snaží aplikácia vytvoriť automaticky.

Na základe toho delíme wrappery na:
- **Manuálne**: jedná sa o pevne napísané wrappery slúžiace k extrakcii jediného cieľového dokumentu, zmeny extrakčných pravidiel sú možné len na programovej úrovni
- **Semi – automatické**: wrappery, ktoré umožňujú urč. spôsobom modifikovať extrakčné pravidlá wrapperu na základe vstupu od užívateľa
- **Automatické**: wrappery vytvárajúce extrakčné pravidla automaticky bez zásahu užívateľa, takéto wrappery je pre extrakciu štruktúrovaných dát z webu takmer nemožné vytvoriť, z dôvodov popísaných v predchádzajúcich častiach dokumentu. Preto sa do takýchto wrapperov často implementuje určitá semi – automatika, ktorá umožní automatické vytvorenie extrakčných pravidiel na základe určitého vstupu, alebo takéto wrappery budú fungovať len pre určitú informačnú doménu(oblasť) dát.

Ďalej delíme wrappery na základe spracovania vstupného dokumentu na :
- **Reťazové wrappery**: spracovávajú vstupný dokument lineárne(SAX parsing, regex, ...)
- **Stromové wrappery**: pracujú s DOM štruktúrou dokumentu(k elementom pristupujú na základe XPath)

Ďalším možným delením je delenie na základe cieľu extrakcie:
- **Záznamové wrappery**: spracovávajú len vstupný dokument bez hĺbkovej analýzy
- **Súborové wrappery**: spracovávajú celý súbor vstupných dokumentov, ten je buď zadaný, alebo vykonávajú prechod do hĺbky na základe hypertextu.

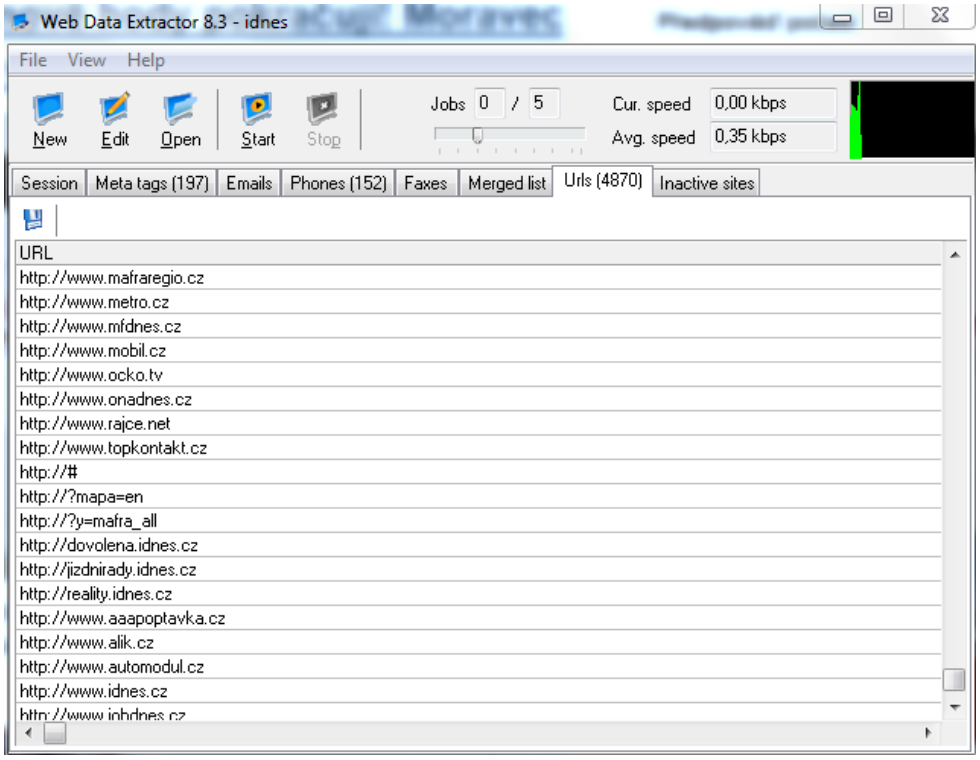
[17] Odvodené od ang. slova wrapper – obálka, to vystihuje činnosť daného programu, ten hľadá obaly/ obálky určitých časti vstupného dokumentu
[18] Odvodené od ang. slova extract – sťahovať, získavať, dolovať

2.2.2 Prehľad niektorých známych voľne dostupných wrapperov

2.2.2.1 Web Data Extractor v 8.3 [19]

Web Data Extractor je web wrapper umožňujúci extrakciu jednoduchých neštruktúrovaných dát z webu. Jedná sa o súborový wrapper prehľadávajúci webové portály do hĺbky, pričom hĺbka extrakcie môže byť nastavená. Množina extrakčných pravidiel pozostáva zo zložených regulárnych výrazov, najčastejšie sa jedná o štandardné regulárne výrazy pre email, dátum, telefónne číslo, URL. Tento wrapper je distribuovaný v platenej „*Pro*“ edícií, ktorá umožňuje aj tvorbu vlastných regulárnych výrazov na základe špecifikovaných slov. Regulárny výraz je potom optimalizovaný algoritmom ReLIE. „*Free*“ verzia programu túto možnosť nezahŕňa.
Web Data Extractor je semi – automatický wrapper poskytujúci možnosť vykonania množstva nastavení a obmedzení extrakcie (napr. obmedzenie niektorých URL, odstránenie obsahu so stránok s urč. špecifickým obsahom a pod.). Nastavenia u tohto wrappera sú až tak rozsiahle a nutné(t.j. bez špecifikovania rozsiahlych vstupných nastavení nejde spustiť), že bez manuálu je takmer nepoužiteľný.

Obrázok 2.10 Ukážka z Web Data Extractoru 8.3 „Free edition“:



2.2.2.2 Web Harvest [20]

Je semi – automatický súborový wrapper napísaný v jazyku JAVA. Distribuovaný je ako spustiteľný .jar súbor. Extrakčné pravidlá sú manuálne definované a vyžadujú značnú mieru znalosti štruktúry stránky, XML, XSLT, jazyka JAVA, XPath a napokon samotnej znalosti programu Web-Harvest. Z toho je jasné, že tento program nie je pre každého, ale len pre skúsených užívateľov so skúsenosťami s programovaním.

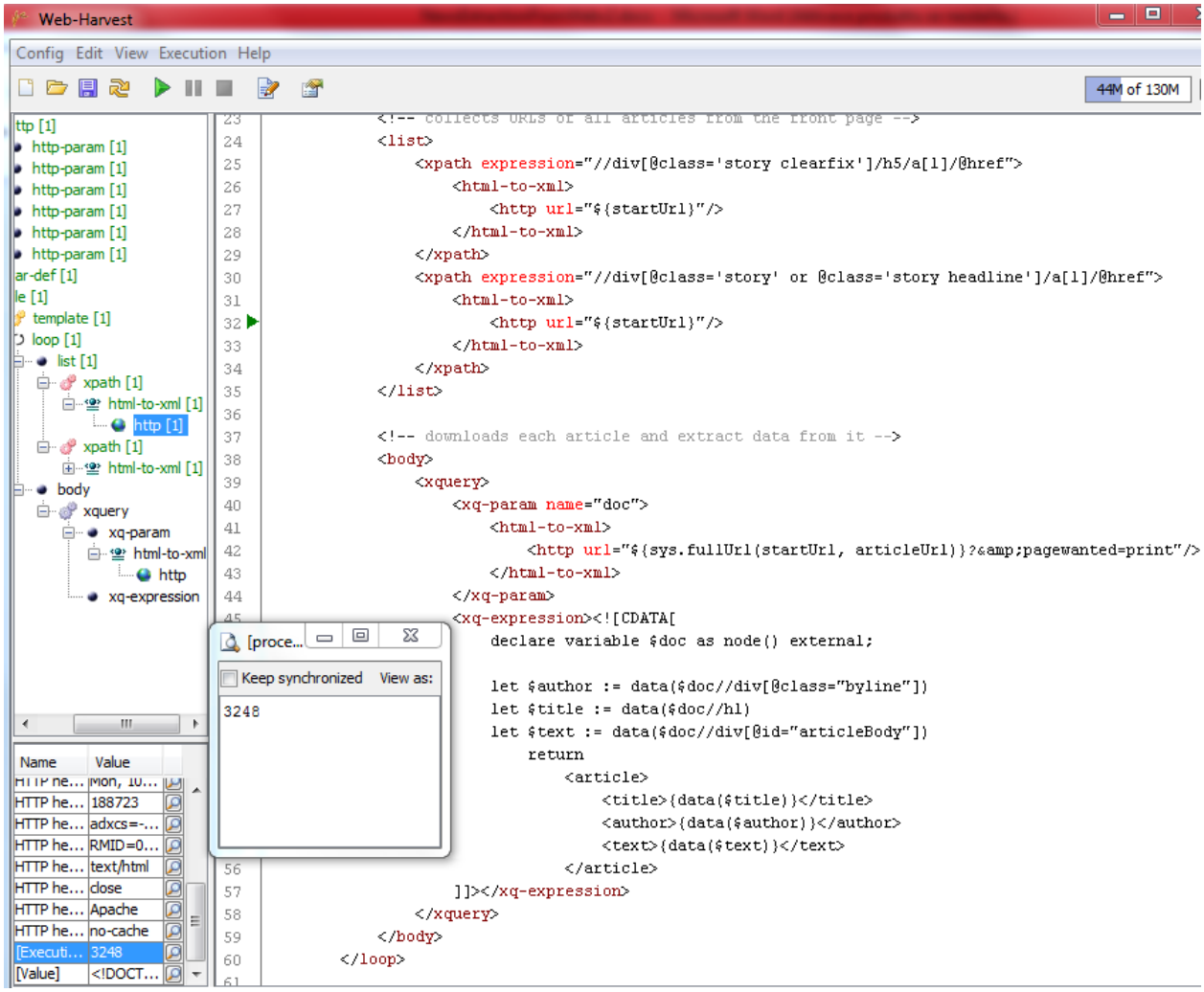
[19] http://www.webextractor.com/wde.htm
[20] http://sourceforge.net/projects/web-harvest/

Samotná extrakcia prebieha až po vytvorení konfiguračného súboru. Aj najmenšia chybička v konfigurácií zmarí celú extrakciu. Program je preto dodávaný s pred pripravenými príkladmi, ktorých modifikáciou môže užívateľ dosiahnuť požadovaného výsledku extrakcie.
Tento wrapper určite patrí medzi tie najdokonalejšie voľne dostupné wrappery, avšak zložitosť vstupu tohto wrapperu je zrovnateľná so zložitosťou vlastného „*hard – coded* “ wrapperu t.j. manuálneho wrapperu slúžiaceho na extrakciu určitého dokumentu alebo skupiny dokumentov s uniformnou pevne danou štruktúrou.

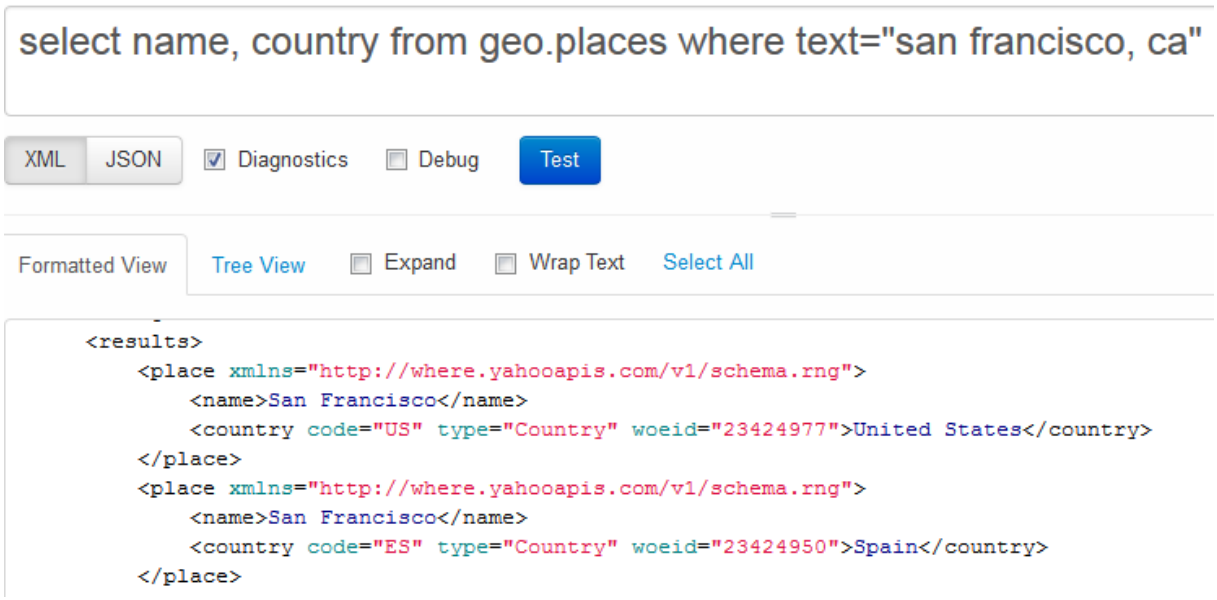
2.2.2.1 YQL (Yahoo! Query Language)

YQL [1] je dotazovací jazyk podobný SQL, ktorý umožňuje filtrovať, štruktúrovať, triediť a dolovať textové dáta z predspracovaných webových stránok. Tento nástroj je dostupný „*online*“ na http://developer.yahoo.com/yql/console/, kde si ho môže každý vyskúšať. YQL vyžaduje databázové spracovanie textu. Jednotlivé texty sú reprezentované ako tabuľky a dotazovanie pripomína SQL.

Obrázok 2.11 Ukážka z Web – Harvest 2.0 :



Obrázok 2.12 Príklad použitia YQL v ponúkanej konzole na hore uvedenej stránke



2.2.2.2 Niektoré iné wrappery

Existuje množstvo iných wrapperov, či už platených alebo voľne dostupných, keďže ich funkcionalita je veľmi podobná tým už spomenutým a pre tému tejto DP nepodstatná tak len uvediem pár ďalších bez popisu a ďalej sa nimi už nebudem zaoberať: WebScraper+[ii], HTMLasText[iii], nástroj ONTEA spomínaný v kapitole 2.1.1.2, HappyHarvest 2.5.6, t.j. nástroj založený na Web Harvest 2.0 spomínaný vyššie, WebScraper - Web Data Extraction[iv] a mnohé iné. Existuje skutočne nesmierne množstvo takýchto nástrojov od najprimitívnejších po celkom sofistikované ale platené nástroje.

2.3 Text Mining a wrappery zhrnutie

TM bolo témou mnohých výskumov a stále je oblasťou skúmania kvôli nesmiernej zložitosti a komplexnosti tohto problému. V tejto kapitole som sa zapodieval niektorými prístupmi k problému TM. Snažil som sa vybrať z množstva techník jednotlivých zástupcov a podrobnejšie ich preskúmať. Okrem uvedených metód však existuje veľmi mnoho iných prístupov a mnou zvolené prístupy nemusia byť tie najefektívnejšie ani najznámejšie. Taktiež som v tejto kapitole spomenul pár existujúcich aplikácií slúžiacich na extrakciu dát z webu, tzv. wrapperov. V súčasnej dobe je mnoho wrapperov a nové sa stále vyvíjajú. Všetky trpia však rovnakými problémami:

- ak sú dostatočne výkonné a schopné sťahovať štruktúrované dáta, tak je veľmi ťažké definovať vstup, obzvlášť užívateľovi bez schopnosti programovať
- ak je jednoduché definovať vstup, program doluje len primitívne dáta bez štruktúry(URL, emaily, ...)

Pre problém „extrakcie správ z webu“, čo je hlavným predmetom záujmu tejto DP, nie je žiaden z nich použiteľný, v malej miere by sa tento problém dal riešiť prostredníctvom Web – Harvesteru 2.0 , ktorému však je veľmi ťažké definovať vstup, ba dokonca sa mi to nepodarilo, tak aby som bol s výsledkom plne spokojný. Taktiež pre každý jeden web by bolo nutné špecifikovať novú konfiguráciou daného wrappera. Aplikáciou poznatkov získaných pri tvorbe tejto kapitoly na problém extrakcie správ z webu sa budem zaoberať v nasledujúcej kapitole.

3. Analýza problematiky extrakcie správ z webu[v]

3.1 Čo je správa a ako sa líši od informácie

Pod pojmom správa v tejto DP rozumieme **spravodajskú textovú správu**, čo je špecifický druh informácie. Spravodajské servery poskytujú nesmierne množstvo informácií, len niektoré z nich sú však správami. Najvšeobecnejšia a najpoužívanejšia definícia informácie je:
Informácia je zdelenie, oznámenie, ktoré má pre užívateľa určitý význam a mení jeho vnútorný stav z nevedomosti na uvedomenosť a odstraňuje nevedomosť v danej oblasti[21].
Na základe tejto definície sú informácie aj oznámenia typu:
Firma XY, zaoberajúca sa zberom informácií, ponúka najlacnejšie služby v celej Európe.
Autor našich článkov František je vášnivý fotograf, zbierku jeho fotografií si môžete prezrieť na http://fotograffrantisek.com.
Je jasné, že hore uvedené oznamy sú síce informáciami nachádzajúcimi sa na spravodajskom webovom portály, ale nejedná sa o správy. Aby sme dokázali správy správne identifikovať musíme si ich najprv definovať, na základe empirického skúmania a pozorovania webových spravodajských portálov ju definujem takto:
Správa je špecifický typ informácie obsahujúca nasledujúce atribúty:

- titulok
- popis
- zdroj
- dátum
- informačné zdelenie – telo správy

Aby bola správa správou nie len informáciou sú všetky tieto atribúty nevyhnutné. Optimálne by sem mal patriť aj autor, bohužiaľ málo kedy je uvedený, najčastejšie sa uvádzajú len autori fotografií k článku, ktorí sú pre nás nepodstatní. Keď aj je autor uvedený, je uvádzaný len ako odkaz na jeho osobnú webovú prezentáciu v rámci portálu. Na základe pozorovania až štyri z piatich správ nemali autora uvedeného a ak aj mali , tak sa v elemente prislúchajúcemu autorovi nachádzal fotograf s popisom, že sa jedná o autora fotografií vo vnútornom texte daného elementu. Na základe týchto vypozorovaných faktov autorov k článkom nebudem uvádzať keďže by boli často buď neznámi, alebo mylne identifikovaní.

3.2 Problém identifikácie tela správy

Ako som už naznačil v predchádzajúcej kapitole, problémom je rozlíšiť správu a informácie obsiahnuté na spravodajských portáloch. Vykonať toto rozlíšenie algoritmicky medzi dvoma textami je nemožné kvôli obsiahlosti a komplexnosti prirodzeného jazyka. Našťastie si môžeme pomôcť **ontológiou** – štruktúrou webových portálov a **sémantikou**(pridaným významom, najčastejším vo forme metadát).

Na identifikáciu správy potrebujeme identifikovať jej jednotlivé atribúty.
Za atribút **zdroj**(*source*) považujeme URL na ktorom sa správa nachádza, z toho dôvodu je identifikácia tohto atribútu bezproblémová. Atribúty **titulok**(*title*) a **popis**(*description*) sú taktiež bezproblémové, keďže spravodajské servery dodržujú trendy a princípy definované W3C[22]. Atribút „*title*“ resp. titulok obsahuje každá webová stránka a jeho extrakcia je bezproblémová. Atribút „*description*“ resp. popis je jedným zo štandardne uvádzaných metadát(spoločne s kľúčovými slovami, verziou HTML, jazykom atď.). Extrakcia metadát sa pro-

[21] Definícia sa opiera o http://www.spsdopke.edu.sk/studmaterial/zakladnepojmyinformatiky.pdf
[22] http://www.w3.org/TR/html401/struct/global.html

stredníctvom štandardných regulárnych výrazov realizuje pomerne ťažko, ale je bezproblémová prostredníctvom DOM objektu daného dokumentu. Existuje niekoľko pomerne dokonalých nástrojov ktoré umožňujú pokročilé spracovanie HTML dokumentov na základe ich DOM objektu. Jedným z nich je napríklad HTML AGILITY PACK[23] pre C# v aktuálnej verzií 1.4.6. Tento pokročilý nástroj som sa rozhodol pri vypracovaní tejto DP používať, keďže podľa zdrojov na MSDN[24] sú jeho mechanizmy a algoritmy efektívnejšie ako je tomu u triedy „*Regex*“, t.j. triedy regulárnych výrazov v jazyku C#.

Atribút **informačné oznámenie**, resp. samotné telo správy je pomerne problémové. Telo správy pozostáva z textu v prirodzenom jazyku, ktorý je formátovaný rôznymi HTML elementmi(nadpisy *h1 – h6, p, span, table*, ...). Identifikácia automatickými algoritmami bola popísaná v kapitole 2. Takáto identifikácia by bola veľmi výkonnostne náročná a žiadna z uvedených metód nie je úplne vhodná. To zapríčiňuje štruktúra webových portálov (viď nasledujúcu kapitola 3.3). Našťastie si môžeme pomôcť štruktúrou daných HTML dokumentov. Keďže správa, aj keď je rozdelená do viacerých HTML elementov má pôsobiť celistvým dojmom využíva jednotné formátovanie definované v CSS dokumente príslušného k danej webovej stránke. Aby bolo možné efektívne priradenie tohto jednotného formátovania, je vytvorený nadradený element s vlastnosťou *class= “dané formátovanie“* alebo *id*= “*dané formátovanie*“. Tento fakt môžeme pri extrakcií veľmi dobre využiť a na základe danej informácie extrahovať celé telo správy ako vnútorný text v prirodzenom jazyku, z ktorého sú odstránené všetky pre prehliadač určené informácie(skriptovanie, obrázky, elementy). Touto štruktúrou sa vyznačujú úplne všetky pozorované weby. Napríklad server http://idnes.cz zaobaľuje telá svojich správ elementom div s atribútom *class=“bbtext“*, server http://novinky.cz divom s hodnotou atribútu *class=“articleBody“*, server http://aktuálne.centrum.cz využíva triedu ktorú identifikuje ako *„clanek“* a takto by som mohol pokračovať ďalej pre každý spravodajský server. Identifikácia daných elementov aj na základe hodnoty atribútu „*class*“ je nutná, keďže niektoré servery týmto elementom nepriradzujú atribút „*id*“ a iné servery dokonca nevyužívajú element „*div*“, ale namiesto neho používajú iný typ elementu, napr. „*span*“.

Atribút **dátum**(„*date*“) je pomerne problémový hoci na základe W3C a princípov tvorby sémantického webu by mal byť k dátumu priradený element obsahujúci hodnotu dátumu v atribútoch v štandardnom formáte[25] alebo aspoň daný dátum by mal byť uvedený v takomto formáte. Bohužiaľ realita je iná a dátumy sú často uvádzané v neidentifikovateľnom formáte typu: *včera, dnes, predvčerom, 5teho februára* a mnohé a mnohé iné. Z môjho pozorovania vyplýva, že len 2 z 5 webových portálov majú správne uvedené dátumy a tým pádom je umožnené ich jednoduché strojové spracovanie. Nanešťastie štruktúra týchto dátumov neumožňuje vytvorenie jednoduchého regulárneho výrazu, ktorý by identifikoval všetky takto uvedené dátumy. Taktiež informačná hodnota dátumu typu včera bez explicitnej informácie dátumu dňa extrakcie je úplne zbytočná. Keďže považujem dátum za nutný atribút správy, pretože inak by správami mohli byť aj reklamné oznámenia a mnohé iné informácie, je potrebné zostaviť zložené regulárne výrazy a imple-mentovať neefektívne vyhľadávanie dátumu, ktoré celú extrakciu značne spomaľuje. Okrem regulárnych výrazov je možné dátumy znova hľadať prostredníctvom tried CSS daného dokumentu. Z pozorovania vyplýva, že až osem z desať webových portálov má dátum umiestnený v špeciálnom elemente, ktoré trieda obsahuje slová ako „*DATE*“ alebo „*TIME*“. Po aplikácií tohto poznatku sa úspešnosti extrakcie dátumov značne zvýši. Pri extrakcií dátumov si tiež treba dať značný pozor, keďže takmer všetky spravodajské servery uvádzajú aktuálny dátum spoločne s informáciou o tom

[23] http://htmlagilitypack.codeplex.com/
[24] http://msdn.microsoft.com/en-US
[25] http://www.w3.org/TR/NOTE-datetime

kto oslavuje v ten deň meniny. Preto je nutné obmedziť hľadanie dátumu správy na jej telo alebo jeho bezprostredné okolie. Štandardne by mali stačiť dva až tri elementy.

3.3 Štruktúra spravodajských serverov, prehľadávanie do hĺbky a navigácia v rámci daného serveru

Pri extrakcií webových správ z určitého servera je nutné navrhnúť algoritmus, ktorý zabezpečí prechod všetkými časťami serveru. K tomu je potrebné vykonať dôkladnú analýzu štruktúry jednotlivých častí daných serverov, hypertextových prepojení medzi nimi a celkovej navigácie, t.j. štruktúry URL odkazov, sledovanie podobností a iné. Z tejto analýzy vyplýva, že HTML dokumenty spravodajského serveru môžeme na základe štruktúry rozdeliť na 5 kategórií.

Prvá kategória je samotný **webový portál**, ktorý slúži len ako **rozcestník**. Neobsahuje žiadne správy, len odkazy na tzv. top správy, čo sú najčastejšie správy, ktoré majú najviac zobrazení alebo sú generované náhodne. Ďalej obsahuje odkazy na hlavné sub. weby, čo sú obdoby hlavného webového portálu, len sú tematicky zamerané na určitú kategóriu správ.

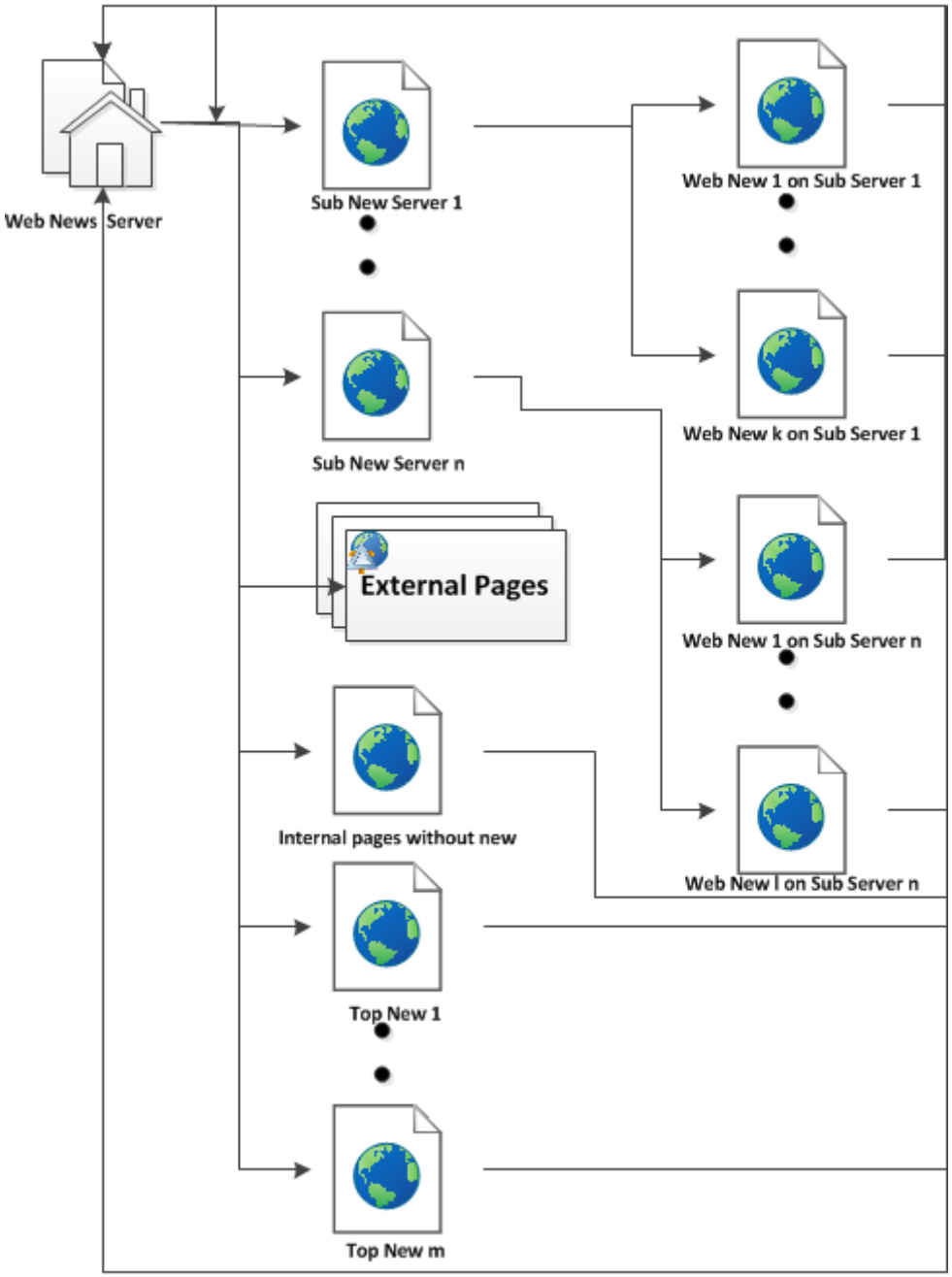
Druhá kategória sú **sub. weby** (pod portály)**.** Na týchto portáloch sa taktiež nevyskytujú žiadne správy, slúžia len ako **návestidlá**, t.j. len odkazujú na jednotlivé správy danej tematickej kategórie. Tiež obsahujú odkazy na tzv. top správy, aby užívateľa nalákali aj na prehľadávanie iných podkategórií. V prípade, že kategória, ktorú daný sub. portál spracováva je príliš obsiahla, **ďalej sa delí na ďalšie dielčie podportály** a tie sa môžu znova ďalej deliť. Majú podobnú štruktúru.

Tretia kategória sú **webové stránky obsahujúce správu**. Tie sú naším hlavným predmetom záujmu. Tieto odkazujú na top správy z rovnakých dôvodov ako členovia predchádzajúcich dvoch kategórií, taktiež kvôli navigácií odkazujú na jednotlivé podportály a hlavný portál. Ak je štruktúra podkategórií príliš členitá, tak odkazuje len na podportály smerujúce k danej správe a podportály s podobným tematickým zameraním.

Štvrtá kategória sú tzv. **interné odkazy**. Sú to dokumenty, obsahujúce videá, informácie, propagačné zdelenia, dokumenty potrebné pre správny chod portálu a pod. Štruktúrou sú veľmi podobné ako predchádzajúca kategória, ale neobsahujú správy, teda sú pre nás nepodstatné. Na tieto dokumenty odkazujú všetky kategórie, pri prechode do hĺbky sa ich prehľadávaniu nevyhneme, dôležite je kontrolovať všetky potrebné atribúty správy, aby ich obsah nebol zahrnutý do výsledku.

Piata kategória sú **externé odkazy**. Sú to odkazy na externé weby. Prehľadávanie týchto webov by mohlo spôsobiť nekonečný beh programu. V prípade, že ich je nutné prehľadávať je potrebné nastaviť určité TTL (time to live t.j. počet dokumentov, ktoré majú byť na externom webe prehľadávané najlepšie iba 1, t.j. len daný dokument). Všetky kategórie majú určitý počet externých odkazov, keďže ale extrakcia a identifikácia správ je vykonávaná na základe štruktúry budem externé odkazy ignorovať(ich dokumenty majú inú štruktúru a nič by sa z nich neextrahovalo).

Obrázok 3.1 Zjednodušená štruktúra spravodajských serverov a prepojení na základe pozorovania niektorých z nich[vi]. Jednotlivé sub. portály sa môžu ďalej členiť (pre prehľadnosť nezachytené):



Z tejto analýzy štruktúry webových portálov vyplýva, že v rámci jedného portálu sa nachádzajú rôzne typy HTML dokumentov a je nutné ich správne rozlíšiť. Z toho vyplýva nemožnosť priameho využitia metód opísaných v kapitole 2. Automatická detekcia správ a ich následná extrakcia je možná a bezproblémová v prípade, že by vstupom boli HTML dokumenty obsahujúce správy. Ak by boli vstupom dokumenty reprezentujúce portál alebo podportály, automatická detekcia by našla časti s najdlhším textom a mylne určila miesto výskytu tela správ pre hľadanie, čiže by podportály vystupovali ako dokumenty so správami a z dokumentov so správami by sa neextrahovalo nič, prípadne obsah elementu, ktorý so správou nijak nesúvisí. Z týchto informácií sa nám kryštalizuje nutný vstup aplikácie. **Vstupom** bude buď odkaz na portál alebo podportál s pripojenou informáciou o elemente obsahujúcom správu alebo bude vstupom odkaz na dokument/y obsahujúce správu, z ktorých sa informácia o elemente automaticky určí.

3.3.1 Analýza tvaru URL a rozlišovanie niektorých kategórií

Na základe tvaru a štruktúry URL môžeme odhadovať do akej kategórie, uvedenej vyššie, daný dokument patrí. Odkazy na externé weby sme schopní veľmi ľahko rozlíšiť, pretože ich „*Host*“ je odlišný od „*Hosta*“ vstupného URL. Tieto odkazy budem jednoducho ignorovať. Ďalšie rozlišovanie môžeme vykonávať na základe „*query stringu*“ alebo číselných hodnôt v URL dokumentu. Na základe pozorovania som zistil, že dokumenty obsahujúce správu majú vždy určitý „*query string*“ alebo aspoň určité číselné hodnoty v URL. Bohužiaľ nie všetky dokumenty, ktorých URL obsahovalo „*query string*“ obsahovalo správy, mnohé takéto URL odkazujú na dokumenty, ktoré som vyššie kategorizoval ako interné odkazy. Z toho dôvodu by rozpoznávanie len na základe tejto informácie bolo nedostačujúce. Preto je nutné implementovať kontrolu pre každý dokument spravodajského serveru, pokiaľ obsahuje všetky povinné atribúty správy, potom bude správa extrahovaná. Naopak URL portálu a podportálov „*query string*“ z pravidla neobsahujú, na základe skúmania štruktúry dokumentov v tejto kategórií (portál a podportály), vieme, že v týchto dokumentoch sa správy nevyskytujú, ale len odkazujú na množinu správ daného portálu alebo podportálu. Toto pravidlo sa potvrdilo na všetkých pozorovaných spravodajských portáloch, z toho dôvodu nemusím analyzovať takéto URL na výskyt správ, ale len získať adresy s možným výskytom správ a tak mierne zrýchliť prehľadávanie.

3.3.2 Návrh algoritmu prechodu spravodajským serverom do hĺbky

Na základe informácií získaných pozorovaním a popísaných v predchádzajúcej časti som schopný vytvoriť rekurzívny algoritmus zabezpečujúci prechod celým webovým spravodajským portálom.

Zjednodušene by tento algoritmus vypadal asi takto:

1. Vstupom bude URL portálu alebo podportálu s pridanou informáciou o triede elementu v ktorom sa vyskytuje správa alebo URL obsahujúce správu s informáciou, že sa trieda elementu so správou má určiť automaticky
2. Algoritmus získa množinu URL adries zo vstupného dokumentu
3. Získané URL budú filtrované a triedené na:
   - URL podportálov a portálu (tie bez „query stringu a číselných hodnôt“)
   - Externé URL, tie s rozdielnym „*Hostom*“, budú ignorované
   - Potencionálne odkazy na dokumenty so správami(tie s „*query stringom*“ alebo číselnými hodnotami a rovnakým „*Hostom*“)
4. URL s možným obsahom správy budú analyzované na výskyt atribútov správy, ak sa budú vyskytovať, správa sa extrahuje. Prehľadané URL budú uložené do pamäti Ω, aby sa zamedzilo viacnásobnej analýze a duplicitám vo výstupe.
5. Program bude postupne rekurzívne popísaným postupom analyzovať všetky zistené URL bez „*query stringu*“ (portál a podportály), táto množina O sa bude zväčšovať každým jedným volaním až pokiaľ v nej nebudú obsiahnuté všetky podportály.
6. Rekurzia sa ukončí vo chvíli, keď všetky pod portály O budú obsiahnuté v pamäti Ω ako prehľadné dokumenty. $O \subset \Omega$ a z posledného pod portálu budú analyzované všetky URL s potencionálnym výskytom správ.

Pseudokód 3.1 Crawlovanie v rámci spravodajského serveru zachytávajúci postup opísaný vyššie

```
List<string> searchedURLs ← new List<string>() // List of already checked URL s
List<string> subWebURLs ← new List<string>() // List of main sub web URL s to search
counter ← -1

Procedure crawl (url, identifier)
//url: URL of starting web page
//identifier: value of class or id to identifi element containing body of web new
for each string u to getURLs (url)
  if u.Host.Equals(url.Host) ∧ ¬searchedURLs.contains(u) then
    if u.isMatch(@"\d+") ∨ u.contains(queryString) then
      searchedURLs.add(u)
      getWebNew(identifier)
    else
      subWebURLs.add(u)
    end if
  end if
end for
increment counter
if counter < subWebURLs.count then
  crawl(subWebURLs[counter], identifier)
end if

Function getURLs(url)
return list of href values in all <a> elements

Function getWebNew(identifier)
//returns webNew object if all web new member variables are found, otherwise return NULL
```

3.4 Pozorovanie stávajúcich spravodajských portálov zhrnutie

Na základe faktov popísaných v predchádzajúcich troch kapitolách je jasné, že je možné zostaviť vlastný semi – automatický web wrapper schopný dôkladnej extrakcie správ z určitého portálu. Táto časť obsahuje sumár vypozorovaných informácií, čo slúži ako analýza pre implementáciu vlastného riešenia. Samotnou implementáciou, princípmi a použitými algoritmami sa budem zaoberať v nasledujúcej kapitole.

4. Semi – automatická extrakcia zo spravodajských portálov

V predchádzajúcej kapitole som popísal vzhľad správy umiestnenej na webovom portály a rekurzívny princíp prehľadávania*(„crawlovania")* spravodajským portálom, ktorý bol založený na štruktúre URL jednotlivých webových stránok(dokumentov) na danom portály. Na základe daných informácií sme schopný zostaviť tzv. semi – automatický wrapper správ z určitého spravodajského portálu. Najproblematickejším aspektom u tohto wrappera bude rozlíšiť u dokumentov, ktoré považujeme za potencionálnych nosičov správy, pripomínam, že sú to dokumenty, ktorých URL obsahuje „query string" alebo číselné hodnoty reprezentujúce určitý identifikátor správy, či sa na nich správa nachádza, alebo nie. Taktiež som v predchádzajúcej kapitole zmienil, že samotné tela správ sú v rámci webových portálov umiestnené do špeciálneho bloku(elementu), ktorý sa u dokumentov, kde sa správy nenachádzajú nevyskytuje. Tento blok sa vyskytuje len v dokumentoch obsahujúcich správy a je identifikovateľný, buď štandardne atribútom identifikátoru *„id"* v HTML elemente alebo priradením špeciálnej triedy z CSS dokumentu, ktorá najčastejšie slúži k nastaveniu jednotného formátovania fontov použitých v správe. Výskyt tohto bloku použijeme ako rozhodujúcu podmienku algoritmu. Ak algoritmus nájde daný blok v dokumente, ktorý zároveň spĺňa ďalšie špecifikované podmienky v predchádzajúcej kapitole(titulok, popis, dátum, zdroj a URL obsahujúce „query string" alebo numerické hodnoty) pristúpime k samotnej extrakcií správy.

Extrakcia titulku a popisu z metadát dokumentu je bezproblémová. K extrakcií samotného tela správy sa využije informácia o identifikátore bloku obaľujúcom telo správy, ktorý som už spomínal, alebo hodnota jeho CSS triedy. Z tohto vyplýva, že táto informácia musí byť špecifikovaná ako vstup na začiatku algoritmu. Bežný užívateľ, bez informatického vzdelania vo väčšine ani nemá poňatia o tom, že HTML dokumenty sú textové dokumenty napísané v jazyku XHTML a môže si prehliadať ich zdrojový kód, v ktorom by si ľahko danú informáciu našiel. Keďže nechcem vytvoriť ďalší užívateľský neprívetiví wrapper(takých už existuje skutočne nespočetné množstvo), je jasné, že nemôžem užívateľa nútiť prezerať zdrojový kód a nájsť hodnotu *„id"* alebo *„class"* atribútu daného hľadaného elementu. Avšak táto informácia je nutná pre správnu funkčnosť wrappera správ z určitého spravodajského portálu. Z toho dôvodu spomínané informácie určím tzv. samo – učiacimi algoritmami(*„self learning algorithms"* alebo aj *„machine learning algorithms"*).

4.1 Samo – učiace algoritmy v semi – automatickej extrakcií správ

Samo – učiaci algoritmus je algoritmus, ktorý je schopný učenia sa na základe vyhodnocovania a spracovávania vstupných dát[26]. Takýto algoritmus potrebuje vstupnú množinu tzv. tréningových dát, o ktorých bezpečne vieme, že splňujú určitú podmienku alebo pravidlo, kto-ré chceme aby sa algoritmus naučil. Na základe týchto dát vytvorí záver, ktorý následne aplikuje na reálne dáta. V našom prípade teda vstupnou, tréningovou množinou dát budú HTML dokumenty spravodajského servera, na ktorých sa vyskytujú správy. Definovať vstup wrappera ako množinu dokumentov obsahujúcich správu je určite užívateľsky prívetivejšie ako nútiť užívateľa sledovať zdrojový kód stránky a vyhľadať požadované *„id"* či *„class"*. V ideálnom prípade stačí jednoprvková množina. Hodnota získaného *„id"* alebo *„class"* určitého elementu sa následne použije ako pravidlo detekcie správy v dokumente. Navyše vďaka danému atribútu sme schopní presne extrahovať telo správy bez nutnosti analýzy každej stránky. Vďaka

[26] Táto definícia vychádza z [22]

popísanému princípu je semi – automatický princíp tým najefektívnejším a najpresnejším spôsobom extrakcie správ z uniformných dokumentov určitého spravodajského webu.

4.1.1 Detekcia bloku obsahujúceho telo správy

Samotná detekcia tela správy v rámci HTML dokumentu je pomerne zložitý problém. Najpoužívanejšom metódou na detekciu tzv. pageletu stránky je porovnávacia metóda popísaná v kapitole 2.1.2. Na spravodajských weboch sa okrem samotného tela správy menia aj odkazy na ďalšie správy nachádzajúce sa najčastejšie tesne pod telom správy a aj iné elementy. Tak je pre nás táto metóda nepoužiteľná, keďže najčastejšie detekuje najbližšieho predka bloku s telom správy a bloku obsahujúceho tieto odkazy. Tento predok najčastejšie zastrešuje takmer celé telo dokumentu. Vďaka tomu pri použití tejto metódy dochádza pri extrakcii k výrazným informačným presahom oproti očakávanému výstupu a zároveň tento blok obsahujú aj stránky na ktorých sa správa nevyskytuje, čo má za následok katastrofálny úpadok presnosti wrappera a takýto wrapper by bol nepoužiteľný. Z toho dôvodu je táto metóda pre riešenie problému detekcie tela správy nevhodná.
Druhou veľmi používanou metódou na detekciu pageletu je metóda „Text to Tag Ratio“, tzv. TTR popísaná v kapitole 2.1.4.1. Táto metóda ráta pomer čistého textu k textu v tagoch HTML dokumentu. Pagelet je detekovaný na základe klastrov vytvorených z daných hodnôt.
Avšak ako už bolo spomenuté pri porovnávacej metóde, v tesnej blízkosti za samotným telom stránky sa nachádzajú odkazy na iné správy, ktoré štandardne pozostávajú z titulku a opisu danej správy. Keďže titulok a opis je pomerne rozsiahly text, budú všetky tieto odkazy zahrnuté do jedného klastra s telom správy a znova dôjde k informačnému presahu. Navyše táto metóda ignoruje štruktúru dokumentu a nie je ňou možné určiť niektorý z elementov HTML dokumentu. Takže táto metóda tak ako bolo navrhnutá a popísaná v [10] je pre nás nepoužiteľná. Avšak stále platí, že čistý text v prirodzenom jazyku v samotnom tele správy je spravidla ten najdlhší text v HTML dokumente.

4.1.1.1 Detekcia tela správy prostredníctvom modifikácie TTR metódy

Metóda TTR pracuje s riadkami dokumentu. To je pre prácu s HTML dokumentmi nevýhodné. A preto si vytvorím vlastnú metódu na detekciu bloku HTML dokumentu obsahujúce telo správy. Táto metóda bude principiálne vychádzať z metódy TTR. Namiesto výpočtu TTR hodnoty pre jednotlivé riadky dokumentu, si vypočítame hodnotu, ktorá bude reprezentovať počet znakov vnútorného textu daného elementu. K tejto hodnote pripočítame počet znakov vnútorného textu priamych potomkov daného elementu. Tento postup by bol dostačujúci v prípade, že by v žiadnom HTML dokumente nebol text formátovaný za pomocí elementov ako sú napr. *„span“* a *„div“*, ktoré v sebe ešte obsahujú elementy typu *„p“* , *„strong“, „h1“* a ďalšie. Zjednodušene povedané, že by text správy v DOM reprezentácií dokumentu nebol rozhodený do elementov ktoré sú od seba vzdialené viac ako je jedna úroveň v stromovej reprezentácií dokumentu. T.j. jednotlivé elementy obsahujúce správu by boli buď susedia, deti daného elementu alebo deti suseda daného elementu. Na základe pozorovania ale viem, že text správy je najčastejšie rozdelený do elementov, ktoré sú od seba vzdialené dve úrovne, niektoré až tri a viac, ale ich percento a vplyv na presnosť algoritmu je minimálny. Preto sme nútení pri výpočte spomínanej hodnoty reprezentujúcej počet znakov textu nútení prejsť aj deti detí daného elementu a pripočítať počet znakov týchto elementov. Hľadaným obalovým elementom tela správy je potom element s najväčšou touto hodnotou resp. s najväčším počtom vnútorným znakov v ňom, jeho deťoch a deťoch jeho detí. Tento algoritmus je pomerne jednoduchý a keďže pracuje s elementmi nepotrebuje už vykonávať klastrovanie.

Pseudokód 4.1 Algoritmus detekcie bloku obsahujúceho telo správy

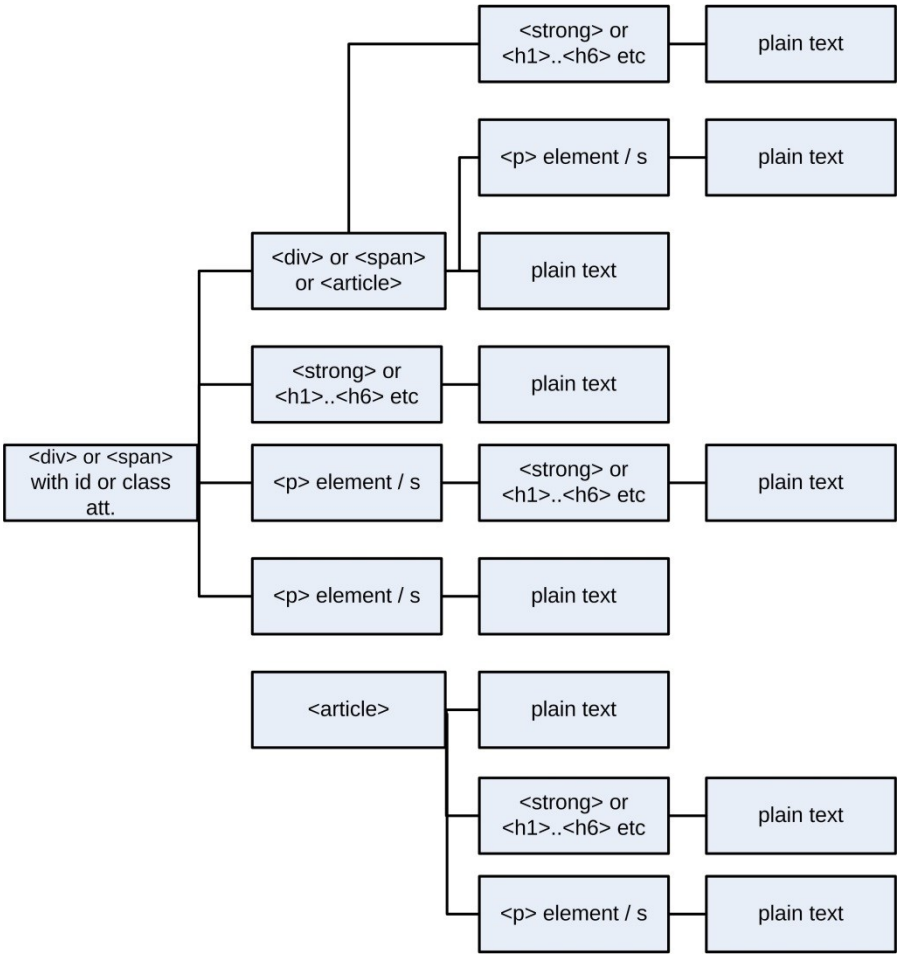
```
input h← HTML source code
begin
  Remove all script, remark tags, styles
  elements ← set of all indentifikable elements by id or class atribute in h
  maximalLength ← 0
  elementIdentifier ← string.Empty
  for each element e to elements do
  nodetextLength ← 0
  if e.hasChildNodes then
     for each element ech to e.ChildNodes
      if ech.name is p, strong, h1...h6 then
        nodetextLength ← nodetextLength + number of non-tag characters in ech
      else if ech.hasChildNodes then
          for each element ech2 to ech.ChildNodes
           nodetextLength ← nodetextLength + number of non-tag characters in ech2
        end for
      end if
    end for
    if nodetextLength > maximalLength then
      maximalLength ← nodetextLength
      elementIdentifier ← e.attributes["id"].value // or class
    end if
 end for
return elementIdentifier
```

Pri testovaní tento algoritmus fungoval skvele pre tri z piatich webov. Pre jeden z piatich webov fungoval ale s informačným presahom, t.j. došlo k extrakcií viac ako len tela danej správy. A pre zvyšný web nefungoval vôbec. Táto nefunkčnosť je spôsobená využívaním tabuliek, prípadne vnorených tabuliek(tabuľky v tabuľke) pri štruktúrovaní tela správy. Týmto nie je myslené, že telo správy obsahuje tabuľku, v takom prípade je algoritmus schopný správnej detekcie bloku s telom správy, ale že telo správy je zložené zo stĺpcov a riadkov gigantickej tabuľky prípadne viacerých tabuliek. Najčastejšie sú takto formátované HTML dokumenty, ktoré sú vytvorené v určitom *„web buildery“*, t.j. programe, kde si užívateľ nakreslí prípadne poskladá webovú stránku v editore a ten mu následne vygeneruje zdrojový kód. V minulosti bolo používanie tabuľky na štruktúrovane webových stránok veľmi časté, v súčasnej dobe sa takmer nepoužíva, nahradzujú ho CSS vlastnosti starajúce sa o umiestnenie elementov na stránke.

Ďalšími príčinami nefunkčnosti môže byť nesprávne použitie tagu *„article“*. Niektoré weby tento tag vnoria do tela správy, s čím si algoritmus stále dokáže poradiť, ale keď je doň vložená rozvetvená štruktúra elementov, tak algoritmus nebude fungovať správne. Toto vždy viedlo buď ku kritickým informačným presahom, t.j. detekovaný blok bola takmer celá stránka a nebola to unikátna časť vyskytujúca sa len v dokumentoch so správami. Alebo ku kritickým informačným podsahom, teda detekovaný blok bol len časť tela správy, čo umožňovalo identifikáciu dokumentov obsahujúcich správy, ale extrahované správy boli neúplné.

Nasledujúci obrázok (4.1 nižšie) zachytáva príklad štruktúry tela správy, pri ktorej bude algoritmus úspešný. Každý z uvedených elementov sa môže vyskytovať ľubovoľný počet krát. Pod pojmom „*plain text*“ rozumieme samotný text správy v prirodzenom jazyku. Zo získaného elementu si algoritmus zapamätá hodnotu jeho *„id“* alebo *„class“* atribútu. Tým detekcia tela správy resp. bloku obaľujúceho telo správy končí. Získane vlastnosti sa použijú pri crawlovaní daným webovým spravodajským portálom na detekciu a extrakciu samotnej správy.

Obrázok 4.1 Štruktúra tela správy identifikovateľná opísaným algoritmom



4.1.2 Dátumy správ[27]

Aj keď extrakcia dátumu správy sa na prvý pohľad javí ako triviálna úloha, podobne ako detekcia a extrakcia titulku a opisu, nie je tomu tak. S dátumami je spojené veľké množstvo problémov, ktoré som odhalil pri implementácií a testovaní svojho riešenia. Jazyk HTML špecifikuje špeciálny element *„<time>“* a atribúty *„pubdate“* a *„datetime“*[28], ktoré slúžia primárne na špecifikáciu dátumu na strojové spracovanie wrappermi, vyhľadávačmi a prehliadačmi. Atribúty *„pubdate“* a *„datetime“* obsahujú ako hodnotu dátum prípadne dátum aj s aktuálnym časom v jednom zo štandardných formátov opísateľných bežnými regulárnymi výrazmi. K dátumom sa potom môže pristupovať prostredníctvom jednoduchého XPath dotazu na elementy typu *„<time>“*. Alebo o niečo menej efektívnejšia možnosť prehľadať hodnoty všetkých atribútov na výskyt regulárnych výrazov popisujúcich dátumy v rôznych formátoch. Takéto riešenie by bolo skutočne triviálne avšak realita je veľmi odlišná. Atribút typu *„<time>“* sa na spravodajských portáloch objavuje skôr sporadicky. Podobne málo častým je aj výskyt dátumov vo formáte dobre opísateľnom regulárnym výrazom. Dátumy sú najčastejšie špecifikované slovne, alebo aspoň časť dátumu je popísaná slovne, obzvlášť na spravodajských weboch z Českej Republiky a Slovenska. Keďže sa snažím zabezpečiť funkcionalitu svojho wrappera nezávislú na jazyku správy je vytvorenie regulárneho výrazu, ktorý by nachádzal dátumy v telách správ alebo v ich okolí takmer nemožné. Vyjad-

[27] Táto kapitola rozvíja problém dátumov spomenutých v 3.3
[28] Zdroj [25]

rené v reči čísel, jeden z piatich českých alebo slovenských spravodajských portálov má správne dobre spracovateľné dátumy. Pre zahraničné obzvlášť anglo – americké weby je tento pomer dva z piatich.

Tabuľka 4.1 Príklady špecifikácie dátumov na českých spravodajských webových portáloch

Novinky.cz	`<span class="clanek-datum">Dnes 19:25</span>`
Aktualne.cz	`<p id="articleDate" class="publicDate"> Dnes 21:15 - Kyjev </p>`
Zive.cz	`<small class="article-tags">` `15. 4. 2014`

Povšimnite si predovšetkým nezmyselnosť dátumov typu „dnes“, „včera“. Tieto dátumy po extrakcií takmer strácajú význam. Užívateľ si totiž musí dátum určiť na základe vypozorovaného dáta poslednej zmeny súboru s extrahovanými správami a od toho dáta si konkrétny dátum odvodiť. Konvertor, ktorý by takéto dátumy previedol do štandardných dátumov je takmer nerealizovateľný, keďže výsledný wrapper má byť jazykovo nezávislý.

Ďalšou komplikáciou, vďaka ktorej väčšina komerčne dostupných wrapperov vracia nesprávne dátumy je dátum zachytiteľný regulárnym výrazom umiestnený buď v metadátach v hlavičke HTML dokumentu alebo niekde tesne za elementom „*<body>*“.Jedná sa o dátum daného dňa najčastejšie spojený s informáciou o mene v kalendári k danému dňu. Približne tri z piatich spravodajských portálov tam tento dátum majú a drvivá väčšina wrapperov tento dátum extrahuje keďže je opísateľný štandardným regulárnym výrazom. Avšak pre naše účely je tento dátum nepoužiteľný a jeho detekcia a priradenie k správe by rezultovalo v zavádzajúcu extrakciu, keďže by boli extrahované nepravdivé dáta.

Možnosťou ako vyriešiť problém extrakcie dátumov by bolo, rovnako ako v prípade extrakcie tela správy, naučiť sa v akom elemente sa dátum nachádza z tréningovej množiny dát. Keďže sa dátum v rámci jedného spravodajského webu bude nachádzať v rovnakom elemente, táto metóda by daný problém vyriešila. Algoritmus by vyhľadával dátumy na základe regulárnych výrazov v tréningovej množine dát. Z toho je jasné, že v prípade, že by množina neobsahovala dátumy opísateľné regulárnym výrazom tak by bola detekcia neúspešná.Táto metóda detekcie dátumov by narušila užívateľskú prívetivosť wrappera, keďže by užívateľ musel manuálne prehľadávať web na výskyt stránok obsahujúcich dátum opísateľný regulárnym výrazom, teda v niektorom zo štandardných formátov. Taktiež by bolo nutné ošetrenie, aby nedošlo k detekcií dátumu daného dňa, teda dátum publikovania správy by musel byť priamo v tele správy alebo v jeho tesnej blízkosti, povedzme napríklad maximálne o dve úrovne vyššie v stromovej štruktúre reprezentujúcej DOM daného dokumentu. Spravodajské weby sú rôzne štruktúrované, niektoré môžu mať dátum správy omnoho ďalej, iné môžu mať dátum dňa v spomínanom okruhu. Z toho plynie, že ani tak nebudeme schopný detekcie dátumu na všetkých weboch. Ďalšou nevýhodou s ktorou musíme počítať, keďže predpokladáme úplne neskúseného užívateľa je, že nebude vedieť aký je štandardný formát dátumu a všetky podporované formáty, teda tie ktoré budeme hľadať regulárnym výrazom špecifikujeme do nápovedy programu, ktorá by sa týmto stala nutnou súčasťou wrappera. Na základe spomínaných nevýhod zavrhujem toto riešenie. Keďže ani hore spomínané riešenie by nebolo sto percentne spoľahlivé a použiteľné môžem za správne riešenie považovať aj nejaké riešenie, ktoré nebude funkčné pre všetky spravodajské weby ale bude zastrešovať aspoň ich väčšinu.

4.1.2.1 Extrakcia dátumov zo správ

Empirickým pozorovaním čo najväčšieho množstva webových portálov som zistil, že väčšina spravodajských serverov špecifikuje svoje dátumy v elementoch ktorých množina atribútov síce neobsahuje W3C definované atribúty *„datetime“* a *„pubdate“*, ale obsahuje atribúty *„id“* alebo *„class“*, ktorých hodnoty obsahujú slová *„date“*, *„time“*. Testovanie metódy založenej na tejto informácií prinieslo pozitívne výsledky pre štyri z piatich zahraničných webov, ale len pre dva z piatich českých spravodajských serverov. Z toho dôvodu som do hľadaných slov pridal aj výraz *„datum“*. Pre spresnenie výsledkov a minimalizovanie šance výskytu portálu, kde by táto metóda nefungovala som navyše pridal kontrolu textu stránky a hodnôt atribútov prostredníctvom najznámejších regulárnych výrazov opisujúcich dátumy. V prípade, že by sa nenašiel žiaden element, ktorého identifikátor alebo trieda by obsahovala hore spomínané výrazy bude dátum doplnený nájdeným dátumom prostredníctvom regulárnych výrazov. Takto implementovaná metóda nájde správne dátumy pre štyri z piatich spravodajských portálov, vrátene českých a slovenských spravodajských portálov.

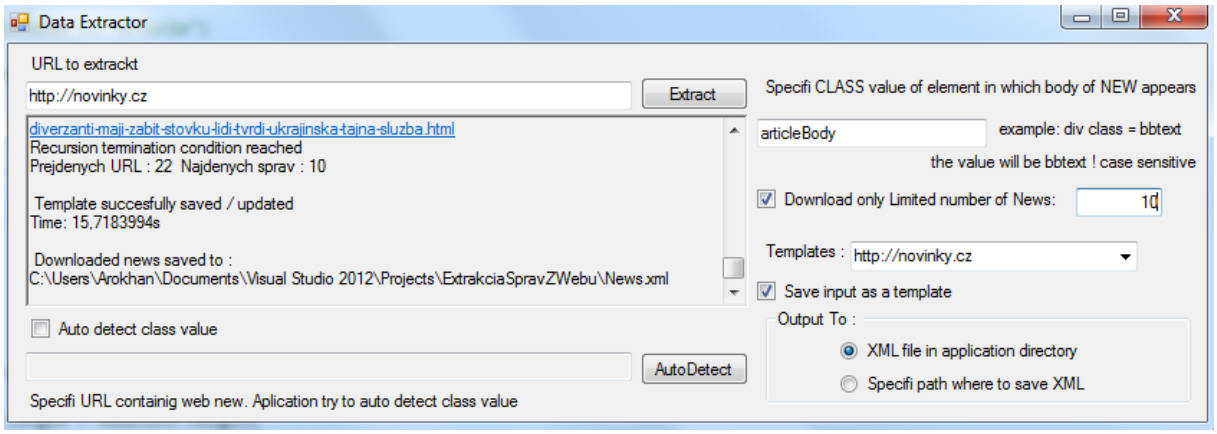
4.2 Semi – automatický wrapper na extrakciu správ – riešenie

V tejto a predchádzajúcej kapitole som opísal metódy extrakcie a crawlovania v rámci spravodajských portálov. Funkčnosť týchto metód dokumentujem vlastným užívateľsky príveti-vým a jednoduchým wrapperom správ. Okrem v predošlých kapitolách spomínaných vlastností tento wrapper disponuje pár dodatočnými funkciami zvyšujúcimi jeho užívateľskú prívetivosť a použiteľnosť:

- Po úspešnej extrakcií si užívateľ môže uložiť vstup do tzv. templatov – šablón a v prípade nutnosti novej extrakcie nemusí špecifikovať vstup znova. Šablóny sa ukladajú do XML súborov v zdrojovej zložke programu a tak sú prenosné z užívateľa na užívateľa.
- Umožňuje nastaviť počet sťahovaných správ
- Počas behu je počítaný počet prejdených dokumentov nutných na extrakciu daného počtu správ a vyrátaný čas behu aplikácie

Vytvorený wrapper ako zástupca skupiny poloautomatických wrapperov je maximálne presný a pomerne výkonný. Pre správnu funkčnosť crawlera vstavaného do wrappera je nutné vytvárať pamäť prejdených URL. Každé nájdené URL je kontrolované, či nebolo analyzované.
To zabraňuje viacnásobnému analyzovaniu jedného dokumentu a zároveň spomaľuje wrapper pokiaľ prechádza veľkým webom obsahujúcim mnoho dokumentov alebo sťahuje veľké množstvo správ.

Obrázok 4.2 Ukážka GUI vytvoreného wrappera



4.2.1 Testovanie vlastného semi – automatického wrappera

Za dve najdôležitejšie kritéria testovania považujem presnosť a efektivitu, keďže poloautomatické wrappery sú využívané a propagované práve kvôli vysokej presnosti detekcie a extrakcie dát a veľkej efektivite, ktorú umožňuje princíp strojového učenia, vďaka ktorému po naučení pravidiel nedochádza k analýze každého dokumentu, ale len k aplikácií získaných pravidiel.

4.2.1.1 Testovanie presnosti vytvoreného wrappera:

Legenda k tabuľke 4.2 nižšie:

- **manuálny** – hodnota atribútu *„id“* alebo *„class“* elementu obaľujúceho telo správy sú zadané manuálne zo zdrojového kódu, alebo pre extrakciu bola vybraná šablóna
- **automatický** – na detekciu bloku s telom správy bol využitý učiaci princíp popísaný v kapitole 4.1.1
-  - telo správy stiahnuté v poriadku bez presahov alebo podsahov
- **presah** – telo správy stiahnuté s určitým nekritickým informačným presahom(telo správy obsahuje navyše niektoré informácie, napr. komerčný obsah, dva krát sa vyskytuje podnadpis a pod.) alebo sa do detekovanej množiny správ dostanú nesprávy
-  - telo správy nebolo stiahnuté, pri manuálnej detekcií sa toto môže stať len ak užívateľ vloží nesprávne dáta napr. preklep pri zadávaní, alebo použije neaktuálnu šablónu
- **podsah –** je extrahovaná len časť tela správy, čo je považované za nepriaznivý výsledok a je závažnejším problémom ako presah
- **!!!** – daný web sa skladá z neuniformných podwebov, wrapper funguje perfektne pre každý z podwebov samostatne, ale nefunguje na celý web všeobecne

Tabuľka 4.2 Testovanie wrappera na najznámejších českých a slovenských spravodajských portáloch

Spravodajský portál	Spôsob detekcie tela správy	Telo správy	Dátum správy
www.idnes.cz	manuálny		
	automatický		
www.novinky.cz	manuálny		
	automatický		
www.aktualne.cz	manuálny		
	automatický		
www.zive.cz	manuálny		
	automatický		
www.sme.sk	manuálny		 [29]
	automatický		 [30]

[29] Uvedený spravodajský portál *www.sme.sk* neuvádza u drvivej väčšiny správ dátumy, z toho
[30] dôvodu som musel uviesť dátum ako nie nutnú časť správy, potom bola extrakcia bezproblémová.

www.topky.sk[31]	manuálny	presah	
	automatický	presah	
www.aktuality.sk	manuálny		
	automatický		
www.24hod.sk	manuálny		
	automatický	[32]	
www.cas.sk	manuálny		[33]
	automatický		

Z uvedenej tabuľky vyplýva, že extrakcia dátumu je dokonca omnoho zložitejšou úlohou ako je samotná extrakcia tela správy z dokumentu. Táto úloha je dokonca tak zložitá, že by nebola riešiteľná ani manuálnym wrapperom. Presnosť vytvoreného wrappera je vyššia pre slovenské a české weby ako pre tie zahraničné. A to aj napriek tomu, že zahraničné weby viac dodržujú štandardy tvorby webu špecifikované konzorciom W3C. Tento fakt je zapríčinený tým, že zahraničné weby(špeciálne tie veľké) sa bránia rôznym wrapperom a snažia sa zabrániť zneužívaniu svojich správ. Týmto problémom sa budem podrobne zaoberať v kapitole 6 .

4.2.1.2 Testovanie efektivity vytvoreného wrappera

Ďalším dôležitým kritérium je efektivita. Pod pojmom efektivita pri tomto testovaní budeme rozumieť čas potrebný na nájdenie, detekciu a extrakciu špecifikovaného počtu správ. Ďalej budeme sledovať priemerný čas potrebný k extrakcií jednej správy. Veľmi dôležitý je taktiež počet dokumentov, ktoré musí wrapper prehľadať aby sa dopracoval k správe. Ten udáva samotná štruktúra spravodajského servera. Niektoré weby totiž obsahujú len zanedbateľný počet správ v porovnaní s celkovým počtom HTML dokumentov daného webu Predpokladám, že samotný čas nutný k extrakcií jednej správy je predovšetkým závislý na počte dokumentov, ktoré musí wrapper analyzovať, kým sa dopracuje k dokumentu, ktorý obsahuje správu. Taktiež predpokladám, že čas potrebný k samotnej extrakcií správy je menší ako je čas, ktorý je potrebný k prechodu(crawlovaniu) jednotlivými dokumentmi a ich analýze. Z toho vyplýva, že najneefektívnejší bude tento wrapper na weboch obsahujúcich pomerne nízky počet správ v porovnaní s celkovým počtom dokumentov daného webu. Tieto hypotézy vychádzajú zo všeobecných vlastností poloautomatických wrapperov. Rozhodol som sa otestovať tieto hypotézy. Týmto testovaním by som chcel dokázať výhody použitia semi – automatických wrapperov vo všeobecnosti a taktiež demonštrovať správnosť návrhu a riešenia mnou navrhnutého a implementovaného wrappera. Špeciálne pri tomto testovaní budem sledovať závislosť priemerného času nutného k detekcií / extrakcií jednej správy od priemerného počtu dokumentov, ktoré je nutné prejsť k danej detekcií. Taktiež budem sledovať či nenastane výrazné spomalenie pri spracovaní väčšieho množstva dokumentov, keďže aby sa zabránilo opakovaniu je

[31] Web *http://www.topky.sk* neobsahuje veľké množstvo správ, ale dá sa tam dohľadať niekoľko desiatok správ. Niektoré nemajú uvedený dátum, tie ktoré majú boli extrahované vrátane dátumu.

[32] Web *http://www.24hod.sk* obsahuje množstvo textu v ľavom menu, v prípade zadania vstupných stránok detekcie s krátkymi správami dôjde k nesprávnej detekcií a nefunkčnej extrakcií.
Taktiež sa v rámci tohto webu vyskytujú rozličné typy kódovania a aj keď je web slovenský ako jazyk uvádza češtinu.

[33] Web *http://www.cas.sk* definuje dátumy, ktoré sú síce v samostatnom elemente, ale ten nie som schopný rozpoznať a tak sa hľadajú dátumy regulárnymi výrazmi, jediný dátum v správnom formáte je ten dnešný a tak ku každej správe wrapper priradzuje dnešný dátum viď kap. 4.1.2

vstavaná v crawlery pamäť prejdených URL dokumentov. Vždy keď crawler narazí na URL kontroluje, či už bolo analyzované alebo nie. Aby som minimalizoval dopad pamäte, je realizovaná prostredníctvom „*hashsetu*[34]“, t.j. kolekcie „*stringov*“, ktorá v použitom jazyku C# sľubuje veľmi efektívne vyhľadávanie prvkov v danej kolekcií[35]. Nasledujúca tabuľka 4.3 uvedená nižšie opisuje spôsob testovania, ktorý som zvolil. Keďže novinky.cz sú najmenším zo sledovaných portálov uviedol som kompletnú tabuľku zachytávajúcu priebeh testov. Pre ostatné weby uvediem len skrátenú tabuľku.

Testy efektivity pre *www.novinky.cz*

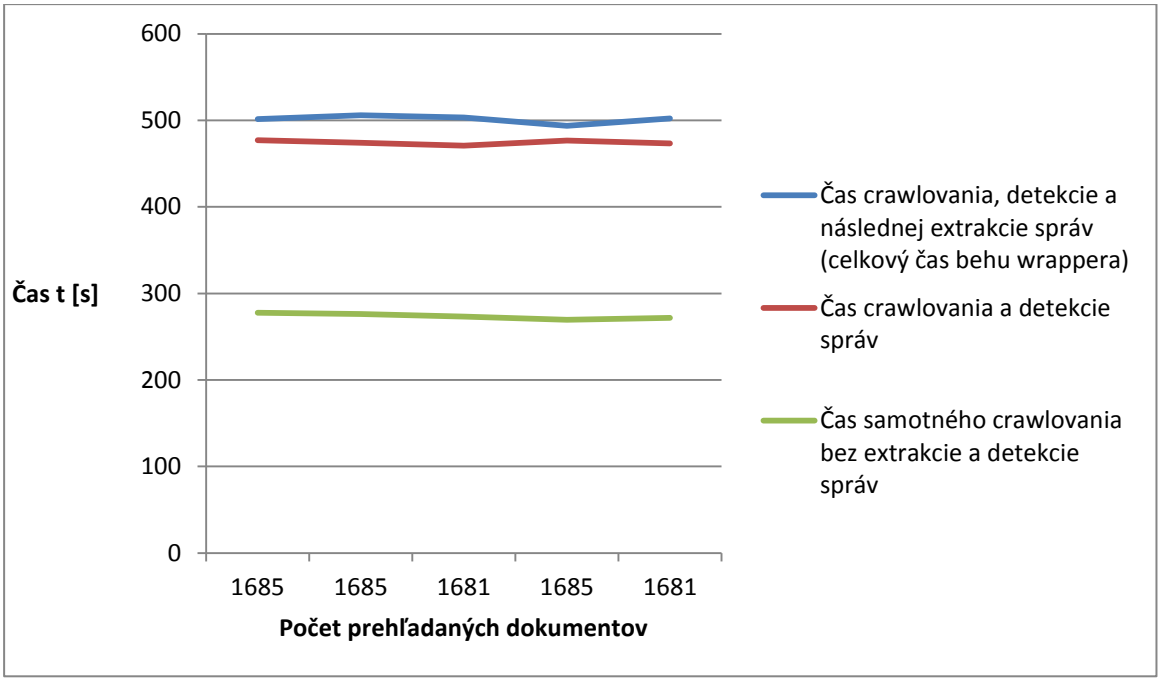
Tabuľka 4.3 Testovanie efektivity extrakcie správ z http://www.novinky.cz ku dňu 17.04.2014

novinky.cz	**prejdených URL 100 správ [s]**	**prejdených URL 200 správ [s]**	**prejdených URL 300 správ [s]**	**prejdených URL Celý server Počet správ/ čas [s]**	
1.	299,00	560,00	801,00	1195,00	
	105,30	211,61	309,14	425,00 / 480,00	
2.	298,00	560,00	801,00	1191,00	
	109,56	205.21	310,58	423,00 / 479,20	
3.	299,00	560,00	801,00	1191,00	
	107,69	207,19	322,50	423,00 / 461,68	
4.	298,00	560,00	802,00	1191,00	
	110,08	216.31	327,73	423,00 / 459,65	
5.	298,00	560,00	802,00	1193,00	
	107,36	203,78	306,21	424,00 / 469,85	
priemer	298,40	560,00	801,40	1192,20	
	108,00	208,82	315,23	423,60 / 470,08	
priemerná odchylka	0.48	0,00	0,48	0,84	
	1.46	4.11	7,91	0,72 / 7.62	***priemerne***
čas na správu [s]	1,08	1.04	1,05	1.11	**1,07**
odchýlka pre správu[s]	0.01	0,02	0,03	0,02	**0,02**
URL/ správa	2,98	2,8	2,67	2,81	**2,82**

[34] Zdroj: http://msdn.microsoft.com/en-us/library/bb359438%28v=vs.110%29.aspx

[35] Viď http://geekswithblogs.net/BlackRabbitCoder/archive/2011/06/16/c.net-fundamentals-choosing-the-right-collection-class.aspx a http://social.msdn.microsoft.com/Forums/vstudio/en-US/8fcd76cc-2665-4e37-b140-dbb-25e447bb0/fastest-way-to-do-string-comparisons-hashsetcontains?forum=csharpgeneral

Obrázok 4.3 Graf zachytávajúci časy crawlovania, crawlovania a detekcie, crawlovania, detekcie a extrakcie správ na servery http://www.noviky.cz ku dňu 28.04.2014



Uvedený graf zachytáva päť rôznych testovaní v rámci serveru www.novinky.cz. Jednotlivé priamky ukazujú rozdiel v celkovom času behu aplikácie v závislostí na činnosti vykonávanej daným programom. Pomerne výrazný čas nutný k detekcií správ je spôsobený najmä problematickou a veľmi neefektívnou identifikáciou dátumov v rámci HTML dokumentov, ktorá plynie z rozsiahleho nedodržiavania štandardov.

Testy efektivity pre pár ďalších českých a slovenských webov

Jednotlivé výsledky sú už spriemerované výsledné hodnoty. Testy prebehli tak ako tomu bolo v stati *„Testy efektivity pre www.novinky.cz“*. Uvádzam len skrátenú tabuľku pre zachovanie prehľadu v dokumente.

Tabuľka 4.4 Testy efektivity na niekoľkých známych českých a slovenských spravodajských portáloch ku dňu 28.04.2014

Spravodajský portál	Čas na jednu správu [s]	Odchýlka merania [s]	URL na jednu správu	Odchýlka merania
www.sme.sk	4,84	0,04	11,20	0,23
www.aktualne.cz	1,58	0,03	3,09	0,16
www.idnes.cz	2,39	0,03	4,73	0,31
www.zive.cz	2,56	0,03	4,86	0,19

Vyhodnotenie testov efektivity

Z grafu 4.3 uvedeného vyššie vyplýva, že najpomalšou časťou wrappera je crawler, resp. jeho časť zodpovedná za navigáciu a stiahnutie lokálnej kópie daného dokumentu.Samotná analýza dokumentu na výskyt správy je druhou najpomalšou časťou navrhnutého wrappera, toto zapríčiňuje hlavne komplikované vyhľadávanie a extrakcia dátumov, ktorej som sa už venoval v kap. 4.1.2.1. Samotná extrakcia správy nadväzuje na detekciu správy a je časovo málo náročná a na čase behu aplikácie sa podieľa minimálne. Fakt, že extrakcia a detekcia sú úlohy menej náročné ako samotné crawlovania(a to je k nim prirátaný aj čas potrebný na serializáciu nájdených správ) potvrdzuje správnosť návrhu a poukazuje na nesmiernu efektivitu poloautomatických wrapperov všeobecne.

Portál novinky.cz je prípadom ideálneho použitia navrhnutého wrapperu. Z výsledkov testov vyplýva, že pomerovo je počet dokumentov obsahujúcich správu a počet dokumentov bez správy v rámci celého webu rovnaký, čo sa prejavuje v približnej nemennom čase extrakcie jednej správy pri extrakcií rozličných počtov správ. Taktiež to poukazuje na fakt, že správy sú v rámci daného webu rovnomerne rozdelené. Tento portál obsahuje len pomerne málo správ a zároveň približne každý tretí dokument na ktorý daný portál odkazuje je dokument so správou, vďaka čomu je extrakcia tohto webu veľmi rýchla.

Napr. portál idnes.cz je presným opakom spomínaného servera novinky.cz. Tento portál je veľmi rozsiahlym portálom obsahujúcim nesmierne množstvo HTML dokumentov a ešte väčšie množstvo externých odkazov, ktoré musia byť tiež kontrolované, aj keď len na úrovni URL daných dokumentov. Taktiež tento portál pozostáva z pomerne rozsiahlych kategórií, pričom niektoré neobsahujú žiadnu správu(ako napr. TV program a iné). Tento fakt ma viedol k nutnej paralelizácií danej aplikácie, tak aby boli analyzované viaceré podweby súčasne. Pri paralelizácií si však treba dať pozor na prílišne množstvo vlákien, pokiaľ bude realizovaná extrakcia veľkým množstvom vlákien, čo môže viesť k zaťažovaniu serveru, bude aplikácia serverom zabanovaná. Taktiež kvôli vstavanej pamäti, ktorá bráni duplicitnej analýze dokumentov, musia všetky vlákna zdieľať tú istú kolekciu, ktorá musí byť uzamykaná, čo znemožňuje maximálnu možnú mieru efektivity paralelizácie. Weby pozostávajúce z veľkého počtu dokumentov a obsahujúce nesmierne množstvo hypertextových odkazov, ktoré sú kontrolované crawlerom, sú najhorším možným prípadom použitia navrhnutého wrappera a to sa odzrkadlí aj na čase extrakcie správ z daného webu(viď Tabuľka 4.4). Z danej tabuľkytaktiež plynie závislosť času potrebného k extrakcií jednej správy od počtu prejdených dokumentov. Keďže sa jednalo o rozličné weby, kde dochádzalo k rôznym spôsobom extrakcie dátumov, nemohol som túto závislosť zaviesť do grafu. Kvôli paralelizácií aplikácie nie je možné túto závislosť zachytiť ani v rámci jedného webu, keďže práve vďaka paralelizácií je odstránený problém podkategórií neobsahujúcich žiadne správy a tak počet prejdených URL potrebných k nájdeniu jednej správy ostáva v rámci celého webu nemenný.

4.2.1.3 Pamäťové nároky aplikácie

Najviac operačnej pamäte zaberajú extrahované správy, ktoré ešte neboli zapísané do súboru. Kvôli minimalizovaniu náročných IO operácií dochádza k zápisu nájdených správ vždy po skončení prehľadávania určitého podwebu resp. kategórie. V pamäti sa taktiež uchováva kolekcia prehľadaných URL. Z toho plynie, že najväčšie pamäťové nároky budú počas prehľadávania rozsiahlych podkategórií obsahujúcich množstvo správ a na konci behu aplikácie. Najväčšie zaznamenané nároky počas testovania boli takmer 200 MB pri testovaní webu idnes.cz. Priemerné pamäťové nároky sú cca 40 MB.

4.2.1.4 Jednoduchosť použitia a užívateľská prívetivosť

Pri vypracovaní tejto práce som testoval veľké množstvo wrapperov. Najväčším problémom všetkých z nich bolo ich samotné použitie. Na základe komplikovaného vstupu programu som množstvo týchto wrapperov ani nedokázal použiť. Z toho dôvodu je jednoduchý vstup najdôležitejším kritériom wrappera a je dôležitejší než jeho efektivita a presnosť. Užívateľskou prívetivosťou som sa zaoberal už v predchádzajúcich kapitolách. Maximálna hodnota tohto kritéria je zaistená poloautomatickým princípom, vďaka ktorému užívateľ špecifikuje len stránku/ stránky obsahujúce správu a wrapper sa postará o ostatné. Taktiež vysokú mieru prívetivosti zabezpečujú vstavané pravidlá pre extrakciu dátumov, aj keď tieto pravidlá spôsobujú zníženie efektivity a aj presnosti wrappera, keďže nie je možné napísať všeobecne platné a zároveň efektívne pravidlá na extrakciu dátumov. Keďže užívateľskú prívetivosť a jednoduchosť použitia považujem za hlavné kritérium hodnotenie wrapperov ponechám extrakčné pravidlá pre dátum vstavané.V prípade nutnosti zefektívnenia wrappera alebo zlepšenie jeho presnosti je možné vstavané pravidlá vymeniť za pravidlo špecifikované na vstupe.

4.3 Súčasné trendy v oblasti wrapperov

Poloautomatické wrappery sú v súčasnosti najpoužívanejším riešením problému extrakcie štruktúrovaných dát z Internetu. Najpoužívanejším trendom v oblasti extrakcie je tzv. extrakcia na základe vzorov (z ang. „*patterns*“). Takýmto typom extrakcie som sa zaoberal v kapitole 2.1.1.2 Mnohé wrappery, ktoré takto pracujú sú autormi označované ako automatické, toto označenie, ale nie je pre mnohé z nich pravdivé. Za wrappery blížiace sa automatickým považujeme wrappery, ktoré prehľadávajú HTML dokumenty a vzory si vytvárajú automaticky na základe nájdených opakovaní v štruktúre dokumentu. Následne vrátia množinu vzorov a užívateľ si vyberie vzor, ktorý najlepšie odpovedá jeho požiadavkám a wrapper extrahuje dáta odpovedajúce danému vzoru. Wrappery, ktorým je nutné na vstupe programu definovať vzor a následne je v dokumentoch tento vzor vyhľadávaný označujeme ako poloautomatické. Tieto wrappery sa od seba líšia len v spôsobe zadávania vstupných vzorov. Tie sofistikovanejšie ponúkajú kooperáciu s prehliadačom a umožňujú špecifikovať vzory ich označením v grafickej reprezentácií stránky. Všetky takéto wrappery boli platené a mal som možnosť vyskúšať si len „*trial*“ verzie. Ceny licencií sa pohybovali od 600 do 1000 euro za ročnú licenciu, čo dokazuje, že má zmysel venovať sa tejto problematike. Tie menej sofistikované nástroje príjmami vstup ako jednoduché dáta špecifikované priamo regulárnym výrazom, alebo príkladom daných dát a regulárny výraz si aplikácia vytvorila sama princípmi popísanými v 2.1.1.1. Tieto wrappery boli taktiež platené a k dispozícií na testovanie bola len „*trial*“ obmedzená verzia. Avšak ako som už spomínal skôr, žiaden zo stávajúcich testovaných wrapperov nebol vhodný k extrakcií správ, dokonca ani tie platené. Spôsobuje to fakt, že jednému vzoru môže odpovedať viacero typologicky odlišných ale vzhľadovo a štrukturálne takmer rovnakých dokumentov. Napríklad životopisná stránka určitej osobnosti pozostáva z titulku, často podnadpisu, vyskytuje sa na nej dátum narodenia, ktorý je navyše veľmi často blízky nadpisu prípadne fotografia v hornej časti dokumentu a samotný životopis osobnosti. Táto štruktúra je v podstate identická so štruktúrou správy. Mnohé spravodajské weby okrem správ obsahujú práve krátke životopisy či popisy osobností, ktorých sa správy týkajú. Komerčne dostupné wrappery fungujúce na princípe vzorov tak vracajú okrem správ práve tieto životopisy či krátke opisy osobností vo svojom výsledku.

Wrapper opísaný v kap. 4.2 je tiež vo svojej podstate založený na vzoroch. Vzor v ňom, t.j. správa odpovedajúca štruktúre popísanej v 3.1 , je definovaný v rámci kódu a premenlivá časť správy sa definuje manuálne na vstupe, alebo sa naučí z tréningovej množiny. Práve fakt, že telo správy je identifikované špeciálnou hodnotou bloku HTML dokumen-

tu, v ktorej sa nachádza, umožňuje môjmu wrapperu na rozdiel od tých komerčne dostupných odlíšiť správu od životopisu. Tak môj wrapper prekonáva dostupné wrappery na trhu pre úlohu extrakcie správ z webu. Avšak komerčné wrappery fungujúce na princípe vzorov sú všeobecné a umožňujú extrakciu akýchkoľvek štruktúrovaných dát opísateľných určitým vzorom. Môj wrapper toto neumožňuje. Viac informácií o samotných vzoroch a spôsobe ich detekcie či extrakcie, prípadne spôsoby ich vyhľadávania v rámci dokumentu nájdete na [16] a následne [17].

4.4 Poloautomatické wrappery zhrnutie

Poloautomatické wrappery sú najpoužívanejším riešením problematiky extrakcie štruktúrovaných dát. Miera ich efektivity a presnosti je závislá na presnosti a obsiahlosti špecifikovaného vstupu. Všeobecne platí, že čím komplikovanejší vstup tým efektívnejší a presnejší je daný wrapper. Tieto wrappery umožňujú vyhľadávať a extrahovať štruktúrované dáta v reálnom čase pri zachovaní pomerne nízkych výkonnostných nárokov na zariadenie, na ktorom sú spúšťané. Tento fakt zabezpečil ich výrazné rozšírenie v súčasnosti. Mnohé poloautomatické wrappery sú tak komplikované, že bežný užívateľ bez obsiahlej užívateľskej príručky alebo iného spôsobu vysvetlenia činnosti daného wrappera, nedokáže tento wrapper používať.

5. Automatická detekcia a extrakcia správ

V predchádzajúcej kapitole som sa zaoberal poloautomatickými wrappermi, tie k svojej správnej funkcií musia pracovať v rámci uniformnej štruktúry HTML dokumentov(webové stránky jedného portálu, výsledok jedného frameworku atď.) alebo sú schopné pracovať len s dátami s určitou uniformnou štruktúrou dát. Nastáva teda problém čo s neuniformnými HTML dokumentmi resp. ako riešiť extrakciu a detekciu správ z na sebe nezávislej množiny HTML dokumentov(napr. výsledky vyhľadávača pre určité kľúčové slová). Prípadne ako riešiť extrakciu dát, ktoré majú rovnaký význam, ale kvôli nesmiernej rozmanitosti Internetu a jeho užívateľov majú tak rozdielnu štruktúru, že nie sú opísateľné jednoduchým vzorom, ktorý by ich presne identifikoval. Tento problém sa rieši za pomoci tzv. automatických wrapperov. Plne automatický wrapper, ktorý by bol schopný extrakcie akýchkoľvek štruktúrovaných dát zatiaľ neexistuje. Avšak ak vieme aké dáta ma daný automatický wrapper extrahovať sme schopní definovať určité všeobecné pravidlá, na základe ktorých budú dané dáta datekované a extrahované. Presnosť a efektivita automatických wrapperov sa s tými poloautomatickými ani nedá porovnať, ale nespornou výhodou týchto wrapperov je ich jednoduchosť použitia.

5.1 Plne automatický wrapper správ - návrh

Je taký wrapper, ktorý len na základe interne definovaných pravidiel, bez akéhokoľvek vstupu užívateľa, detekuje a extrahuje správy. Pre súrodú množinu HTML dokumentov nemá zmysel, pretože jeho efektivita a presnosť je v porovnaní s poloautomatickým wrapperom veľmi malá. Z toho dôvodu bude vstupom pre takýto wrapper množina HTML dokumentov vrátená vyhľadávačom „*google*“, tak bude zabezpečená na sebe nezávislá a nesúrodá množina HTML dokumentov na vstupe. Aby bolo toto možné, musím do svojho projektu vložiť „*Google Custom Search API*[36]“, je to API, ktoré umožňuje prehľadávanie Internetu prostredníctvom najznámejšieho vyhľadávača „*www.google.com*“. Vstupom bude hľadaný výraz, výstupom bude množina sto JSON objektov reprezentujúcich prvých sto dokumentov nájdených daným prehliadačom. Získané JSON objekty následne deserializujem a získam si ich URL. Tým si získam kolekciu URL odkazov na rôznorodé HTML dokumenty, ktoré budem automatickým wrapperom analyzovať. Google Custom Search API je distribuované v platenej a neplatenej verzií, neplatená verzia vykoná maximálne 100 vyhľadávaní denne, pre viac vyhľadávaní je nutné mať platenú verziu. Rozlišovanie je realizované prostredníctvom potreby registrácie projektu, ktorý dané API využíva v tzv. „*google developers console*“[37]. Na jedno vyhľadávanie, resp. jednu špecifikáciu kľúčových slov vyhľadávania pripadá desať výsledkov, v prípade, že chceme viac výsledkov bude toto vyhľadávanie považované za $\frac{počet\ výsledkov}{10}$ vyhľadávaní. Pre testovacie účely požadujem sto výsledkov, čo bude považované za desať vyhľadávaní. Keďže denný limit je sto vyhľadávaní som schopný na neplatenej verzií Google Search API vykonať desať testov denne.

Najväčším problémom bude samotná detekcia, resp. rozlíšenie dokumentov na ktorých sa správa nachádza, a na ktorých nie je. V kapitole 3.3.1 som sa zaoberal pozorovaním, na základe ktorého môžem povedať, že URL dokumentov obsahujúce správy obsahujú „*query string*“ alebo numerické hodnoty. Preto skontrolujem získané URL na výskyt týchto znakov a ak dané pravidlo nespĺňajú, automaticky ich označím ako dokumenty, na ktorých sa správa

[36] Viac o tomto API https://developers.google.com/custom-search/json-api/v1/overview a http://googledotnet.codeplex.com/

[37] https://console.developers.google.com

nenachádza. Týmto nesmierne jednoduchým pravidlom som schopný o dvadsiatich až štyridsiatich percentách vstupných dokumentov rozhodnúť, že sa nejedná o dokumenty so správou. Počet odfiltrovaných dokumentov je závislý od špecifikovaných kľúčových slov, tento algoritmus odstraňuje len dokumenty neobsahujúce správy. Je jasné, že pre niektoré kľúčové slová budú vo výsledku takmer samé správy(ako napr. „štátny rozpočet, politika, ...“) a tak tento postup odfiltruje len minimálne množstvo vstupných URL. Pre iné(ako napr. „motýľ, včielka Mája“) bude vo výsledku len minimálne množstvo spŕav. Tak tento algoritmus odfiltruje maximálny počet dokumentov. Samotná presnosť tohto pravidla je až zarážajúca počas pomerne rozsiahleho testovania som mal problém naraziť na dokument obsahujúci správu a nespĺňajúci toto pravidlo.

Ďalším redukčným pravidlom vstupných dokumentov je výskyt opisu v metadátach, t.j. tag s názvom „*description*“, niektoré weby špecifikujú tento tag s názvom „*og_description*“ alebo aj inými podobnými názvami, z toho dôvodu budeme vyhľadávať tag v metadátach s názvom obsahujúcim slovo „*description*“, čiže opis. Z pravidla platí, že všetky spravodajské webové servery majú tento tag definovaný. Na základe tohto pravidla odfiltrujeme ďalších desať až dvadsať percent z pôvodných vstupných dokumentov. Po aplikácií týchto redukčných pravidiel ostáva teda štyridsať až šesťdesiat percent dokumentov, ktoré môžu alebo nemusia obsahovať správu.

O určitej podmnožine týchto dokumentov sme schopní rozhodnúť bezproblémovo. Jedná sa o množinu HTML dokumentov implementujúcu schému „*http://schema.org/NewsArticle*[38]“. Práve táto schéma popisuje vzhľad a štruktúru webových správ. Detekcia schémy HTML dokumentu je triviálnym problémom, keďže schéma stránky sa vždy nachádza v atribúte pomenovanom „*itemproperty*“. Na detekciu schémy stačí získať hodnotu daného atribútu, pokiaľ sa tento atribút v HTML dokumente vyskytuje. Keby všetky HTML dokumenty implementovali túto schému a dodržiavali štandardy v nej špecifikované, bola by detekcia a extrakcia správ triviálnym problémom. Zo vstupnej množiny správ približne desať až osemnásť percent dokumentov implementovalo túto schému, čo predstavuje šestnásť až tridsať percent z nerozhodnutej množiny dokumentov. O týchto dokumentoch teda môžem prehlásiť, že sa jedná o dokumenty obsahujúce správu a pristúpiť k samotnej extrakcií. Extrakcia správy z dokumentu o ktorom rozhodneme, že obsahuje správu je pomerne bezproblémová a bude principiálne vychádzať z princípov popísaných v kap. 4.1. Keďže, ale budeme extrahovať správy z nesúrodej množiny HTML dokumentov, bude musieť dochádzať k analýze a detekcie tzv. „*pageletu*“ pre každý dokument zvlášť. Najčastejšie sa budeme snažiť extrahovať správy z najväčších svetových spravodajských webových portálov, niektoré z nich tela svojich správ nezaobaľujú blokom identifikovateľným atribútmi „*id*“ alebo „*class*“, prípadne nemajú telá svojich správ len v jednom jedinom bloku. Je tomu tak kvôli tomu, že sa tieto weby bránia častému „*data miningu*“ z nich a tak aspoň pribaľujú k telu správy určitý komerčný obsah alebo bránia detekcií správy. Pre viac informácií viď nasledujúcu kapitolu nižšie. Z toho dôvodu nebol detektor tela správy použitý v rámci vytvoreného poloautomatického wrappera najefektívnejším riešením, pre weby, ktoré pribaľovali komerčný obsah do tela správy, alebo nemali telo obalené blokom identifikovateľným atribútmi „*id*“ a „*class*“ dochádzalo k informačným presahom, čo nebolo až tak problémové, bohužiaľ správy, ktoré boli rozmiestnené do viacerých blokov stránky boli extrahované nesprávne s kritickými informačnými podsahmi. Detektor „*pageletu*“ z môjho poloautomatického wrappera mal približnú presnosť sedemdesiat percent. Rozhodol som sa preto implementovať nový detektor „*pageletu*“ pre tento automatický wrapper založený na porovnávaní ciest k elementom, resp. metódy opísanej v kap. 2.1.2.

[38] Kompletný popis schémy http://schema.org/NewsArticle

5.1.1 Detektor pageletu pre automatický wrapper

Za základ tohto algoritmu budem považovať fakt, že samotný text sa v HTML dokumentoch vyskytuje vždy vložený do elementov typu „<*p*>, <*strong*>, <*h1*>, ...“, všetky tieto elementy som schopný dohľadať bez nutnosti špecifikácie akéhokoľvek vstupu.
Ďalším faktom, s ktorým tento algoritmus počíta je fakt, že v rámci pageletu stránky bude takýchto elementov najviac, resp. že blok obsahujúci najviac elementov typu <*p*><*h1..h6*>,... je pagelet daného HTML dokumentu.

5.1.1.1 Popis algoritmu detekcie pageletu

Vyššie spomínaný algoritmus by sme mohli v niekoľkých krokoch popísať asi takto:
1. Nájdi všetky elementy typu <*p*>, <*h1...h6*>, <*strong*>
2. Pre nájdené elementy zapíš do poľa cestu k ich predkovi(keďže hľadáme im nadradený prvok)
3. Nájdi modus, resp. najčastejšiu hodnotu v poli, tento uzol budeme považovať za pagelet
4. Urč koľko percent z daných elementov je potomkom nájdeného uzlu
5. Kým nie je aspoň tridsať päť percent elementov potomkom daného uzlu opakuj:
   - Hľadaný uzol je rodič stávajúceho nájdeného uzlu

Spomínaných tridsať päť percent je hodnota, ktorá sa my pri testovaní najviac osvedčila. Nastavením tejto hodnoty ovplyvňujeme presnosť, nižšie hodnoty môžu viesť k podsahom, naopak tie vyššie k presahu. Uvedená hodnota tridsať päť percent je ideálnou a v približne deväťdesiatich piatich percentách prípadov fungovala dokonale.

Základné popisné vlastnosti 5.1 určenie pageletu HTML dokumentov pri automatickej extrakcií

```
const c ← 35 // this value change granularity of detection

input h ← HTML source code
begin
  Remove all script, remark tags and styles
  var textNodes ← select all p , h1..h6 nodes from h
  var[ ] pathArray ← new string [textNodes.count]
  for i ← 0 to textNodes.count do
    pathArray[ i ]← textNodes[ i ].parent.XPath
   end for
  var mostFrequentPath ← pathArray.modus
  while ((number of select all p,h1..h6 nodes in select node ( mostFrequentPath ) / textNodes .count) * 100 < c) do
    mostFrequentPath ← select node(mostFrequentPath).Parent
  end loop
return select node(mostFrequentPath)
```

5.1.2 Extrakcia zvyšných atribútov správy

Extrakcia atribútov titulok(*title*), popis(*description*) a dátum bude prebiehať rovnako ako tomu bolo vyššie(konkrétne v kapitole 4.2) opísaného poloautomatického wrappera. Najproblémovejším atribútom je bezpochyby dátum, extrakčné pravidlá v kapitole 4. ostatnú nezmenené. Keďže Google Custom Search API vracia predovšetkým anglo – americké dokumenty presnosť extrakcie dátumov bude omnoho vyššia ako tomu bolo u českých webových portálov.

5.1.3 Využitie kľúčových slov pre rozšírenie detekcie

Ako už bolo spomenuté v kap. 5.1 prostredníctvom detekcie schémy dokumentu som schopný nájsť niekoľko správ týkajúcich sa danej témy a tie následne extrahovať. Avšak po tejto extrakcií budeme mať stále veľké množstvo dokumentov o ktorých nie sme schopný rozhodnúť či obsahujú alebo neobsahujú správu. Približne tridsať až štyridsať percent vstupných dokumentov. Už spomínaná schéma „*http://schema.org/NewsArticle*“ predpisuje dokumentom, ktoré ju implementujú špecifikovať kľúčové slová danej správy(metadáta „*keywords*“). Všeobecne dokumenty obsahujúce správy v drvivej väčšine prípadov obsahujú špecifikované kľúčové slová. Hrubým odhadom približne osemdesiat percent dokumentov so správou definuje kľúčové slová danej správy. Vykonávajú to kvôli zabezpečeniu maximálnej miery tzv. SEO(z ang. Search Engine Optimalization[39]), kde definícia kľúčových slov je jedna z najdôležitejších úloh. Keďže na vstupe máme HTML dokumenty ktoré boli nájdené vyhľadávačom na základe určitého výrazu, môžeme predpokladať, že dané dokumenty budú tematicky príbuzné. Obzvlášť dané správy môžu popisovať tú istú udalosť len ju priniesli odlišné weby. Táto príbuznosť ma vedie k hypotéze, že aj kľúčové slová daných dokumentov budú do určitej miery podobné. Zrejme sa vo všetkých dokumentoch v kľúčových slovách bude vyskytovať daný zadaný výraz do vyhľadávača, z toho dôvodu zhodnosť jediného kľúčového slova nie je postačujúca. Taktiež vyhľadávač vracia stránky rôznych jazykov, platí, že väčšina je v angličtine ale vyskytujú sa tu aj iné jazyky. Preto si na analýzu na základe kľúčových slov zostavím vlastný algoritmus.

5.1.3.1 Popis algoritmu detekcie na základe analýzy kľúčových slov

Navrhnutý algoritmus musí pracovať s možnosťou, že vstupné stránky budú mať rozdielne jazyky. Z toho dôvodu musím do projektu implementovať detekciu jazyka. Detekcia jazyka textu je jedným z bežných problémov patriacich do problematiky NLP, mojimi prvými pokusmi bola detekcia na základe analýzy tzv. „*stop slov*“. Tento algoritmus podával pomerne presné výsledky, ale fungoval len pre angličtinu a slovenčinu, pri pokusoch o pridanie češtiny nastali problémy kvôli veľkej podobnosti stop slov jazyka českého a slovenského. Taktiež môj algoritmus bol veľmi pomalý, z toho dôvodu som sa rozhodol využiť niektorého z dostupných riešení problému detekcie jazyka. Keďže som už pracoval s API od google ponúkala sa možnosť využiť ich riešenia detekcie jazyka, lenže táto neplatená verzia detekcie jazyka by mala spoločné denné limity s už implementovaným vyhľadávaním. Preto som sa rozhodol zavrhnúť ich riešenie.

Po dlhšom pátraní som narazil na voľne dostupný detektor jazyka, skúmajúci výskyt špeciálnych znakov daného jazyka, vykonávajúci frekvenčnú analýzu n-gramov a analýzy výskytu najčastejších slov čiže „*stop slov*“. Tento detektor jazyka bol schopný detekcie až tridsaťpäť jazykov a umožňoval pridanie ďalších, čo bolo pre mňa dôležité, keďže neobsaho-

[39] Viac o SEO nájdete na http://en.wikipedia.org/wiki/Search_engine_optimization alebo na [26]

val slovenčinu. Navyše rýchlosť jeho detekcie bolo neporovnateľne lepšia ako moje jednoduché riešenie. Viac informácií o tomto detektore vrátane odkazu na stiahnutie nájdete na [18]. Daný projekt som pripojil do svojho, pridal slovenský jazyk a úspešne ho používal.

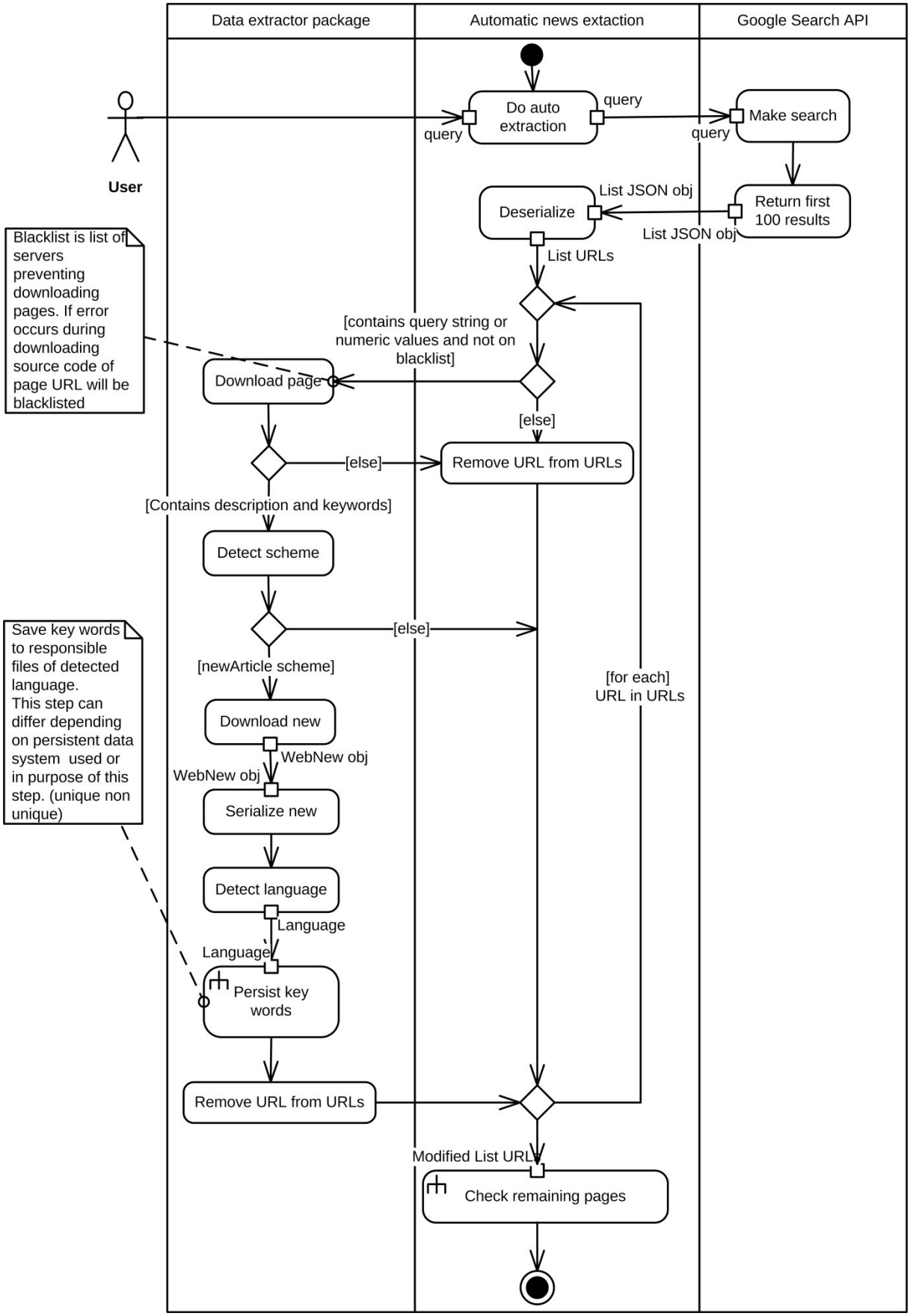
Algoritmus detekcie na základe kľúčových slov

Samotný algoritmus detekcie dokumentov obsahujúcich správy na základe kľúčových slov by sme mohli popísať nasledujúcimi krokmi:

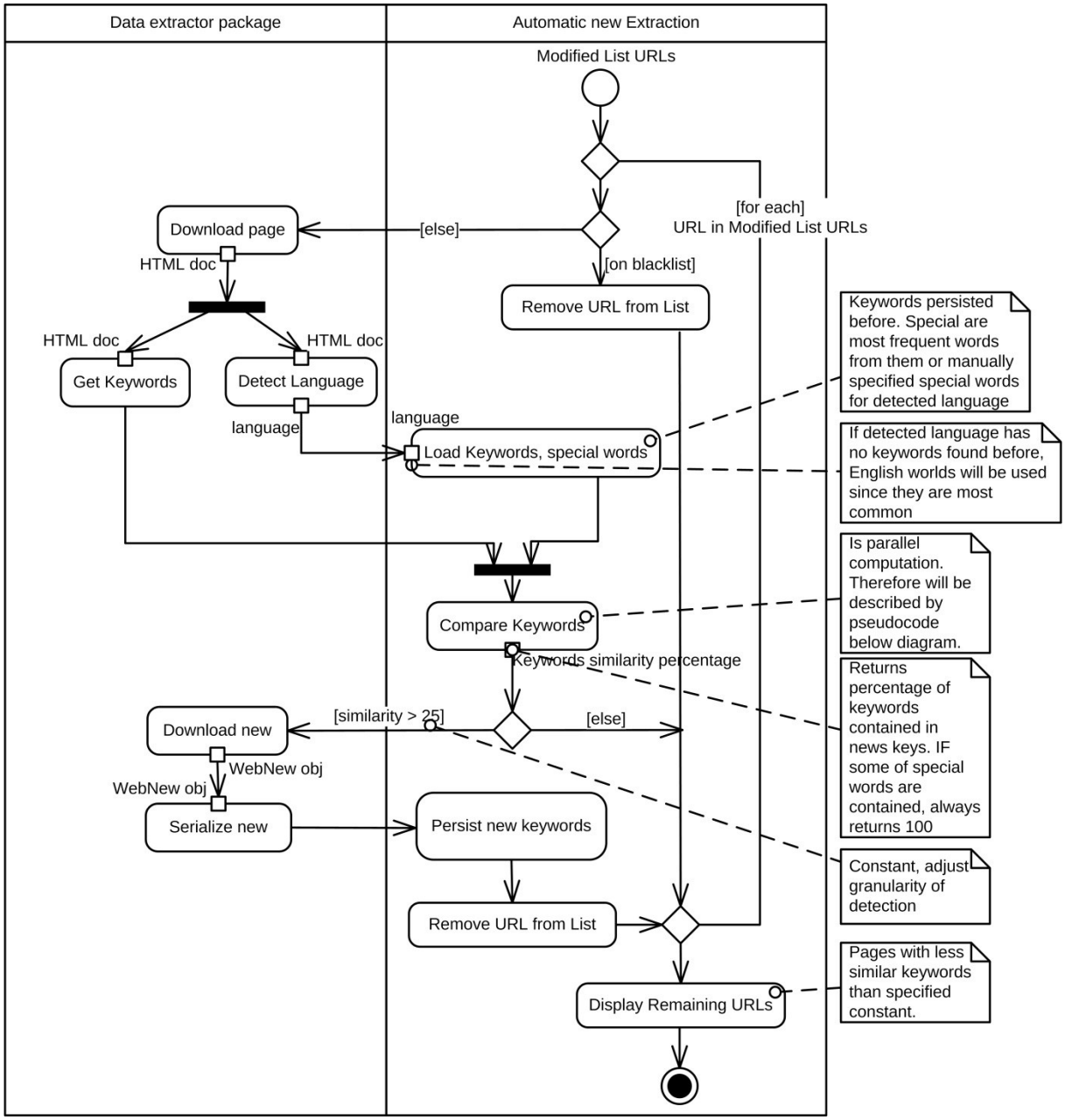
1. Pre každý dokument implementujúci schému „*http://schema.org/NewsArticle*“:
    1.1. Zisti jazyk dokumentu
    1.2. Ak neexistuje kolekcia kľúčových slov pre daný jazyk vytvor ju
    1.3. Paralelne pre každé slovo z kľúčových slov daného dokumentu kontroluj či sa vyskytuje v kolekcií
    1.4. Ak sa slovo v kolekcií nevyskytuje zapíš ho do nej
2. Vytvor premennú, ktorá bude zachytávať počet zhodných kľúčových slov, na začiatku bude jej hodnota nulová
3. Pre každý dokument, o ktorom sa ešte nerozhodlo či obsahuje alebo neobsahuje správu:
    3.1. Zisti jazyk dokumentu
    3.2. Ak neexistuje kolekcia slov pre daný jazyk použi anglické slová(najčastejšie)
    3.3. Paralelne pre každé kľúčové slovo dokumentu hľadaj jeho výskyt v danej kolekcii
    3.4. Ak nájdeš výskyt navýš hodnotu premennej počet zhodných slov o jedna
    3.5. Ak nájdeš výskyt špeciálneho slova pre daný jazyk(napr. „*new*“ pre angličtinu), navýš hodnotu počet zhodných slov na maximálnu hodnotu t.j. počet kľúčových slov a ukonči prehľadávanie(„*break*“)
4. Vypočítaj koľko percent kľúčových slov sa vyskytovalo v kolekcií($\frac{zhodných}{celkovo} * 100$)
5. Ak bolo zhodných viac ako dvadsať percent kľúčových slov považuj dokument za dokument obsahujúci správu a správu extrahuj
6. Zapíš do kolekcie kľúčové slová novo nájdenej správy, ktoré sa tam nevyskytujú

Pre takýto nízky počet vstupných dokumentov(100) by bolo možné vynechať detekciu jazyka, ale šlo my o principiálny popis algoritmu automatického wrappera a ten by sa bez detekcie jazyka nezaobišiel. Takto špecifikovaný algoritmus dokáže nájsť a rozhodnúť o ďalších desať až v najideálnejších prípadoch dvadsať percentách pôvodných dokumentov. Po vykonaní tohto algoritmu nám teda ostane dvadsať až tridsať percent správ o ktorých nie sme schopný rozhodnúť či obsahujú alebo neobsahujú správu.

Obrázok 5.1 Diagram aktivít zachytávajúci počiatočnú fázu automatickej detekcie správ(bod 1. v popise algoritmu vyššie)



Obrázok 5.2 Pokračovanie diagramu aktivít zachytávajúceho automatickú extrakciu na základe kľúčových slov (body 3 až 6 z popisu algoritmu vyššie)



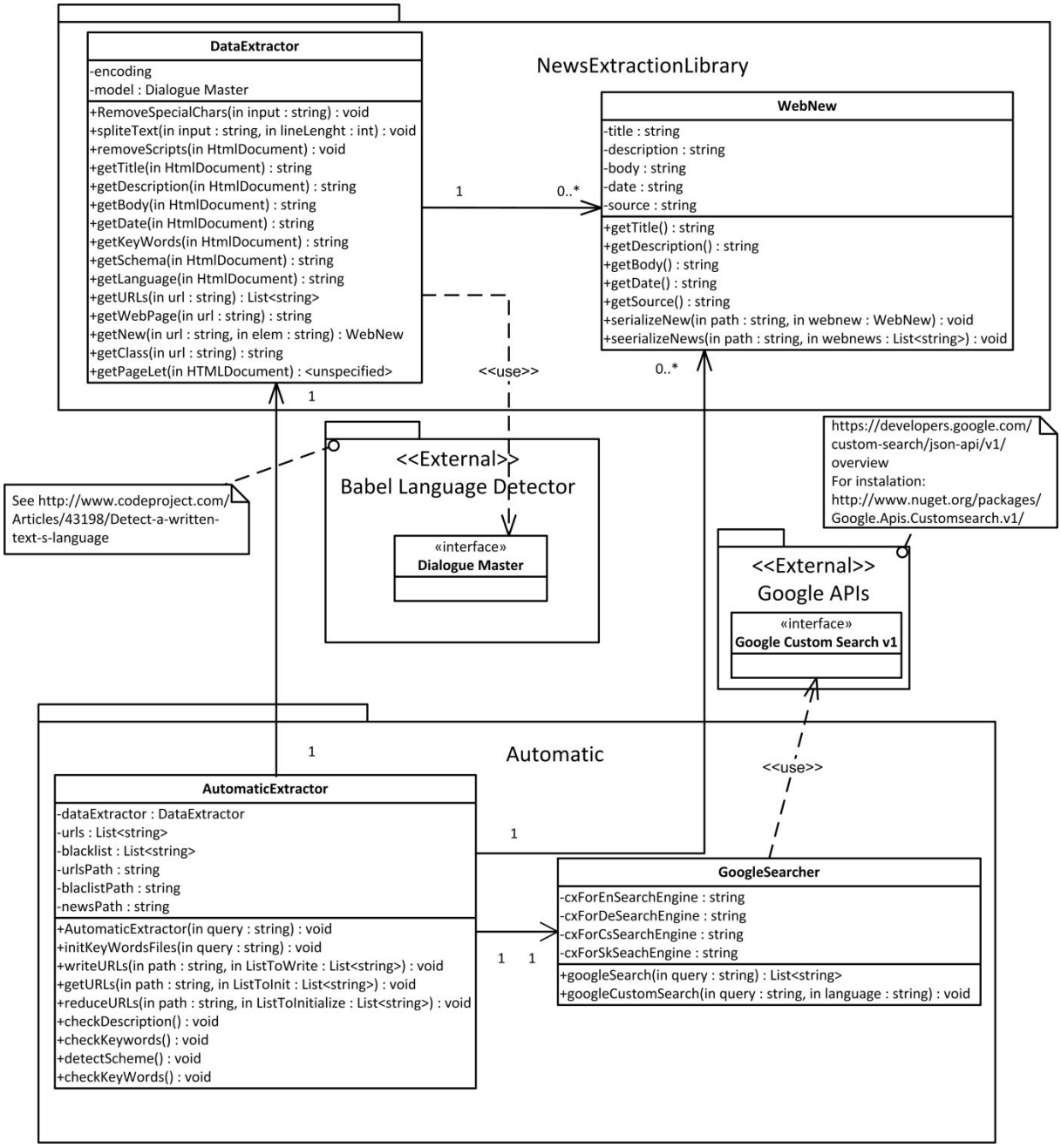
Preudokód 5.2 pre aktivitu "Compare Keywords" z diagramu aktivít vyššie

```
Function CompareKeyWords(keywords_page , keywords_lang , specialWords)
//keywords_page : set of all keywords from page to detect
//keywords_lang : set of all keywords from pages containing web news in specific language
//specialWords  : set of special words for specified language, they can be manually
//written or obtained by frequency analysis
begin
  var matched ← 0
  parallel for each string keyword in keywords_page , currentWord => do
    for each string word in specialWords do
      if currentWord.contains(word) then
        matched ← keywords_page .count
        break
      end if
    end for
    if keywords_lang .contains( keyword ) ∧ ¬ keywords_lang is string.Empty then
      increment matched
    end if
  end parallel loop
return matched / keywords_page .count * 100 // similarity in %
```

Je jasné, že pri získaní väčšej množiny kľúčových slov by sme boli schopní rozhodnúť o ďalších dokumentoch vstupnej množiny. Vykonával som experimenty, kde som plnil kolekcie kľúčovými slovami získanými poloautomatickou analýzou pevne zadaných spravodajských serverov pre niektoré jazyky a to tak, že ak sa vyskytovalo kľúčové vstupné slovo v kľúčových slovách dokumentu na daných portáloch tak boli kľúčové slová danej správy pridané do kolekcie. Prehľadávanie týchto pevne daných webov bolo veľmi zdĺhavé a pri kompletnej analýze daných webov sme získali len niekoľko kľúčových slov, čo zlepšilo automatickú extrakciu o dve až päť percent, ale časové nároky na beh aplikácie sa zmenili z rádovo niekoľko minút až na niekoľko hodín. Pritom som tieto weby analyzoval paralelne.
Z toho dôvodu sa uspokojím s výsledkom detekcie na základe schémy a na základe slov získaných zo správ implementujúcich danú schému. Taktiež tieto slová nemusím zahadzovať ale po skončení vyhľadávania si ich zapísať pre ďalšie vyhľadávanie, keďže sa jednalo o kľúčové slová správ.

Ďalšou možnosťou vylepšenia boli špeciálne slová, ktorých výskyt indikuje fakt, že dokument obsahuje správu. Najčastejšie je to slovo „správa“ v rôznych jazykoch a synonymá slova „správa“ v rôznych jazykoch. Príklady takýchto slov pre angličtinu by boli napríklad „*new*“, „*news*“, pre slovenčinu „správa“, „správy“, „aktualita“, „aktuality“, pre češtinu „zpráva“, „zprávy“. V prípade, že nájdem dané slovo v kľúčových slovách dokumentu môžem ho považovať za dokument obsahujúci správu. Táto možnosť je popísaná ako bod 3.5 v popise algoritmu vyššie. Tieto slová sa dajú identifikovať frekvenčnou analýzou získaných kľúčových slov z dokumentov, o ktorých som rozhodol, že obsahujú správu. Najčastejšie slová vyskytujúce sa v množine kľúčových slov sú ony spomínané špeciálne kľúčové slová pre správy.

Obrázok 5.3 Triedny diagram zachytávajúci statickú štruktúru navrhnutého automatického wrappera a rozdelenie tried do jednotlivých balíčkov



5.2 Testovanie navrhnutého automatického wrappera

Keďže je vstupom tohto wrappera vždy sto správ získaných vyhľadávačom google je testovanie efektivity nepotrebné a aplikácia vždy končí v reálnom čase rádovo do niekoľkých minút. Hlavným kritériom testovania bude presnosť detekcie správ. Pri detekcií môžu nastať štyri situácie a to:

1. správa sa vyskytuje a bola nájdená
2. správa sa nevyskytuje a nebola nájdená
3. správa sa vyskytuje a nebola nájdená
4. správa sa nevyskytuje a bola nájdená

Tieto štyri možnosti môžeme popísať pravdepodobnostnou tabuľkou. Testovanie prebehlo pre niekoľko zadaných vstupných výrazov pre vyhľadávač. Keďže pre všetky nájdené dokumenty bolo nutné manuálne prejsť a rozhodnúť o tom či obsahujú alebo neobsahujú správu som toto testovanie vyhodnocoval len pre niekoľko výrazov. Ako som už spomínal v predchádzajúcej kapitole, presnosť bude závisieť od počtu kľúčových slov, ktoré sme schopní získať. Keďže kľúčové slová nepremazávam ale pre jednotlivé jazyky ich ukladám do textových súborov, predpokladám, že čím viac vyhľadávaní s podobnou tematikou vykonám, tým presnejšiu analýzu získam. Taktiež sa, ale o niečo predĺži doba behu programu, keďže porovnávam s o to väčšou kolekciou kľúčových slov.

Nasledujúca tabuľka(t.j. Tabuľka 5.1) zachytáva úspešnosť automatického wrappera pre výraz „Putin“. Jedná sa o prvé vyhľadávanie, kde nie sú špecifikované žiadne kľúčové slová.
Pre najviac testované jazyky a to slovenčinu, češtinu a angličtinu som špecifikoval špeciálne slová ako statické premenné a to „*new, news*“ pre angličtinu, „*správa*, *správy*, *aktualita*, *aktuality*“ pre slovenčinu a „*zpráva*, *zprávy*, *aktualita, aktuality*“ pre češtinu. V prípade, že sa narazí na dané kľúčové slovo v kľúčových slovách dokumentu s odpovedajúcim jazykom, je dokument považovaný za dokument nesúci správu.

Dané slová by sa dali nájsť aj automaticky frekvenčnou analýzou kľúčových slov dokumentov implementujúcich schému „*http://schema.org/NewsArticle*“, keďže ale tieto slová ukladám ako unikátne slová, aby som zrýchlil vyhľadávanie v nich, pre účely testovania si vystačím so statickými premennými pre pár špecifikovaných jazykov.

Tabuľka 5.1 Pravdepodobnostná tabuľka zachytávajúcu úspešnosť automatického wrappera bez akýchkoľvek kľúčových slov na začiatku

	Správa sa vyskytuje	**Správa sa nevyskytuje**
Správa bola nájdená	26	2
Správa nebola nájdená	37	35

Z tabuľky(Tabuľka 5.1) vyplýva, že, šesťdesiat jedna percent dokumentov bolo určených správne. Dva dokumenty boli identifikované mylne a extrahovali sa tak iné dáta ako správy, konkrétne životopisy, ktoré štrukturálne správy najviac pripomínajú.
Zaujímavá je množina tridsiatich siedmich dokumentov, ktoré správu obsahujú, ale neboli identifikované. Väčšina týchto dokumentov má špecifikované kľúčové slová, len sme neboli schopný získať dostatok kľúčových slov pre správy aby sme tieto dokumenty správne identifikovali. Z toho dôvodu vykonám niekoľko automatických extrakcií na podobné témy, čím sa mi rozrastie množina kľúčových slov.

Tabuľka 5.2 Pravdepodobnostná tabuľka zachytávajúcu úspešnosť automatického wrappera s kľúčovými slovami dvoch podobných extrakcií na začiatku

	Správa sa vyskytuje	**Správa sa nevyskytuje**
Správa bola nájdená	28	3
Správa nebola nájdená	34	35

Vykonal som dva ďalšie vyhľadávania, s výrazmi tematicky súvislými s prvou extrakciou, keďže výrazy tematicky súviseli, našlo sa len pár nových kľúčových slov. Aj týchto pár kľúčových slov stačilo na to, aby sa čiastočne zlepšila úspešnosť pre pôvodný výraz extrakcie. Úspešnosť vzrástla z šesťdesiat jedna na šesťdesiat tri. Takýmto spôsobom by sa s každým novým vyhľadávaním pre daný jazyk mohla úspešnosť zlepšovať, až po strop a to budú sprá-

vy, ktoré nemajú špecifikované kľúčové slová, alebo ich kľúčové slová sa nevzťahujú na danú správu, ale na web na ktorom sa nachádzajú(tento prípad je pomerne častým javom).

Aby som simuloval zisk väčšieho množstva adekvátnych kľúčových slov bez mnohonásobného testovania, vyhľadám si prostredníctvom „*Google Search API*“ rovnakým kľúčovým slovom sto správ implementujúcich schému „*http://schema.org/NewsArticle*“.
Tak výrazne navýšim počet kľúčových slov, ktorými si pomôžem pri detekcií ďalších správ.

Tabuľka 5.3 Pravdepodobnostná tabuľka zachytávajúcu úspešnosť automatického wrappera s kľúčovými slovami dvoch podobných extrakcií a kľúčovými slovami stovky správ implementujúcich schému "*http://schema.org/NewsArticle*" na začiatku

	Správa sa vyskytuje	**Správa sa nevyskytuje**
Správa bola nájdená	31	3
Správa nebola nájdená	31	35

Po získaní kľúčových slov zo stovky správ sa úspešnosť zvýšila na šesťdesiat šesť percent. Zvyšné správy nie sú stiahnuté, pretože buď nešpecifikujú všetky požadované elementy(nadpis, opis, telo, dátum zdroj) alebo nešpecifikujú kľúčové slová, prípadne ich kľúčové slová sa nevzťahujú k danej správe.

5.3 Navrhnutý automatický wrapper – zhrnutie

Mnou navrhnutý wrapper som sa snažil koncipovať ako jednoduchú užívateľskú, ľahko otestovateľnú aplikáciu. Šlo mi hlavne o teoretický popis metódy jednoduchej a efektívnej automatickej extrakcie správ. Avšak automatické wrappery sa zriedka kedy používajú ako užívateľské aplikácie. Najčastejšie bežia nepretržite na serveroch a využívajú pokročilé databázové systémy(DBMS[40]), ktoré sú schopné samostatne pracovať so získanými dátami a ďalej ich analyzovať a využívať. V kapitole 5.1 popísaná a v 5.2 testovaná metóda by mohla slúžiť ako základ pre takýto automatický wrapper. Aplikačná vrstva takéhoto wrappera by pozostávala z jedného nepretržite bežiaceho crawlera, vyhľadávajúceho stránky implementujúce schému „*http://schema.org/NewsArticle*“. Pre všetky nájdené správy by dochádzalo k detekcií jazyka a extrakcií kľúčových slov, tie by boli uložené do DBMS do tabuľky pre daný jazyk. DBMS by vykonával frekvenčnú analýzu z daných slov a na špeciálne miesto ukladal najfrekventovanejšie slová.

Druhý crawler by spracovával užívateľské požiadavky na analýzu určitých dokumentov, t.j. by prehľadával dané dokumenty a analyzoval ich kľúčové slová na výskyt získaných „všeobecných“ slov pre správy pre daný jazyk a tak by identifikoval správy. DBMS by mohlo byť využité aj na vytváranie kľúčových slov dokumentov, ktoré ich neobsahujú prostredníctvom frekvenčnej analýzy slov textu v „*pagelete*“, ktorého detekciu som v tejto práci už niekoľko krát opísal. Takýto wrapper by mal omnoho lepšie efektivitu, ako uvádzajú testy v kapitole 5.2. Keďže som nebol schopný prevádzkovania nepretržitého crawleru, ktorý by navyše bol schopný nachádzať slušné množstvo správ implementujúcich spomínanú schému „*http://schema.org/NewsArticle*“, rozhodol som sa že jednoduchá užívateľská aplikácia, ktorú som popísal v kap. 5.1 bude postačujúca na dokumentovanie metódy automatickej extrakcie.

Celková úspešnosť automatických wrapperov je v porovnaní s poloautomatickým prístupom veľmi nízka, to čo ale strácajú na presnosti, doháňajú vhodnosťou použitia na serveroch a nepretržitou prevádzkou. Navyše presnosť automatických wrapperov sa vo všeobecnosti zvyšuje so zvyšujúcim sa počtom dát, ktoré môžu vyhodnocovať.

[40] http://www.techterms.com/definition/dbms

6. Problémy pri vypracovaní práce

Počas vypracovania tejto práce som narazil na niekoľko problémov. Jedným z hlavných, ktorý som už v práci niekoľko krát spomínal je rozsiahle nedodržiavanie štandardov a najlepších zaužívaných praktík(tzv. „*best practices*“) pre tvorbu webu. Na mnohých webových dokumentoch nájdeme takzvané kríženie tagov, zastarané hodnoty niektorých atribútov, ktoré sa už nepoužívajú alebo len je štruktúra HTML dokumentu nesmierne neprehľadná, pokrížená alebo komplikovaná, čo je najčastejšie problémom pri použití tzv. WYSIWIG editorov. Tento fakt veľmi komplikuje akékoľvek extrakčné metódy. Niektoré stránky sú vytvorené technológiou „*flash*“ od spoločnosti Adobe. Tieto dokumenty sú reprezentované ako binárne objekty(súbory s príponou *.swf*), ktorých prezentáciu zabezpečuje zásuvný modul do prehliadača, bez ktorého sú tieto objekty neprístupné.

Ďalšou problematickou skupinou sú stránky s dynamickým obsahom. Zobrazenie obsahu týchto stránok je realizované prostredníctvom rôznych serverových technológií a je závislé od určitej podmienky, napr. prihlásenia do systému, overenia tzv. „*captcha*“, či sa jedná o užívateľa a iné. Existuje niekoľko spravodajských serverov vo svete, na ktorých pre zobrazenie musí byt užívateľ prihlásený. Tieto stránky často implementujú napr. spomínanú schému „*http://schema.org/NewsArticle*“, alebo sú prostredníctvom kľúčových slov detekované ako dokument obsahujúci správu, ale samotné telo správy je generované dynamicky serverom po overení prihlásenia užívateľa napr. hodnotou v „*cookies*“. Pokiaľ narazíme na takúto dynamickú správu dôjde k jej detekcií a následne nesprávnej extrakcií, kde bude namiesto tela extrahovaný zobrazený text o prihlásení a často mnoho komerčného obsahu, keďže v kap. 5.1.1 popísaný detektor hľadá rodičovský element textov a v tomto dokumente nie je text tela správy nájde rodičovský element časti stránky s najväčším obsahom textu mimo tagov.

6.1 Obrana proti automatizovanému „*data miningu*“

Tvorba webových správ je proces, na ktorý majitelia spravodajských portálov vynakladajú nemalé prostriedky či už finančné alebo časové. Plagiátorstvo[41] je závažným problémom súčasnej doby. Prostredníctvom automatizovaných prostriedkov a metód(napr. takých, aké som popisoval v tejto práci) nie je problém získavať a ďalej pre svoje účely využívať informácie a dáta, ktorých nie sme autorom. Takto môže vznikať škoda vlastníkom a jedná sa o porušenie tzv. autorského zákona – zákon č. 618/2003 Z.z.[42] a teda o trestný čin. Z toho dôvodu sa niektoré weby bránia automatickej extrakcií dát a to aj za cenu zníženia miery SEO daného webu. Vo všeobecnosti platí, že veľké weby, ktoré sú svetoznáme a sú v povedomí ľudí, majú maximálnu snahu brániť sa automatickej extrakcií a vo všeobecnosti „*data miningu*“ z nich. Naopak malé weby obranu vôbec neriešia, dbajú na maximálnu možnú mieru SEO a sú radi za akýkoľvek „*traffic*“ v rámci ich webu. Fakt, že odkazy na ich správy sa objavia napr. na *„https://news.google.com/“* alebo iných podobných službách berú skôr ako propagáciu.

6.1.1 Banovanie crawlerov

Crawlovanie a extrakcia dát sú operácie vyťažujúce serverové zdroje, v drvivej väčšine prípadov vykonávané na serveroch, ktoré užívateľ daného crawlera / wrappera nevlastní. Vo väčšine prípadov sú crawlery vítané, pretože generujú premávku(„*traffic*“) na danom servery a ich činnosť obvykle smeruje k určitej propagácií daného servera a informácií na ňom. Avšak nie

[41] Viac na: http://en.wikipedia.org/wiki/Plagiarism
[42] Plné znenie autorského zákona platné k dátumu 27.04.2014 nájdete na: http://www.juls.savba.sk/dokumenty/autorsky_zakon.html

vždy je tomu tak. Vo veľkej miere závisí na konfigurácií daného crawlera. Na serveroch je vedený tzv. „*log*“ kde sú zapisované všetky prístupy na daný server za určitý čas. Typicky tie záznamy obsahujú IP adresu pristupujúceho, čas, URL dokumentu, ktorý bol vyžiadaný. Na základe analýzy tohto „*logu*“ je možné identifikovať činnosť crawlera(zhodnosť IP a krátke časové intervaly prístupu). Crawlery zaťažujúce daný server, alebo vykonávajúce neprívetivú činnosť sú identifikované a zabanované. Vlastníci serverov taktiež spolu komunikujú a spoločne si vytvárajú „*banlisty*“. Existuje niekoľko jednoduchých pravidiel, ktoré treba dodržiavať pri tvorbe a používaní crawlerov, tak aby nedochádzalo k ich banovaniu.

6.1.1.1 Pravidlá pre tvorbu „*slušných*“[43] crawlerov:

1. Nikdy nevykonávaj analýzu kontinuálne, ale v časových intervaloch(napr. raz za deň, týždeň)
2. Aktivuj crawler v hodinách najmenšieho zaťaženia serveru
3. Ber si len to čo potrebuješ
4. Pozor na paralelizmus. Paralelizmus zrýchľuje crawlovanie, ale nesmierne zaťažuje server(navyše môže byť považovaný za DoS útok). Preto obmedziť počet vlákien na rozumnú mieru.
5. Pozor na mnohonásobné načítania dokumentov a dát. Stiahnuť dokument a analyzovať jeho lokálnu kópiu.

6.1.2 Ďalšie spôsoby obrany a problémy extrakcie

Niektoré servery namiesto banovania konkrétnych wrapperov udržujú aktívny prenos informácií a tak znemožňujú efektívnu prácu crawleru. Medzi takéto weby patrí napr. *http://en.wikipedia.org* a iné. Pri pokusoch o stiahnutie celej stránky z daného webu do môjho počítača, kde by sa vykonávala ďalšia analýza nikdy nedošlo k správnemu ukončeniu prenosu dát. Tak wrapper čakal na „*TimeoutException*[44]“, čo spomaľovalo celkový beh wrappera.
Podobný problém spôsobujú aj odkazy na neprístupné stránky, neexistujúce stránky a podobne. Keďže som vykonával mnoho testov, tieto zdržania pre mňa boli výrazný problém a tak som do svojho projektu pridal „*blacklist*“ takýchto serverov a pokiaľ sa novo nájdený dokument nachádzal na jednom zo serverov na „*blackliste*“, nepokúšal som sa o jeho extrakciu. Čo taktiež mierne negatívnym spôsobom ovplyvňovalo testy v kap. 5.2 .
Existuje aj veľké množstvo softwaru a softwarových prvkov, ktoré znemožňujú činnosť automatizovaných crawlerov. Za všetky spomeniem aspoň „*mod-anticralwl*[45]“, t.j. mód najpoužívanejšieho serveru „*Apache*“, ktorý znemožňuje prácu crawlerov v závislosti na svojej internej konfigurácií. Ďalší je napr. „APC Anti Crawler[46]“, čo je trieda v jazyku PHP umožňujúca nastaviť limity http požiadaviek na jednu IP v určitom čase.
Okrem týchto spomínaných existujú aj mnohé iné.

S rozvojom výkonnosti dostupných PC zostáv, rastúcim výkonom sieťových prvkov a rozširovaním optických sietí je crawlovanie a jeho obmedzenie na mieru, ktorá nespôsobí záťaž serveru problematika, ktorú by mal každý vlastník alebo správca serveru riešiť. Inak môže nadmerne crawlovanie pôsobiť ako krátkodobý DoS útok, aj keď rôzny užívatelia crawlerov nemali v úmysle danému webovému serveru spôsobiť žiadnu škodu.

---

[43] Viac o informácií o tejto problematike sa nachádza na [19].
[44] http://msdn.microsoft.com/en-us/library/system.timeoutexception.aspx
[45] https://code.google.com/p/mod-anticrawl/
[46] http://www.ohloh.net/p/apcanticrawler

7. Záver

Internet je skladiskom nesmierneho množstva informácií, ktoré už v dnešnej dobe nie je možné manuálnymi postupmi analyzovať. Z toho dôvodu je vývoj aplikácií vykonávajúcich extrakciu dát automatickou alebo poloautomatickou cestou rozvíjajúcim sa a ekonomicky potencionálnym odvetvím informatiky.

V tejto diplomovej práci som sa zaoberal problematikou extrakcie správ z prostredia Internetu. Vykonal som rozsiahly prieskum stávajúcich prístupov k danej problematike, ktorého výsledky som prezentoval v 2. kapitole tejto práce. Následne som sa venoval empirickému skúmaniu najznámejších českých, slovenských a aj svetových webových spravodajských portálov. Výsledky tohto skúmania sú sumarizované v 3. kapitole tejto práce. Poznatky získané pri tvorbe druhej a tretej kapitoly tejto práce som využil pri implementácií vlastnej knižnice napísanej v jazyku C# využívajúcej balíčka *Html Agility Pack*[47] v aktuálnej verzií 1.6. Táto knižnica obsahuje metódy umožňujúce činnosti ako napr. detekcia časti stránky nesúcej obsah(„*pagelet*“), detekcia schémy, jazyka. Ďalej extrakcia opisu, kľúčových slov, tela správy, prehľadávania stránky na výskyt dátumov a mnohé iné. Jej použitie je možné bez akejkoľvek znalosti balíčka „*Html Agility Pack*“ alebo „*LINQ*[48]“ vo všeobecnosti.

V súčasnej dobe sa najčastejšie k extrakcií dát z Internetu využíva poloautomatický alebo plne automatický prístup.V kapitolách 4. a 5. som sa týmto prístupom venoval podrobne a navrhol som vlastné riešenia problému extrakcie správ z Internetu poloautomatickou a plne automatickou cestou. Obe riešenia boli postavené na spomínanej knižnici a využívali jej funkcionalitu. Pre poloautomatický wrapper som taktiež vytvoril jednoduché a užívateľsky príveti vé GUI prostredníctvom technológie „*Windows Froms*[49]“, tak aby daný wrapper mohol konkurovať dostupným wrapperom použiteľným na riešenie problematiky extrakcie správ z Internetu . Pri extrakcií dát z Internetu dochádza k využívaniu a zaťažovaniu serverových zdrojov, čo v prípade zlých wrapperov vedie k obmedzeniu služby postihnutých serverov. Z toho dôvodu sa kladú nároky na návrh a činnosť crawlerov, tak aby nepoškodzovali vlastníka zdrojov extrakcie. Touto problematikou som sa podrobnejšie zaoberal v 6. tej kapitole tejto práce. Mnou navrhnuté wrappery som sa snažil optimalizovať tak, aby spĺňali tzv. „*best practices*“ pre činnosť crawlerov a zároveň boli schopné podávať výsledky v reálnom čase.

Za možnosť ďalšieho rozvoja a pokračovania tejto práce považujem vytvorenie komplexného plne automatického serverového riešenia využívajúce DBMS na problém extrakcie a následného spracovania správ, tak ako bolo teoreticky popísané v kap. 5.3 . Získané správy a kľúčové slová by mohli byť taktiež na databázovej úrovni katalogizované, čo by umožnilo väčšiu prehľadnosť celého riešenia. Ďalšou možnosťou by bola optimalizácia paralelnej extrakcie a detekcie správ poloautomatického wrappera opísaného v 4.2, tak aby bol vytvorený wrapper jedným z najefektívnejších voľne dostupných riešení určených na extrakciu správ z určitého spravodajského webu.

[47] http://htmlagilitypack.codeplex.com/
[48] http://msdn.microsoft.com/en-us/library/bb397926.aspx
[49] http://msdn.microsoft.com/en-us/library/dd30h2yb%28v=vs.110%29.aspx

8. Zdroje a literatúra

1. Yahoo Query Language: What is YQL? [Online] 10. 02 2014. http://developer.yahoo.com/yql/.

2. **Li, Yunyao, a iní, a iní.** Regular Expression Learning for Information Extraction. *pages 21–31.* [Online] October 2008. [Dátum: 05. 02 2014.] http://www.qbic.almaden.ibm.com/cs/projects/avatar/pubs/emnlp2008.pdf.

3. **Selman, Bart a GOMES, Carla P.** Hill-climbing Search New York. [Online] [Dátum: 26. 02 2014.] http://www.cs.cornell.edu/gomes/selman-gomes-encycl-hillclimbing.pdf.

4. **Laclavik, Michal, a iní, a iní.** Ontology based Text Annotation – OnTeA. [Online] [Dátum: 05. 02 2014.] http://ups.savba.sk/~misos/publications/P626_ios_press.pdf.

5. **Handschuh, S. a Staab, S.** *Authoring and annotation of web pages in cream.* NY : ACM Press, 2002. s. pages 462-473. ISBN 1-58113-449-5.

6. **Domingue, J. a Dzbor, M.** *Magpie: supporting browsing and navigation on the semantic web.* NY : ACM Press, 2004. s. 191-197. ISBN 1-58113-815-6.

7. **Liu, a iní, a iní.** Mining Data Records in Web Pages. [Online] 2003. [Dátum: 06. 02 2014.] http://www.cs.uic.edu/~liub/publications/kdd2003-dataRecord.pdf.

8. **Miao, Gengxin, a iní, a iní.** Extracting Data Records from the Web Using Tag Path Clustering. [Online] 2009, Madrid. [Dátum: 06. 02 2014.] http://www2009.eprints.org/99/1/p981.pdf.

9. **Shi, Jianbo a Malik, Jitendra.** Normalized Cuts and Image Segmentation. [Online] [Dátum: 07. 02 2014.] http://www.cs.berkeley.edu/~malik/papers/SM-ncut.pdf.

10. **Weninger, Tim a Hsu, William H.** Text Extraction from the Web via Text-to-Tag Ratio. [Online] [Dátum: 07. 02 2014.] http://www3.nd.edu/~tweninge/pubs/WH_TIR08.pdf.

11. **Hochbaum, D. S. a Shmoys, D. B.** A Best Possible Heuristic for the KCenter Problem. *Mathematics of Operational Research, vol. 10.* [Online] pages 180-184, May 1985. DOI:10.1287/moor.10.2.180.

12. **Dempster, A., Laird, N. a Rubin, D.** Maximum likelihood from incomplete data via the EM algorithm. [Online] 1977. http://web.mit.edu/6.435/www/Dempster77.pdf. doi=10.1.1.133.4884.

13. Analysis of Multivariate Observations. *Probability, Proc. of 5th Berkeley Symposium on Mathematical Statistics and.* [Online] 1967. http://www.google.sk/books?hl=sk&lr=&id=IC4Ku_7dBFUC&oi=fnd&pg=PA281&dq=Analysis+of+Multivariate+Observations&ots=nM-kC_G8rM&sig=g3HbBCIdzqYyEbn-NmfXXp1ZJ-o&redir_esc=y#v=onepage&q=Analysis%20of%20Multivariate%20Observations&f=false. DOI:10.1080/00401706.1978.10489650.

14. **Hirschberg, D. S.** Algorithms for the Longest Common Subsequence Problem. *Journal of ACM, vol. 24, no. 4.* [Online] 1977. [Dátum: 07. 02 2014.] DOI:10.1145/322033.322044.

15. **Cai, Deng, a iní, a iní.** Extracting Content Structure for Web Pages based on Visual Representation. [Online] 2003, Berlin. Dostupné: http://www.dbs.ifi.lmu.de/~spyu/paper/VIPS-APWeb.pdf.

16. **Kudelka, Milos, a iní, a iní.** Semantic Analysis of Web Pages Using Web Patterns. [Online] 2006. [Dátum: 20. 04 2014.] http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4061388. DOI: 10.1109/WI.2006.153.

17. **Hollink, Laura, Mika, Peter a Blanco, Roi.** [Online] 2013, Rio de Janeiro. [Dátum: 20. 04 2014.] http://www2013.wwwconference.org/proceedings/p561.pdf.

18. **Zeumer, Carsten.** [Online] 21. 10 2009. [Dátum: 21. 04 2014.] http://www.codeproject.com/Articles/43198/Detect-a-written-text-s-language.

19. Writing a Web Crawler: Politeness. [Online] 20. 12 2011. [Dátum: 28. 04 2014.] http://blog.mischel.com/2011/12/20/writing-a-web-crawler-politeness/.

20. **Goyvaerts, Jan.** Regular Expressions: The Complete Tutorial. [Online] 2007. [Dátum: 04. 02 2014.] http://www.princeton.edu/~mlovett/reference/Regular-Expressions.pdf.

21. **Cohen, William a Mccallum, Andrew.** Information Extraction from the World Wide Web. [Online] 2002. [Dátum: 04. 02 2014.] http://isi.edu/info-agents/courses/csci548/Slides/McCallum_IE_tutorial.pdf.

22. Machine learning. *Wikipedia: the free encyclopedia.* [Online] [Dátum: 16. 02 2014.] http://en.wikipedia.org/wiki/Machine_learning.

23. **Rada, Peter.** OnTeA - Ontológia založená na textovej anotácií. [Online] [Dátum: 05. 02 2014.] http://www2.fiit.stuba.sk/~kapustik/ZS/Clanky0809/rada/index.html.

24. LevensteinDistance. *Wikipedia: the free encyclopedia.* [Online] [Dátum: 06. 02 2014.] http://en.wikipedia.org/wiki/Levenshtein_distance.

25. *HTML/Elements/time.* [Online] [Dátum: 16. 04 2014.] http://www.w3.org/wiki/HTML/Elements/time.

26. Search Engine Optimization: Starter Guide. [Online] 2014. [Dátum: 21. 04 2014.] http://static.googleusercontent.com/media/www.google.com/cs//webmasters/docs/search-engine-optimization-starter-guide.pdf.

9. Zoznam príloh

CD „Diplomová práce“	Obsah adresára
ExtrakciaSpravZWebu.zip	komprimovaná „solution“ kompletnej testovacej aplikácie
NewsDownloaded.zip	komprimované XML serializované správy získané pri testoch

[i] Štandardný problém v jednom dokumente v tesnej blízkosti niekoľko mien, niekoľko mailov, niekoľko adries. Algoritmus len obtiažne a často nepresne určuje, ktorá entita patrí ku ktorej.

[ii] http://sourceforge.net/projects/webscraper-plus, veľmi jednoduchý wrapper pracujúci na základe štandardných regulárnych výrazov

[iii] http://download.cnet.com/HTMLAsText/3000-13455_4-78220.html, tento nástroj doluje plain texty zo vstupných HTML dokumentov odstránením „tagov“ a obrázkov. Jedná sa o veľmi primitívny nástroj, ktorý navyše pracuje len s lokálnymi kópiami HTML dokumentu.

[iv] http://sourceforge.net/projects/websitescraper/, jednoduchý wrapper napísaný v jazyku JAVA, využíva mechanizmu reg. výrazov v tomto jazyku na spracovanie cieľového dokumentu, znalosť reg. výrazov je nutná!

[v] Informácie v 3. kapitole pochádzajú z empirického skúmania webových stránok: http://www.idnes.cz/, http://www.novinky.cz/, http://aktualne.centrum.cz/, http://www.denik.cz/, http://aktualne.atlas.sk, http://www.aktuality.sk, http://www.topky.sk.

[vi] Táto štruktúra stránok odpovedá štruktúre napr. na http://www.idnes.cz/, http://www.novinky.cz/, http://aktualne.centrum.cz/, http://www.denik.cz/, http://aktualne.atlas.sk, http://www.aktuality.sk, http://www.topky.sk.