Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Výběr informací z lékařských zpráv

Plzeň 2016 Stanislav Škovran

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 3. května 2016

Stanislav Škovran

Poděkování

Děkuji Prof. Ing. Karlu Ježkovi, CSc., za odborné vedení mé bakalářské práce a za cenné rady, které mi pomohly tuto práci vytvořit.

V Plzni dne 3. května 2016

Stanislav Škovran

Abstract

Text mining for medical records.

The introduction of the theoretical part provides a basic outline of the possibility of general textmining followed by a description of the Java library Apache OpenNLP. The problem of anonymity of medical records is considered as well.

The practical part contains description of main ideas of the Java programs Anonymizer and ReportsInfoMiner. These programs are the essential part of the thesis. They perform the anonymization of the medical data and create relation database with structured data. The user documentation of the programs is placed at the attachment.

Abstrakt

V teoretické části je uveden obecný přehled vytěžování informací z nestrukturovaných textů. Je také zmíněna knihovna Apache OpenNLP. Dále je probírán problém anonymizace lékařských zpráv.

Realizační část se soustřeďuje na popis programu Anonymizer a ReportsInfoMiner. Tyto programy jsou náplní bakalářské práce a slouží k anonymizaci zpráv a k jejich strukturalizaci do relační databáze. V příloze je pak uživatelská dokumentace k oběma programům.

Obsah

1 Úvod

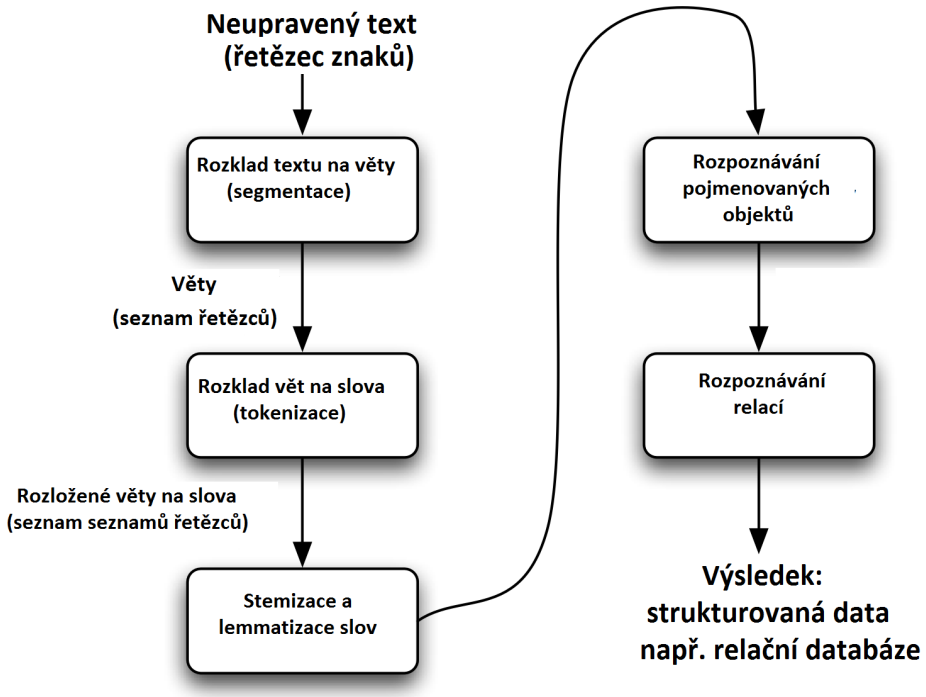
V lékařské dokumentaci se často vyskytují zprávy psané jako souvislý, nestrukturovaný text. Je obtížné z většího množství takovýchto textů vybírat důležité informace, případně vytvářet statistiky úkonů, podaných léků a dalších položek. Cílem této práce je vytvořit program, který dokáže vytěžit strukturované informace z takovýchto zpráv. Výstupem by měla být relační databáze. Její strukturu a definici vyhledávaných entit by měl mít uživatel možnost konfigurovat bez zásahu do programového kódu.

Se zpracováním lékařských zpráv je také spojen problém anonymizace dat. Ukazuje se jako důležité mít možnost nahradit ve zprávách osobní údaje (jméno, příjmení) anonymními tvary. S takto anonymizovanými zprávami pak provedeme strukturalizaci do databáze a jsou připraveny například pro statistické přehledy. Program pro anonymizaci je také součástí této bakalářské práce.

Jako programovací jazyk jsem použil Javu. Důvodem je využití javovské knihovny OpenNLP pro některé operace s textem a využití GUI objektů Javafx pro práci s tabulkami databází.

2 Počítačové zpracování přirozeného jazyka

Proces zpracování přirozeného jazyka si můžeme zobrazit na obrázku 2.1.



Obrázek 2.1: Zpracování přirozeného jazyka

Projdeme si postupně všechny kroky tohoto procesu.

2.1 Rozklad textu na věty

Tento krok se zdá být nejjednodušší. Na rozdělení se dají použít interpunkční znaménka. Znaky „?!“ znamenají téměř vždy konec věty. Výjimky typu ”Yahoo! Inc.”jsou vzácné. Horší situace je se znakem tečka. Mějme následující text:

"Tak např. USA mají hlavní město Washington D.C. Situace ve světě na počátku 20. století. Na počátku 20. století se velmoci zaměřovaly na výboje mimo evropský kontinent – dobývání nových území v Asii a Africe. Koncem 19. st. však byly kolonie již rozděleny mezi velmoci, přičemž se měnil poměr sil mezi nimi."

Vidíme, že orientace podle interpunkčních znamének a velkých počátečních písmen zde nepovede k přesnému rozkladu na věty. Do algoritmu by se musel zabudovat seznam zkratek a provést i další úpravy. Proto se používají dokonalejší postupy, některé nastíníme v následujícím textu. Začněme definicí pojmu ***věta*** (podle [5]):

Ukazuje se, že definice věty je poměrně obtížný úkol. Ještě v 19. století nebyla věta pokládána za systémovou jednotku na rozdíl od ostatních systémových jednotek (slova, znaky). V našem případě ještě přistupuje problém s odborností lékařských zpráv a potenciální problémy se zkratkami netypickými pro běžný text. Dodnes se nepodařilo vytvořit obecně přijatelnou definici věty. Uveďme aspoň některé vlastnosti, které se vyskytují ve více definicích:

• Každý prvek věty má vztah k ostatním prvkům, neexistuje prvek věty, který by nevstupoval do vztahu alespoň s jedním dalším prvkem téže věty.

• Pořadí prvků ve větě může být využito k vyjádření jejich syntaktické funkce.

• Prvky věty nemusí být vždy vyjádřené formou, ale i bez znakového nositele mohou zastupovat syntaktickou pozici, nést nějakou funkci ve větě.

• Věta je myšlenka vyjádřená slovy, ucelená ve vztahu k této myšlence.

Pro potřeby softwarového zpracování textu bude vyhovovat intuitivní a čistě formální pohled:

• Věta začíná velkým písmenem (před nímž stojí interpunkce) a končí tečkou, vykřičníkem, otazníkem, případně trojtečkou či horními uvozovkami (tedy věta jednoduchá i souvětí, každý autonomní celek).

K rozdělení textu na věty se používají různé strategie, např.:

• Hranice se určují podle pravidel daných formální definicí věty (např. pokud za interpunkčním znaménkem (tečkou) následuje velké písmeno, vloží mezi ně program znak hranice věty; pokud je předcházející tečce obsažen v předpřipraveném seznamu zkratek, není za tečkou znak hranice věty). Pro angličtinu fungují tyto programy s přesností 95% .

• Automatické naučení konců vět. Z trénovacích dat, ve kterých jsou anotátory vyznačeny hranice vět, se naučí pravidla, potřebná pro rozhodování o hranicích vět. Řešení může být založené např. na modelu maximální entropie. Na vstupu je na každém řádku věta, díky čemuž se snadno vytvoří seznam slov, za kterými je tečka, ale věta jí nekončí (zkratky, řadové číslovky, e-mailové adresy, emotikony, webové adresy, apod.) Přesnost pro angličtinu je až 99,5% - záleží na velikosti a kvalitě trénovacích dat. - Právě tuto strategii uplatňuje knihovna OpenNLP, která je použita v této bakalářské práci.

2.2 Rozklad vět na slova

I zde by se dala použít knihovna OpenNLP, která má metodu, která rozděluje větu na slova. Je založena opět na principu tréninkových dat. V této práci je ale využíván vlastní algoritmus, který prochází větu a jako oddělovače slov bere všechny znaky, které nejsou obsaženy v následující sadě písmen:

”aáäbcčdd’eéěfghiíjklmnňoóöpqrřsštt’uúůüvwxyýzž”

Současně je použit tento oddělovací algoritmus i jako filtr, který odstraňuje ze slov všechny znaky, které neodpovídají písmenu. Při použití knihovny OpenNLP vznikl problém při zpracování některých vstupních textů, kdy si knihovna nedokázala poradit s dělícími značkami <,>. Tyto značky jsou obsaženy např. ve formátu XML, který má čím dál větší uplatnění i v administrativě. Program Anonymizator byl vyzkoušen na anonymizaci vzorového XML souboru Policie ČR:

```
<?xml version="1.0" encoding="utf-8"?>
<Document Id="1080000504284">
  <PlainText>
    <![CDATA[
```

```
Osoba TEST Test, nar. 1.10.1983/1234, zaměstnán u firmy TEST a.s.,
    IČO 123456, jel dne 1.1.2008 v 20:00 s vozidlem ŠKODA FAVORIT,
    bílá barva, RZ TTT1234. Při náhodném zastavení hlídkou PČR bylo
    v zavazadlovém prostoru objeveno 5kg OPL - kokain.
Spolujezdcem byla osoba HOŘÍNEK Jakub, nar.7. května 1987.

]]>
  </PlainText>
  <Entities>
  <Entity id="1080000353709" type="Person">HOŘÍNEK JAKUB, nar.
     19870507</Entity>
  <Entity id="1080000504286" type="Person">TEST TEST, nar.
     19831001</Entity>
  <Entity id="1080000504294" type="Firm">TEST, IČ 123456</Entity>
  <Entity id="1080000504302" type="Car">TTT1234,ŠKODA ,BÍLÁ</Entity>
  <Entity id="1080000504312" type="UO">kokain, 5kg</Entity>
  </Entities>
  <Edges>
  <Edge parent="1080000504286" child="1080000353709"></Edge>
  <Edge parent="1080000504286" child="1080000504294"></Edge>
  <Edge parent="1080000504286" child="1080000504302"></Edge>
  <Edge parent="1080000504302" child="1080000504312"></Edge>
  </Edges>
</Document>
```

Po anonymizaci:

```
<Document Id="1080000504284">
  <PlainText>
    <![CDATA[
Osoba TEST Test, nar. 1.10.1983/1234, zaměstnán u firmy TEST a.s.,
    IČO 123456, jel dne 1.1.2008 v 20:00 s vozidlem ŠKODA FAVORIT,
    bílá barva, RZ TTT1234. Při náhodném zastavení hlídkou PČR bylo
    v zavazadlovém prostoru objeveno 5kg OPL - kokain.
Spolujezdcem byla osoba Novák001 Adam001, nar.7. května 1987.

]]>
  </PlainText>
  <Entities>
  <Entity id="1080000353709" type="Person">Novák001 Adam001, nar.
     19870507</Entity>
  <Entity id="1080000504286" type="Person">TEST TEST, nar.
     19831001</Entity>
```

```
  <Entity id="1080000504294" type="Firm">TEST, IČ 123456</Entity>
  <Entity id="1080000504302" type="Car">TTT1234,ŠKODA ,BÍLÁ</Entity>
  <Entity id="1080000504312" type="UO">kokain, 5kg</Entity>
  </Entities>
  <Edges>
  <Edge parent="1080000504286" child="1080000353709"></Edge>
  <Edge parent="1080000504286" child="1080000504294"></Edge>
  <Edge parent="1080000504286" child="1080000504302"></Edge>
  <Edge parent="1080000504302" child="1080000504312"></Edge>
  </Edges>
</Document>
```

Při použití knihovny OpenNLP by došlo k porušení XML struktury

2.3 Zpracování slov

Vzhledem k velké ohebnosti češtiny je problém při vyhledávání slov v textu z důvodu různých tvarů jednoho slova. Řešením může být tzv. lemmatizace a stemmizace [11].

2.3.1 Lemmatizace

Při procesu lemmatizace je hledán základní neboli normalizovaný tvar daného slova tzv. „lemma“. Za normalizovaný stav je považován u podstatných jmen první pád jednotného čísla, u přídavných jmen první pád jednotného čísla základního stupně mužského rodu, u sloves jde o tvar slova v infinitivu a podobně. Příklad: Být - je, byla, bude. Pán - pánům, pánovi, pane.

Existují tři základní typy lemmizačních algoritmů:

• Algoritmy založené na pravděpodobnostním modelu. Učí se na trénovacích datech, což je tabulka slov v různých gramatických tvarech a jim odpovídající lemma - základní tvar. Na základě maximalizace entropie se pak pro dané slovo vybere nejpravděpodobnější tvar lemmatu.

• Algoritmy založené na gramatických pravidlech. Zpracovávají přípony slov

a snaží se z daných pravidel určit základní tvar slova. Tyto algoritmy ale mají problémy s nepravidelnostmi gramatiky.

• Algoritmy založené na hrubé síle. Využívají obrovský slovník se všemi tvary slov a jim odpovídajícími základními tvary. Nevýhodou těchto algoritmů je potřeba obrovské databáze dat a nutnost jejich aktualizace s vývojem jazyka, hlavně slovní zásoby. Tento algoritmus je využit v bakalářské práci.

V praxi se také používají hybridní algoritmy, například kombinace algoritmu založeného na gramatických pravidlech s algoritmem hrubé síly, použitém ale jen na nepravidelných tvarech.

2.3.2 Stemmizace

Stemmizace je podobná procesu lemmatizace až na skutečnost, že se v tomto případě nehledá normalizovaná forma, nýbrž kořen (stem) slova. Příklad: vodárna, vodovod, vodník, zavodnit, odvodnit, voda, ... společný kořen slova je „vod“.

Z morfologického hlediska dělíme jazyky na následující základní skupiny[7]:

• Izolační typ - Izolační typ jazyka se vyznačuje špatnou diferenciací morfému a slova, existuje pouze diferenciace slova a věty. Skloňování jmen a časování sloves se neděje pomocí afixy, slova jsou neměnná, proto většinou krátká, jednomorfematická. V lexiku se vyskytuje výrazné množství formálních slov, málo synonymie a homonymie. Větná konstrukce se tvoří pevným slovosledem (substantivum před verbem je subjektem, substantivum za verbem značí objekt apod.) a pomocnými slovy. V syntaxi můžeme pozorovat hojnost vedlejších vět. Mezi zástupce izolačního typu patří jazyky západní Evropy, např. angličtina, francouzština, zčásti němčina.

• Aglutinační typ - U aglutinačního typu jazyka na rozdíl od izolačního se dobře projevuje vyvinutá diferenciace morfému a slova, slova a věty. V rámci morfologické stavby existuje kořen, na který se navěšují formální elementy, tím vzniká dlouhé slovo složené z kořene a afixů (např. u substantiva číslo, pády včetně mnoha adverbiálních významů). S tím také souvisí skutečnost, že aglutinační typ je charakteristický menším počtem formálních slov a převahou slov funkčních. V lexikonu se vyskytují bohaté prostředky

pro skloňování a časování, které nahrazují nedostatek slovních druhů. Ve slovotvorbě je malý výskyt složenin. U sloves se nacházejí hojně infinitivy, participia a slovesná jména V syntaxi je dán pevný slovosled, vedlejší věty jsou na ústupu. K zástupcům aglutinačních jazyků patří jazyky altajské (tzn. turečtina, mongolština), ugrofinské (maďarština, finština), arménština, nová perština, japonština a korejština.

• Flexivní typ - Flexivní jazykový typ je charakteristický dobře vyvinutou diferenciací morfému a slova, slova a věty. Vlivem redukce koncovek dochází k hromadění významů v koncovce, slovo se skládá z kmene a koncovky. Koncovka nese najednou sémantické a syntaktické vlastnosti, díky tomuto jevu je možno nejjasněji rozlišit slovní druhy. V morfologické rovině jazyka je silně rozvinutá shoda, velmi zřetelné vyjádření kategorie slova a kategorie věty. Zástupci flexivních jazyků jsou jazyky slovanské (čeština, ruština, polština), baltské (litevština, lotyština), bantuské, latina, řečtina.

Stemmizace je vhodná pro jazyky z izolačního typu, speciálně v angličtině existuje několik algoritmů (Porterův, Lovinsové, Paice-Huskův, Dawsonův), založených na pravidlech pro koncovky slov. Pro flexivní jazyky je stemmizace značně obtížnější vzhledem k variabilitě koncovek. Využívají se, stejně jako u lemmizace, algoritmy hrubé síly, založené na velkých slovnících se všemi tvary slov, případně algoritmy založené na pravděpodobnosti. V této práci se stemmizace použije na speciální typ slov - jména a příjmení. Bude použit modifikovaný algoritmus hrubé síly, založený na porovnání tvaru slov v 1. a 5. pádě.

2.4 Vytvoření databáze

V okamžiku, kdy máme text rozdělený na věty a věty rozdělené na jednotlivá slova, můžeme přistoupit k poslednímu kroku zpracování - nalezení vybraných slov a vytvoření relační databáze s nalezenými entitami. Volíme takovou strukturu databáze abychom mohli pomocí sql příkazů vytěžovat co nejvíce informací z dat. Konkrétní databáze pro tuto bakalářskou práci bude popsána v dalších kapitolách.

2.5 Software na zpracování textu

Existuje poměrně velké množství softwaru na jednotlivé kroky zpracování textu i komplexní softwary zahrnující zpracování velkého množství dat. Zde popíšeme jeden open source softwarový balík GATE a jednu javovskou knihovnu pro zpracování textu.

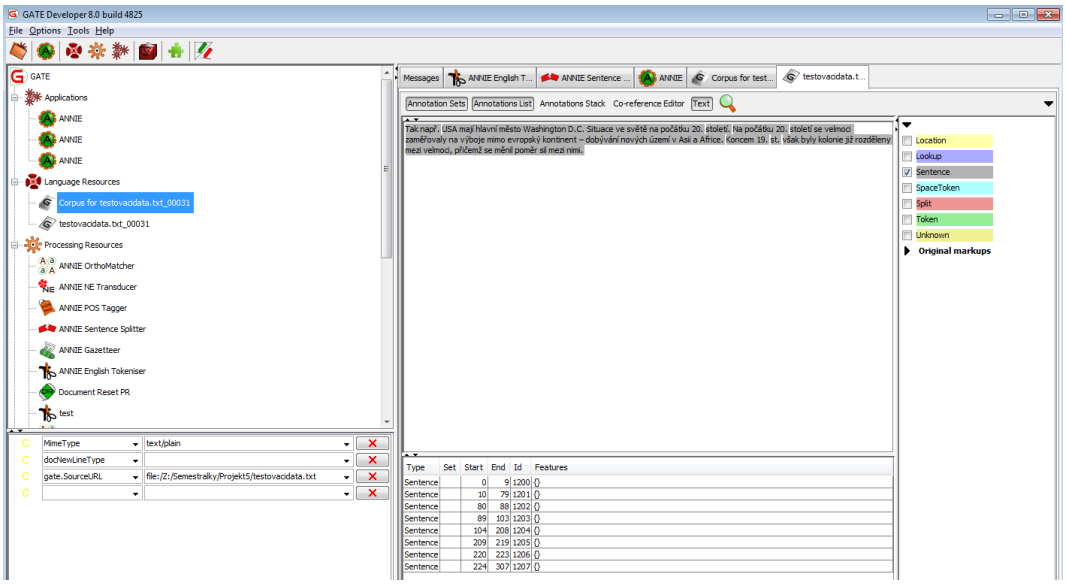
2.5.1 Program GATE

GATE je open-source free software pro zpracování přirozeného jazyka. Vyvíjí se již 15 let a pokrývá prakticky všechny úlohy zpracování textu. Pro češtinu ale neexistuje velká podpora. V roce 2012 byla na FAV ZČU obhájena bakalářská práce, která se zabývala tvorbou pluginu GATE pro češtinu [9]. Vyzkoušíme základní funkčnost na anglické verzi programu.

Pro ilustraci si ukážeme použití programu GATE na následujícím textu:

"Tak např. USA mají hlavní město Washington D.C. Situace ve světě na počátku 20. století. Na počátku 20. století se velmoci zaměřovaly na výboje mimo evropský kontinent – dobývání nových území v Asii a Africe. Koncem 19. st. však byly kolonie již rozděleny mezi velmoci, přičemž se měnil poměr sil mezi nimi."

Po zpracování programem GATE dostaneme následující rozdělení do vět:

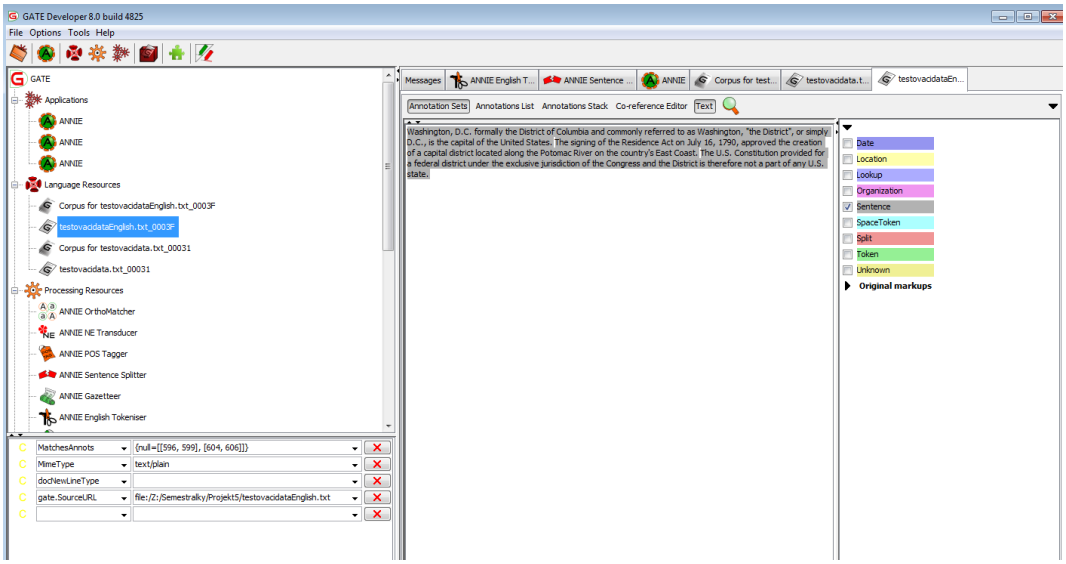


<Tak např.>
<USA mají hlavní město Washington D.C. Situace ve světě na počátku 20.>
<století.>
<Na počátku 20.>
<století se velmoci zaměřovaly na výboje mimo evropský kontinent – dobývání nových území v Asii a Africe.>
<Koncem 19.>
<st.>
<však byly kolonie již rozděleny mezi velmoci, přičemž se měnil poměr sil mezi nimi.>

Vidíme, že rozdělení dopadlo neuspokojivě. Anglická verze si vůbec neporadila se zkratkami a řadovými číslovkami. Pro srovnání zpracujeme následující anglický text:

"Washington, D.C. formally the District of Columbia and commonly referred to as Washington, "the District", or simply D.C., is the capital of the United States. The signing of the Residence Act on July 16, 1790, approved the creation of a capital district located along the Potomac River on the country's East Coast. The U.S. Constitution provided for a federal district under the exclusive jurisdiction of the Congress and the District is therefore not a

part of any U.S. state."



Jednotlivé věty:
<Washington, D.C. formally the District of Columbia and commonly referred to as Washington, "the District", or simply D.C., is the capital of the United States.>
<The signing of the Residence Act on July 16, 1790, approved the creation of a capital district located along the Potomac River on the country's East Coast.>
<The U.S. Constitution provided for a federal district under the exclusive jurisdiction of the Congress and the District is therefore not a part of any U.S. state.>

Zde si program poradil se všemi zkratkami a věty rozdělil správně. Nicméně pro češtinu je tento program nevhodný.

2.5.2 Knihovna OpenNLP

Další možností je použít knihovnu, která obsahuje metody pro zpracování textu. Ukážeme si Java knihovnu Apache OpenNLP, kterou je možno použít i pro češtinu a navíc v ní existuje možnost strojového učení. Nejdříve si ukážeme výsledky segmentace našeho cvičného textu, který jsme zkoušeli v programu GATE a potom přistoupíme k popisu knihovny. Rozklad textu na jednotlivé věty:

<Tak např. USA mají hlavní město Washington D.C.>
<Situace ve světě na počátku 20. století.>
<Na počátku 20. století se velmoci zaměřovaly na výboje mimo evropský kontinent – dobývání nových území v Asii a Africe.>
<Koncem 19. st. však byly kolonie již rozděleny mezi velmoci, přičemž se měnil poměr sil mezi nimi.>

Rozložení proběhlo naprosto správně.

OpenNLP je javovská knihovna pro zpracování textu v přirozeném jazyce, využívající strojové učení. Samotný algoritmus strojového učení je založen na principu maximalizace entropie. Tento princip je založen na praděpodobnostním modelu.Knihovna je rozdělena do několika komponent:

SentenceModel - rozdělení textu do jednotlivých vět
TokenizerModel - rozdělení vět do slov
TokenNameFinderModel - nalezení konkrétních entit ve větách
DoccatModel - kategorizace celých dokumentů
ChunkerModel - rozdělení vět do syntakticky souvisejících skupin slov
POSModel - identifikace slov jako slovních druhů (Part-of-Speech Tagger) v závislosti na kontextu

Obecný princip užití komponent:

Všechny komponenty OpenNLP pracují na podobném principu. Vytvoří se instance modelu, která se inicializuje souborem s daty získanými při strojovém učení, potom se vytvoří instance vlastního nástroje, která zpracuje zkoumaný text:

```
// inicializace modelu
InputStream modelIn = new FileInputStream("lang-model-name.bin");
SomeModel model = new SomeModel(modelIn);

// vytvoreni nastroje
ToolName toolName = new ToolName(model);

// zpracovani textu
String output[] = toolName.executeTask("Toto je cvicny text pro
    zpracovani.");
```

- Komponenta pro rozdělení textu do vět

Základní komponenta na segmentaci vstupního textu. Tuto komponentu využijeme i v této bakalářské práci.

```
// priprava modelu
InputStream modelIn = new FileInputStream("cz-sent.bin");
SentenceModel model = new SentenceModel(modelIn);
// vytvoreni nastroje a zpracovani textu
SentenceDetector = new SentenceDetectorME(model);
String sentences[] = SentenceDetector.sentDetect(text);
```

V poli sentences jsou uloženy jednotlivé věty textu.

Strojové učení komponenty:

V textovém souboru cz-sent.train je text pro strojové učení. Obsahuje jednotlivé věty, každou na novém řádku.

```
// ucebni data
ObjectStream<String> lineStream = new PlainTextByLineStream(new
    FileInputStream("cz-sent.train"));
ObjectStream<SentenceSample> sampleStream = new
    SentenceSampleStream(lineStream);

SentenceModel model;
// ucici proces
model=SentenceDetectorME.train("cz",sampleStream);

// zpracovani vysledku uceni do souboru cz-sent.bin
OutputStream modelOut = null;
modelOut = new BufferedOutputStream(new
    FileOutputStream("cz-sent.bin"));
model.serialize(modelOut);
modelOut.close();
```

Výsledkem učení je binární soubor cz-sent.bin, který následně slouží k inicializaci modelu SentenceModel.

- Komponenta pro rozdělení vět na slova

Příklad použití:

```
// priprava modelu
InputStream modelIn = new FileInputStream("cz-token.bin");
TokenizerModel model = new TokenizerModel(modelIn);

// priprava nastroje
Tokenizer tokenizer = new TokenizerME(model);

//zpracovani vety
String tokens[] = tokenizer.tokenize("Zkusebni veta pro rozklad na
    slova");
```

V poli tokens jsou jednotlivá slova: ”Zkusebni”, ”veta”, ”pro”, ”rozklad”, ”na”, ”slova”.

Strojové učení komponenty:

V souboru cz-sent.train je text pro strojové učení. Obsahuje věty, každou na novém řádku. Ve větách jsou slova oddělena mezerou případně (pro víceslovné výrazy) speciální značkou <SPLIT>.

```
// ucebni data
ObjectStream<String> lineStream = new PlainTextByLineStream(new
    FileInputStream("cz-token.train"));
ObjectStream<SentenceSample> sampleStream = new
    SentenceSampleStream(lineStream);

TokenizerModel model;
// ucici proces
model=TokenizerME.train("cz",sampleStream);

// zpracovani vysledku uceni do cz-token.bin
OutputStream modelOut = null;
modelOut = new BufferedOutputStream(new
    FileOutputStream("cz-token.bin"));
model.serialize(modelOut);
modelOut.close();
```

Výsledkem je soubor cz-token.bin.

- Komponenta pro vyhledávání jmen a dalších entit

Je schopna vyhledávat různé entity, čísla, data. Její výkon závisí na kvalitě a velikosti tréninkového souboru. Je výhodné používat na trénink text stejného typu jaký pak budeme zkoumat (např. novinové články, zdravotní dokumentace, beletrie atd.). Použití:

```
//priprava modelu
InputStream modelIn = new FileInputStream("en-ner-person.bin");
TokenNameFinderModel model = new TokenNameFinderModel(modelIn);

// nastroj
NameFinderME nameFinder = new NameFinderME(model);

// zpracovani jedne vety
```

```
String sentence[] = new String[]{
    "Petru",
    "Novakovi",
    "je",
    "61",
    "let"
    "."
    };

Span nameSpans[] = nameFinder.find(sentence);
String[] result = Span.spansToStrings(nameSpans, sentence);
```

V poli result jsou všechny nalezené entity ve větě.

Strojové učení komponenty:

V souboru cz-entita.train je text pro strojové učení. Obsahuje věty v nichž entity, které učíme, jsou obaleny <START:nazev-entity> entita <END>. Např.:

Když zakrátko Magora zavřeli , šli <START:surname> Němec <END> a <START:surname> Jirousová <END> za Václavem a rozhodli se , že případ budou publikovat.

Zde je vyznačena entita "surname". Podobně můžeme vytvořit učební soubory i pro další entity, např. pro křestní jména:

Když zakrátko Magora zavřeli , šli Němec a Jirousová za <START:forename> Václavem <END> a rozhodli se , že případ budou publikovat.

```
// priprava modelu
ObjectStream<String> lineStream =new PlainTextByLineStream(new
    FileInputStream("cz_entita.train"));
ObjectStream<NameSample> sampleStream = new
    NameSampleDataStream(lineStream);

// proces uceni
TokenNameFinderModel model = NameFinderME.train("cz",
    nazev-entity, sampleStream, Collections.<String,
    Object>emptyMap());
```

```
// ulozeni vysledku do souboru entita.bin
FileOutputStream outFileStream = new
    FileOutputStream("entita.bin");
model.serialize("entita.bin");
outFileStream.close();
```

V binárním souboru entita.bin je uložen výsledek učení.

Testování modelu pro vyhledávání jmen:
Možnosti knihovny si ukážeme na konkrétním textu, ve kterém budeme vyhledávat různé entity. Největší problém je vyvoření dostatečně velkého tréninkového souboru. Nalezl jsem na internetu poměrně velký korpus, který se dal pro naše účely použít - [10]. Korpus obsahuje přes 6000 vět. Soubor jsem musel ale upravit. Původní korpus má entity vyznačeny pomocí html tagů. OpenNLP ale používá pro vyznačení entit jinou syntaxi, jak jsme si ukázali výše.

Vytvořil jsem tři učební soubory - forename.train, countries.train, surname.train. Pomocí nich jsem pak vygeneroval tři binární soubory jako výsledky strojového učení - forename.bin, surname.bin, countries.bin.
Jako testovací soubor pro analýzu jsem vzal novinový článek:

Letošní Silvestr byl ten horší, tvrdí záchranáři. Vážně se zranily i děti 31. prosince 2014 20:25, aktualizováno 1. ledna 2015 13:00 Letošní oslavy příchodu Nového roku byly podle záchranářů bouřlivější a náročnější než v minulých letech. Pražská záchranka byla dokonce přetížená už před půlnocí a na pomoc v jednu chvíli čekalo až 26 lidí. V Brně vypadl z okna 15letý opilý mladík, na Kaplicku ležela na ulici opilá 14letá dívka a v Bochově pyrotechnika vážně zranila 12letého chlapce. Silvestrovské oslavy na Václavském náměstí | (1:04) | video: iDNES.cz Od půlnoci do šesti hodin rána zasahovali hasiči u 92 požárů. „Dlouhodobý denní průměr počtu požárů byl během prvních šesti hodin nového roku překročen o 80 procent," uvedla zastupující mluvčí hasičů Michaela Franclová.Nejvíce, 16 požárů, bylo v Moravskoslezském a Jihomoravském kraji, o jeden méně měl Ústecký kraj. „V hlavním městě prvních šest hodin nového roku hořelo desetkrát," dodala mluvčí Franclová. Nejklidnější oslavy strávili hasiči v Královéhradeckém kraji, kde nehořelo ani jednou.Mnohem více výjezdů hlásí také pražská záchranka, během noční směny na Nový rok jich bylo 317. „To je o cca 20 procent více než vloni," uvedla její mluvčí Jiřina Ernestová. „Výrazně, o 50 procent, přibylo událostí řešených do půlnoci, bylo jich 115. To plně vytížilo dostupné kapacity, takže tradiční popůlnoční

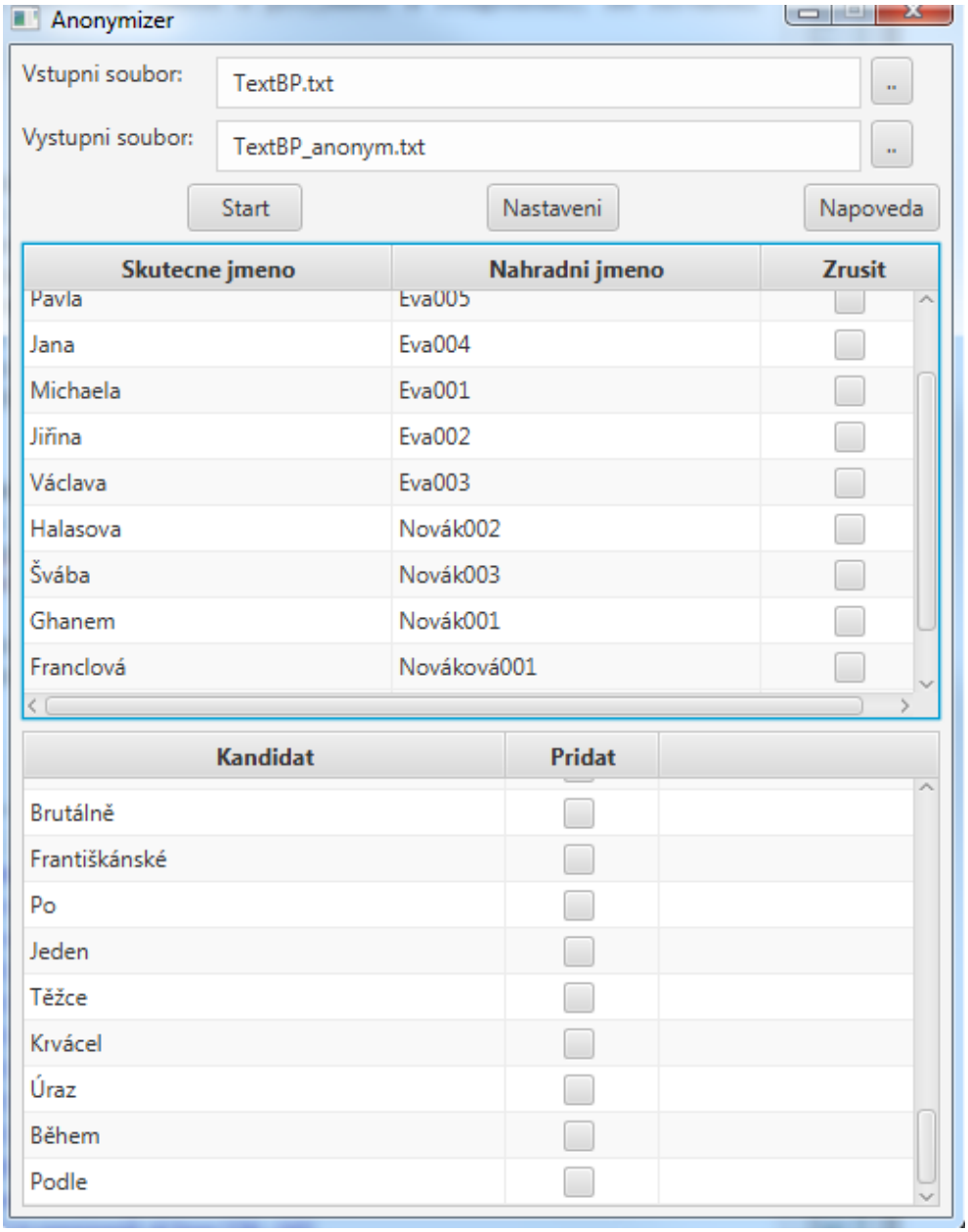
nápor, navíc o něco vyšší než vloni, spolu s obtížně sjízdnými komunikacemi, způsobil vážné kapacitní potíže," popsala mluvčí.Mezi půlnocí a čtvrtou hodinou byla podle ní pražská záchranka přetížená. „Maximem bylo 26 čekajících, nevyřízených událostí. V případě méně závažných událostí se tak výrazně prodloužily čekací doby, které se pohybovaly až kolem jedné hodiny," vysvětlila.V Praze vyrazily do ulic díky teplému počasí odhadem tisíce lidí, většinou se jednalo o zahraniční turisty. Shromažď'ovali se nejvíce v okolí Václavského a Staroměstského náměstí. Někteří z nich dokázali odpalovat pyrotechniku i několik hodin.„Pár minut před první hodinou ranní provedli hasiči v Praze 8 násilný vstup do bytu, aby vysvobodili čtyři osoby uvízlé na balkoně," popsala mluví Franclová. V Dlouhé ulici hasili plameny na střeše. Palety a obaly na plechové střeše chytly od zábavní pyrotechniky.Město se snažilo před silvestrovskými oslavami chránit památky, kolem sochy svatého Václava na Václavském náměstí i sousoší Mistra Jana Husa na Staroměstském náměstí před oslavami vyrostly ploty. Pro auta bylo uzavřené Václavské náměstí i přilehlé ulice, stejně jako okolí Staroměstského náměstí. Vodičkovou ulicí jezdily jen tramvaje.Nad ránem už v centru Prahy pracovaly úklidové čety:Úklid centra Prahy po novoročních oslavách | (1:03) | video: iDNES.cz Strážníci v Brně měli na Silvestra napilno už od brzkého odpoledne. Zaměstnali je hlavně podnapilí lidé. Před půl druhou strážníky přivolali kolemjdoucí k viditelně otřesené a plačící sedmadvacetileté ženě do Kobližné ulice.„Brutálně ji napadla jiná, opilá žena, která ji údajně bila pěstmi do obličeje a potom ji ještě připravila o 80 korun a utekla," řekl mluvčí brněnské městské policie Jakub Ghanem. Strážníci ženě krvácející z obličeje přivolali sanitku a případ předali kolegům ze státní policie.O chvíli později si věřící nevěděli rady s opilým návštěvníkem kostela ve Františkánské ulici. „Muž se choval nevhodně a ostatní pohoršoval. Po příjezdu hlídky se ovšem uklidnil a z kostela odešel," přiblížil mluvčí.Jeden opilec v Brně už skončil v nemocnici. Těžce podnapilého osmačtyřicetiletého muže našli strážníci ležet na lavičce u Halasova náměstí. Krvácel z hlavy a byl podchlazený. Úraz si podle vlastních slov způsobil sám, když opilý upadl.Chlapec vypadl z okna, 14letá dívka ležela opilá na ulici Během noci museli jet záchranáři k patnáctiletému chlapci, který s téměř dvěma promile alkoholu vypadl v Brně při oslavování Nového roku z okna druhého patra. Podle brněnského policejního mluvčího Pavla Švába se mladík zranil, ale nebyl v ohrožení života.

Výsledek vyhledávání příjmení:

„To je o cca 20 procent více než vloni," uvedla její mluvčí Jiřina Ernestová.

Ernestová
———————
„Pár minut před první hodinou ranní provedli hasiči v Praze 8 násilný vstup do bytu, aby vysvobodili čtyři osoby uvízlé na balkoně," popsala mluví Franclová.
Franclová
———————
Město se snažilo před silvestrovskými oslavami chránit památky, kolem sochy svatého Václava na Václavském náměstí i sousoší Mistra Jana Husa na Staroměstském náměstí před oslavami vyrostly ploty.
Mistra
Husa
———————
„Brutálně ji napadla jiná, opilá žena, která ji údajně bila pěstmi do obličeje a potom ji ještě připravila o 80 korun a utekla," řekl mluvčí brněnské městské policie Jakub Ghanem.
Ghanem
———————
Krvácel z hlavy a byl podchlazený.
Krvácel
———————
Podle brněnského policejního mluvčího Pavla Švába se mladík zranil, ale nebyl v ohrožení života.
Švába

Vidíme, že program si s úkolem docela poradil, je tam pouze jedna chyba: "Krvácel"není v tom článku příjmení. Všechna ostatní příjmení byla nalezena. Původně jsem chtěl použít tuto komponentu na vyhledávání lékařských pojmů v textu. Nepodařilo se mi ale vytvořit dostatečně velký korpus pro to, aby byly výsledky uspokojivé. Rozhodl jsem se tedy pro vlastní řešení s předem definovanými slovníky pojmů, které se v textu vyhledávají za pomoci lemmatizace.
Pro porovnání: program Anonymizator našel v článku všechna jména a příjmení a rozpoznal, že Krvácel není příjmení:



Obrázek 2.2: Programu Anonymizátor

3 Anonymizace dat

Vzhledem k citlivé povaze zpracovávaných lékařských dat se ukazuje jako nezbytné provést před samotným vytěžováním dat z lékařských zpráv jejich anonymizaci. Spočívá v nahrazení jmen a příjmení anonymními tvary. Protože počet různých jmen v textu není obecně omezený, zvolil jsem pro anonymizaci očíslované tvary jmen Eva Nováková a Adam Novák. Tedy například text

Až při závěrečné čtvrtině nedělního Pražského maratonu spatříte utíkat subtilní blondýnku s číslem R 6644 na červeném podkladě, jakým se budou lišit čísla členů štafet, vězte, že ho nese nejlepší česká triatlonistka ***Vendula Frintová****. Před ní ve stejném týmu osobností poběží skifař* ***Ondřej Synek****, biatlonista* ***Michal Šlesingr*** *a herec a moderátor* ***Dalibor Gondík****.*

se převede na text

Až při závěrečné čtvrtině nedělního Pražského maratonu spatříte utíkat subtilní blondýnku s číslem R 6644 na červeném podkladě, jakým se budou lišit čísla členů štafet, vězte, že ho nese nejlepší česká triatlonistka ***Eva001 Nováková001****. Před ní ve stejném týmu osobností poběží skifař* ***Adam001 Novák001****, biatlonista* ***Adam002 Novák002*** *a herec a moderátor* ***Adam003 Novák003****.*

Součástí anonymizace musí být také vytvoření souboru se seznamem všech nahrazených jmen a příjmení spolu s jejich náhradami - pro případnou zpětnou interpretaci. Tedy zde by to bylo:

Vendula	*Eva001*
Frintová	*Nováková001*
Ondřej	*Adam001*
Synek	*Novák001*
Michal	*Adam002*
Šlesingr	*Novák002*
Dalibor	*Adam003*
Gondík	*Novák003*

3.1 Základní algoritmus

Anonymizaci provádí program Anonymizer, který nahrazuje jména a příjmení v textu anonymními tvary. Největší problém je samozřejmě nalézt v textu všechna jména a příjmení. Původně jsem chtěl použít knihovnu OpenNLP s komponentou pro vyhledávání jmen. Protože ale potřebujeme rozeznat, jestli je jméno/příjmení mužské nebo ženské, musel by se učící korpus podle rodu jmen rozdělit. Při velikosti korpu je to ale velmi časové náročná práce. Rozhodl jsem se pro jiné řešení: na internetu existuje soubor se seznamem jmen a příjmení a jejich 5. pádů. [2]. Na uvedené adrese je ke stažení zip soubor se šesti soubory ve formátu csv. Tyto soubory obsahují pro každé jméno a příjmení tři položky: četnost, 1. pád, 5. pád. Soubory jsem převedl do sqlite databáze NamesRoots.db, která obsahuje 4 tabulky: ženská jména, mužská jména, ženská příjmení, mužská příjmení. Každá tabulka má strukturu:

ID	- pořadové číslo (primary key)
Frequency	- četnost v populaci
Nominative	- 1. pád
Vocative	- 5. pád
Root	- kořen slova

Table: FORENAME_FEMALE

	ID	Frequency	Nominative	Vocative	Root
	Filter	Filter	Filter	Filter	Filter
1	1	316559	Marie	Marie	Mari
2	2	274303	Jana	Jano	Jan
3	3	160317	Eva	Evo	Ev
4	4	150573	Hana	Hano	Han
5	5	148688	Anna	Anno	Ann
6	6	124871	Věra	Věro	Věr
7	7	119366	Lenka	Lenko	Lenk
8	8	110936	Kateřina	Kateřino	Kateřin
9	9	110046	Alena	Aleno	Alen
10	10	103702	Lucie	Lucie	Luci
11	11	102321	Petra	Petro	Petr
12	12	94475	Jaroslava	Jaroslavo	Jaroslav
13	13	85939	Ludmila	Ludmilo	Ludmil
14	14	82985	Helena	Heleno	Helen
15	15	81543	Martina	Martino	Martin

Obrázek 3.1: Část databáze pro stemming ženských jmen

Na obrázku 3.1 je struktura databáze a část tabulky pro ženská jména. Sloupec Root je stem jmen a příjmení a je klíčový v algoritmu pro určení, zda je slovo jménem. Ukazuje se, že pro určení kořene speciální skupiny slov - jmen a příjmení lze s vysokým procentem úspěšnosti použít porovnání prvního a pátého pádu, respektive jejich průnik. Příklad:
Jana, Jano -> Jan
Pavel, Pavle -> Pav
Valenta, Valento -> Valent
Existuje několik výjimek, které se musejí vyřešit zvlášť v programu, např.:
Marie, Marie -> Mari
Celý algoritmus vyhledání jména vypadá následovně:

1. Ze vstupu se načte jedna věta - na rozdělení textu na věty se použije knihovna OpenNLP.

2. Věta se rozloží na jednotlivá slova.

3. Slovo S se hledá v datábazi, ve sloupci Nominative nebo Vocative. Pokud je S nalezeno, je prohlášeno za jméno nebo příjmení a pokračuje se krokem 5.

4. Slovo S v nominativu ani vokativu nebylo nalezeno. Hledáme ve sloupci Root nejdelší kořen, který se shoduje se začátkem S. Pokud je nalezen, kontrolujeme zbytek slova, zda odpovídá nějaké možné koncovce jména nebo příjmení. Při zpracovávání databáze jmen a příjmení byly rozeznány následující typy koncovek, které se přidávají ke kořenu:

   - Ženská křestní jména:
   - typ Marie - Mari-**e**, Mari-**i**, Mari-**í**
   - typ Jana - Jan-**y**, Jan-**ě**, Jan-**u**, Jan-**o**, Jan-**ou**
   - Ženská příjmení:
   - typ Nováková - Novák-**ovou**, Novák-**ové**
   - Mužská křestní jména:
   - typ Jan - Jan-**a**, Jan-**ovi**, Jan-**e**, Jan-**em**
   - typ Petr - Pet-**ra**, Pet-**rovi**, Pet-**ře**, Pet-**rem**
   - typ Pavel - Pav-**la**, Pav-**lovi**, Pav-**le**, Pav-**lem**

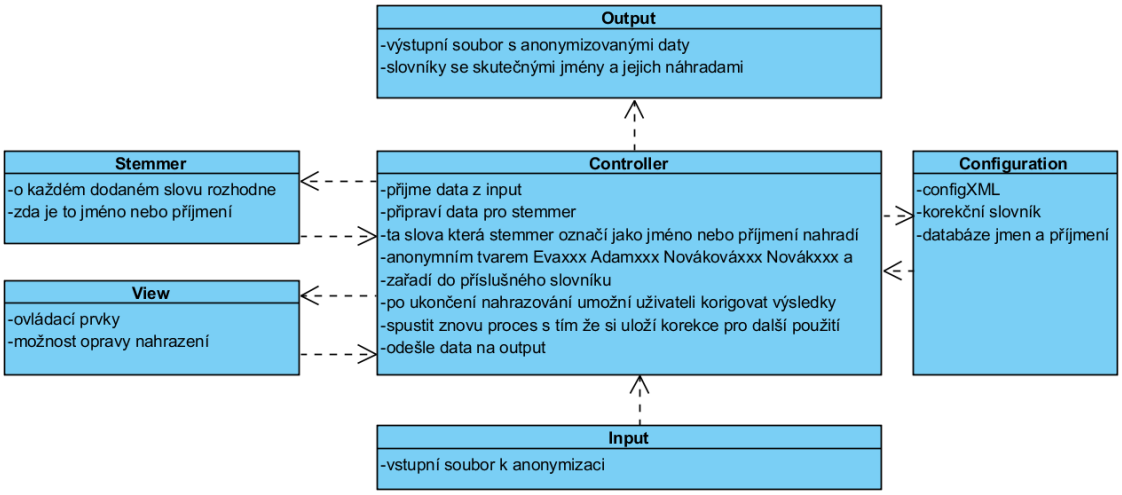
- typ Tomáš - Tomáš-**e**, Tomáš-**ovi**, Tomáš-**i**, Tomáš-**em**
- typ František - Františ-**ka**, Františ-**kovi**, Františ-**ku**, Františ-**kem**
- typ Zdeněk - Zde-**ňka**, Zde-**ňkovi**, Zde-**ňku**, Zde-**ňkem**
- typ Ivo - Iv-**a**, Iv-**ovi**, Iv-**em**
- typ Jiří - Jiří-**ho**, Jiří-**mu**, Jiří-**m**
- Mužská příjmení:
- typ Novák - Novák-**a**, Novák-**ovi**, Novák-**u**, Novák-**em**
- typ Svoboda - Svobod-**y**, Svobod-**ovi**, Svobod-**o**, Svobod-**ou**
- typ Novotný - Novotn-**ého**, Novotn-**ému**, Novotn-**ém**, Novotn-**ým**
- typ Král - Král-**e**, Král-**ovi**, Král-**i**, Král-**em**
- typ Němec - Něm-**ce**, Něm-**covi**, Něm-**če**, Něm-**cem**
- typ Vlček - Vlč-**ka**, Vlč-**kovi**, Vlč-**ku**, Vlč-**kem**

Rozhodnutí, do kterého typu slovo přísluší se děje na základě porovnání nominativu, vokativu a kořene. Pokud slovo vyhovuje některému typu i s koncovkou, prohlásíme slovo za jméno nebo příjmení.

5. Nalezené slovo je nahrazeno odpovídajícím anonymním tvarem. Na základě stemmu a přidané koncovky se určí, zda jiný gramatický tvar patří ke stejnému jménu nebo příjmení.

6. Postprocesingová kontrola uživatelem.

Schema celého programu je na Obr. 3.2. Zdrojový kód hlavní metody je v příloze A.1.



Obrázek 3.2: Schema programu Anonymizer

3.2 Funkčnost programu

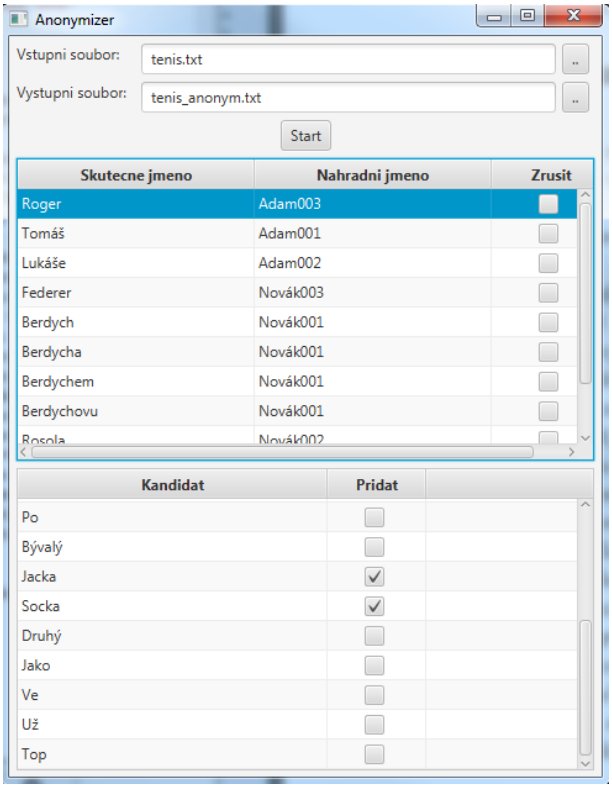
Ukážeme si funkčnost programu na testovacím příkladu a také vyzkoušíme výkonnost programu na větších datech.

Mějme následující text:

Tomáš Berdych porazil ve čtvrtém kole turnaje v Indian Wells v české tenisové bitvě Lukáše Rosola 6:2, 4:6 a 6:4. Coby nasazená devítka se probil mezi osm nejlepších. Nyní ho čeká druhý hráč světa Roger Federer ze Švýcarska. Zápas dvou momentálně nejlepších českých tenistů vypadal v prvním setu jako jasná záležitost pro devátého nasazeného Berdycha. Úvodní sadu získal za 27 minut, ve druhé se ale hra vyrovnala a Rosol dokázal soupeři dvakrát sebrat servis. Třetí set, v němž oba spolehlivě hájili svá podání, rozhodl Berdych brejkem v desáté hře. Po hodině a 52 minutách proměnil druhý mečbol a Rosola porazil i ve druhém vzájemném zápase. Federer oslavil na turnaji v Indian Wells 50. vítězství. Bývalý první hráč světa, jenž už na kalifornském podniku získal rekordní čtyři tituly, porazil domácího Jacka Socka 6:3, 6:2. Druhý nasazený hráč, jenž v Indian Wells startuje už po patnácté, se bez problémů vyrovnal s větrným počasím a dvaadvacetiletého soupeře porazil za hodinu a devět minut. Jako jasný favorit půjde i do souboje s Berdychem. Ve vzájemné bilanci vede 12:6. „Už si ani nepamatuju, kdy jsme spolu hráli poprvé, ale sledoval jsem, jak se jeho hra vyvíjela a jak se dokázal usadit v

Top 10,“ řekl Federer na Berdychovu adresu na tiskové konferenci po utkání.

Na obrázku 3.3 je vidět výsledek anonymizace textu programem Anonymizer



Obrázek 3.3: Výsledek anonymizace po 1. kroku

V první tabulce jsou zobrazeny provedené náhrady, ve druhé tabulce všechna slova začínající velkým písmenem, která nebyla vyhodnocena jako jména nebo příjmení. Ve sloupci "Přidat"máme možnost nastavit je také k náhradě. To jsme provedli pro jméno Jacka Socka. Spustíme znovu celý proces a napodruhé dostaneme ve výstupním souboru **tenis_anonym.txt** následující text:

Adam001 Novák001 porazil ve čtvrtém kole turnaje v Indian Wells v české tenisové bitvě Adam002 Novák002 6:2, 4:6 a 6:4. Coby nasazená devítka se probil mezi osm nejlepších. Nyní ho čeká druhý hráč světa Adam003 Novák003 ze Švýcarska. Zápas dvou momentálně nejlepších českých tenistů vypadal v prvním setu jako jasná záležitost pro devátého nasazeného Novák001a. Úvodní sadu získal za 27 minut, ve druhé se ale hra vyrovnala a Novák002 dokázal soupeři dvakrát sebrat servis. Třetí set, v němž oba spolehlivě hájili svá podání, rozhodl Novák001 brejkem v desáté hře. Po hodině a 52 minutách proměnil druhý mečbol a Novák002 porazil i ve druhém vzájemném zápase. Novák003 oslavil na turnaji v Indian Wells 50. vítězství. Bývalý první hráč světa, jenž už na kalifornském podniku získal rekordní čtyři tituly, porazil domácího Novák004 Novák005 6:3, 6:2. Druhý nasazený hráč, jenž v Indian Wells startuje už po patnácté, se bez problémů vyrovnal s větrným počasím a dvaadvacetiletého soupeře porazil za hodinu a devět minut. Jako jasný favorit půjde i do souboje s Novák001. Ve vzájemné bilanci vede 12:6. „Už si ani nepamatuju, kdy jsme spolu hráli poprvé, ale sledoval jsem, jak se jeho hra vyvíjela a jak se dokázal usadit v Top 10," řekl Novák003 na Novák001 adresu na tiskové konferenci po utkání.

Vidíme zde hlavní nedostatek programu - nahrazené anonymní tvary jsou jen v 1. pádu. Odstranění tohoto nedostatku by si vyžádalo další analýzu nalezených jmen. Pro naše účely získávání strukturovaných informací z lékařských zpráv je ale zatím toto nahrazování dostatečné. Ještě si uvedeme jak vypadají soubory, kde jsou uloženy slovníky náhrad:

Soubor **tenis_surname.txt**:

Federer	- Novák003
Berdycha	- Novák001
Berdych	- Novák001
Berdychem	- Novák001
Berdychovu	- Novák001
Rosol	- Novák002
Rosola	- Novák002

Soubor **tenis_forename.txt**:

Roger - Adam003
Tomáš - Adam001
Lukáše - Adam002
Lukáš - Adam002

Program správně rozeznal různé tvary stejných jmen a příjmení a přiřadil jim stejnou náhradu.

Výkonnost programu:

Program jsem použil pro anonymizaci lékařských zpráv z Domova seniorů. Obsahovaly 36 000 položek. Anonymizace trvala cca 120 minut na následující konfiguraci:

Windows 7 64-bit SP1
Intel Core i5-4460 CPU @ 3,20GHz, 8,0GB RAM.

4 Vytěžování informací z textu

Data z lékařských zpráv jsou v anonymizovaném tvaru a může se přistoupit k vytěžování informací. K tomu slouží program ReportsInfoMiner, princip jeho činnosti si teď popíšeme.

4.1 Data

4.1.1 Vstupní data

Vzhledem ke specifikaci lékařských zpráv jsou zvoleny dva formáty vstupních dat:

1. textový soubor *.txt v kódování UTF-8
2. sqlite *.db databázový soubor

Textový formát je vybrán pro jeho obecnost.
Databázový formát proto, že lékařské zprávy mohou být částečně strukturovány a uživatel požaduje vytěžovat informace jen z určité části. Časté jsou zprávy ve formě excelovských tabulek:

	Datum	Čas	Zápis	ID obyvatele
138	22.3.2014	19:08	Zápis: Strženiny na obou horních končetinách a defekt v sakru jsou nadále v péči sester. Sestru není nutné	58
139	22.3.2014	9:44	Zápis: Dnes proběhla kontrola opruzenin v oblasti konečníku a tísel - nadále přetrvávají. V ošetřování přímé péče - omýt, osušit a namazat tenkou vrstvou zinkové masti.; Dne: 22.3.2014	58
140	20.3.2014	11:02	Zápis: (Zapsal: xxx) Při ranní hygieně zjištena opruzenina v oblasti konečníku a třísel. V ošetřování přímé péče. při ranní a večerní hygieně omýt, osušit a namazat tenkou vrstvou zinkové masti.; Dne: 20.3.2014	58
141	24.3.2014	7:15	Zápis: Opruzeniny v oblasti konečníku a třísel přetrvávají. Nadále v ošetřování přímé péče. Při ranní a večerní hygieně omýt, osušit a namazat tenkou vrstvou Zinkové masti.; Dne: 24.3.2014	58
142	7.3.2014	9:58	Zápis: Po domluvě se zdravotní sestrou se u obyvatelky jedná o trvalou změnu pigmentace. Sakrum se bude při ranní a večerní hygieně promazávat Regenerační mastí.; Dne: 7.3.2014	60
143	5.3.2014	20:23	Zápis: Při ranní hygieně byla zjištěna suchá zrohovatělá kůže v sakru. Při ranní a večerní hygieně promazávat Regenerační mastí. Sestra byla informována.; Dne: 5.3.2014	60
144	1.3.2014	6:53	Zápis: Mírné začervenání na levém kotníku přetrvává,nehorší se.Při ranní a večerní hygieně promazávat Emspomou. Pokud bude obyvatel v lůžku dávat antidekubitní botičky, vypodkládat dolní končetiny tak aby se paty nedotýkali podložky. Při vysazování do invalidního vozíku vkládat do ponožky vatu.; Dne: 1.3.2014	30

Obrázek 4.1: Příklad zdravotních zpráv ve formátu Excel

Uživatel může chtít extrahovat jen informace ze sloupce se zápisem. Ostatní sloupce by rád zachoval. Excelovský soubor snadno převedeme na sqlite databázi (excel -> csv formát -> sqlite).

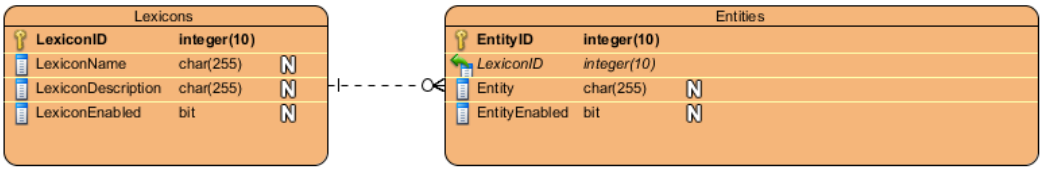
4.1.2 Pojmové slovníky

Při vyhledávání klíčových slov v textu jsou dvě hlavní možnosti: jedna založená na pravděpodobnostním modelu a obrovských trénovacích datech, druhá založená na slovnících s klíčovými pojmy. Druhý postup je zvolen v této práci. Má jednu podstatnou výhodu: dává možnost přesně definovat, které údaje jsou ve zprávách podstatné, které by se měly strukturalizovat. Klíčové pojmy lékařských zpráv jsou rozděleny do čtyř skupin:

- **Potíže** - zahrnuje nemoci, zranění, všechny stavy a události, které vyžadují zásah lékaře nebo zdravotní sestry
- **Lokalizace** - místo na těle
- **Léčba** - léčebné a ošetřovatelské postupy
- **Léky** - použité léky, masti a další zdravotní materiál

Výrazy v těchto čtyřech skupinách lze libovolně editovat, přidávat, mazat, případně přechodně vypínat a zapínat. Slova v nich jsou v 1. pádu jednotného čísla. Pro jejich vyhledávání v textu se uplatňuje lemmatizace.

Pojmy jsou uloženy v sqlite databázovém souboru lexicons.db. Databáze se skládá ze dvou tabulek, v jedné je seznam pojmových skupin, ve druhé všechny vyhledávané entity. Tabulky jsou spojeny vazbou 1:N:



Obrázek 4.2: Struktura databáze pojmových slovníků

Tabulka Lexicons - seznam pojmových skupin:

- LexiconID - číslo skupiny (primary key)

- LexiconName - název skupiny (Potíže, Lokalizace, Léčba. Léky)
- LexiconDescription - uživatelský popis skupiny
- LexiconEnabled - Ano/Ne - vypínač skupiny - pro případ, že chce uživatel hledat výsledky jen pro některé skupiny

Tabulka Entities - seznam pojmů:

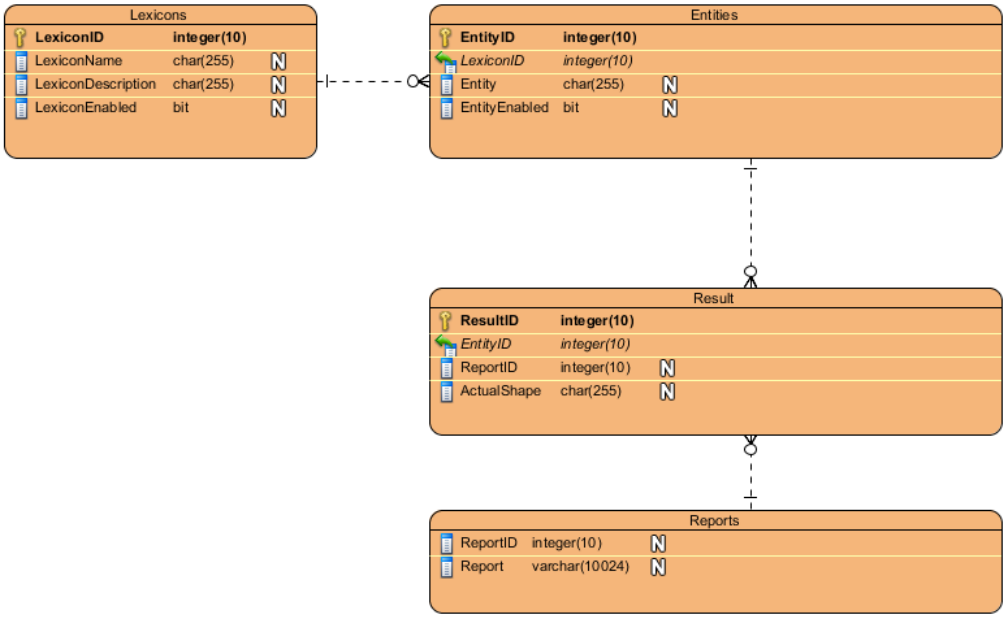
- EntityID - číslo entity (primary key)
- LexiconName - název pojmové skupiny kam výraz patří
- Entity - výraz v základním gramatickém tvaru
- EntityEnabled - Ano/Ne - vypínač výrazu - pro případ, že chce uživatel hledat výsledky jen pro některé výrazy

4.1.3 Výstup

Výstupem programu je databázový sqlite soubor. Obsahuje všechna data, která jsou potřeba pro postprocesing. Skládá se ze čtyř tabulek:

- **Reports** - očíslované vstupní zprávy
- **Entities** - vyhledávané pojmy
- **Lexicons** - seznam pojmových slovníků
- **Results** - výsledková tabulka - obsahuje odkazy na zprávy a nalezené entity

Tabulky Lexicons a Entities jsou spojeny vaznou 1:N, tabulky Entities a Reports jsou spojeny přes vazební tabulku Results vazbou M:N. V tabulce Results je přidán sloupec ActualShape, kde je uložen skutečný gramatický tvar nalezeného výrazu.



Obrázek 4.3: Struktura databáze výsledků

4.1.4 Pomocná data

Pro chod programu jsou třeba ještě dva pomocné soubory:

1. **config.xml** - konfigurační soubor, ukládá se zde nastavení programu a naposledy použité cesty k souborům
2. **vocabulary.db** - stěžejní soubor pro lemmatizaci. Sqlite tabulka obsahuje všechny tvary českých slov, celkem 2 176 498 položek. Struktura:
   - id - číslo položky (primary key)
   - word - slovo v některém z gramatických tvarů
   - base_shapes - slovo v základním tvaru, pokud připadá na jeden gramatický tvar více základních tvarů, jsou odděleny znakem '|', například aparátníkovi->aparátník|aparátníkův

4.2 Algoritmus

1. **Příprava keše s gramatickými tvary vyhledávaných slov.**
   Z důvodu rychlosti vytvoříme paměťovou mapu se všemi gramatickými tvary slov z pojmových slovníků. Klíčem mapy je gramatický tvar a hodnotou EntityID z tabulky Entities. Vytvoření mapy trvá jistou dobu, řádově 10-20 sekund. Při ukončení programu se mapa uloží pomocí serializace do souboru **lexicons.map**. Při příštím otevření programu se načítá z tohoto souboru což je již rychlá operace. Vytváří se znovu při jakékoliv změně v Pojmových slovnících.

2. **Vytvoření výstupního databázového souboru**

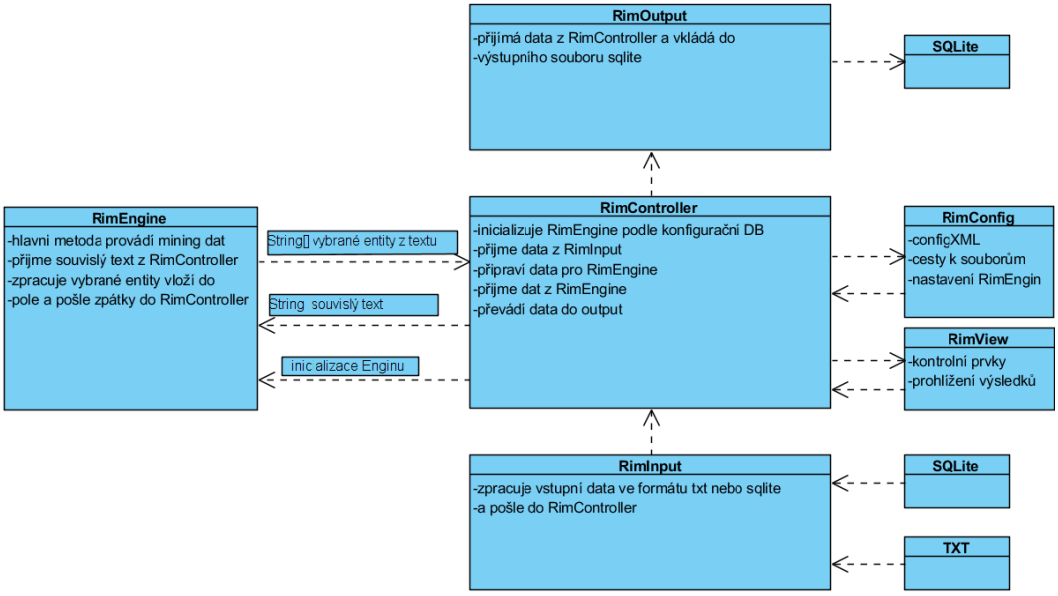
   - Jako základ se zkopíruje vstupní databázová tabulka se zprávami.
   - Přidají se tabulky Lexicons a Entities - pojmové slovníky.
   - Vytvoří se struktura tabulky Results - sem se budou ukládat výsledky vyhledávání.

3. **Hlavní smyčka přes jednotlivé zprávy**
   Procházejí se jednotlivé zprávy:

   - Rozdělí se na věty (pomocí knihovny OpenNLP).
   - Věta se rozdělí na slova.
   - Každé slovo se vyhledá v mapě gramatických tvarů výrazů z pojmových slovníků. Pokud je nalezeno, založí se nová položka v tabulce Results, vloží se rovněž aktuální tvar slova.

Jednotlivé moduly programu (**RimInput**, **RimOutput**, **RimEngine**) jsou relativně samostatné části, které mohou být dále rozšířeny. Hlavně vstup a výstup je možno na základě požadavků upravit na další formáty aniž by se muselo něco měnit na výkonné jednotce RimEngine.

V celém projektu využívám databázi SQLite. Je pro to několik důvodů. Je to souborový databázový systém, jedna databáze = jeden soubor. Není třeba spouštět žádný server - stačí knihovní funkce Javy. Javovská knihovna pro práci s SQLite databází je k dispozici například tady [3]. Pro práci s SQLite je vytvořen wrapper **MySQLiteTool**. Zdrojový kód je v příloze.



Obrázek 4.4: Schema programu

4.3 Vytěžování dat

Po skončení procesu vyhledávání je možno přistoupit k vytěžování dat z vytvořené databáze. Výsledky jsou uloženy v tabulce Results, která má následující strukturu:

- **ID** - pořadové číslo (primery key)
- **EntityID** - číslo výrazu z tabulky Entities
- **ReportID** - číslo zprávy z tabulky Reports
- **ActualShape** - gramatický tvar výrazu nalezený ve zprávě

4.3.1 Přehled zpráv s nalezenými výrazy

Základní výstupní tabulka s přehledem zpráv a identifikací nalezených výrazů. Tabulka má následující sloupce:

- **Zpráva č.** - pořadové číslo zprávy (odpovídá číslu v tabulce Reports)

- **Potíže** - počet nalezených výrazů ze skupiny Potíže
- **Lokalizace** - počet nalezených výrazů ze skupiny Lokalizace
- **Léčba** - počet nalezených výrazů ze skupiny Léčba
- **Léky** - počet nalezených výrazů ze skupiny Léky
- **Zpráva** - text zprávy s barevně vyznačenými nalezenými výrazy - je zobrazeno napravo od tabulky pro každý aktivní řádek

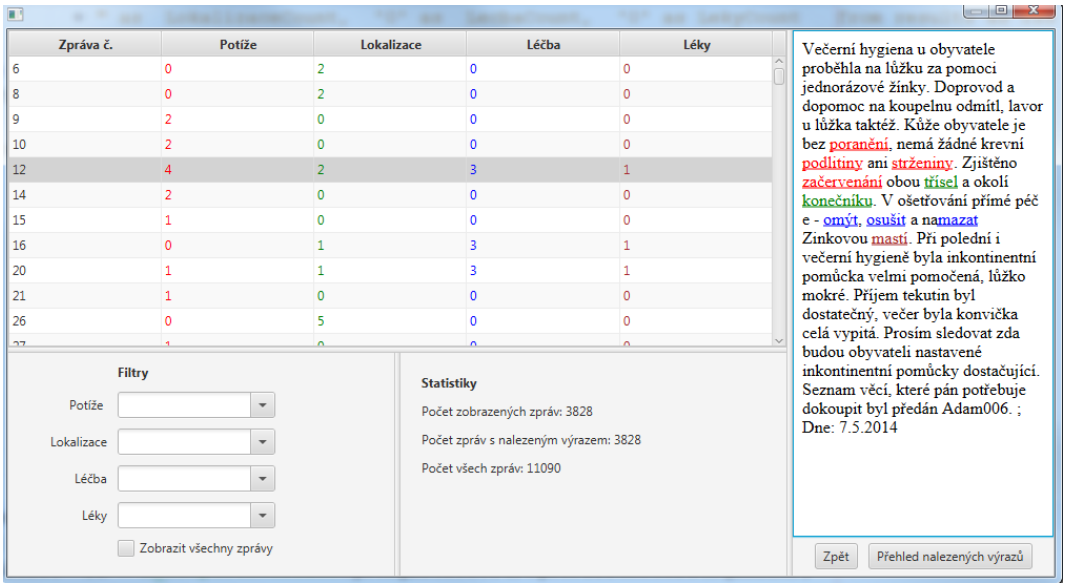
Pro získání těchto dat byl použit sql příkaz:

```
select  ReportId as rid, report, ActualShape,
        PotizeCount, LokalizaceCount,  LecbaCount,
        LekyCount from
        (select r.ReportId, rp.report, r.ActualShape,
        e.LexiconName,
        sum(e.lexiconname = 'POTIZE') as PotizeCount,
        sum(e.lexiconname = 'LOKALIZACE') as
                LokalizaceCount,
        sum(e.lexiconname = 'LECBA') as LecbaCount,
        sum(e.lexiconname = 'LEKY') as LekyCount
        from results as r inner join entities as e
                on e.Entityid = r.entityid
                        inner join reports as rp
                        on r.ReportID = rp.ReportID
        group by r.reportid ) as a
```

K tabulce lze ještě přiřadit filtr, pokud se mají zobrazit jen zprávy, které obsahují určité výrazy. Postup je popsán v dodatcích, v uživatelském popisu programu.

4.3.2 Přehled nalezených výrazů v textu

Jiným způsobem jak se podívat na data, je seznam nalezených výrazů podle frekvence výskytu a zobrazit jim odpovídající zprávy. Taková tabulka má následující strukturu:



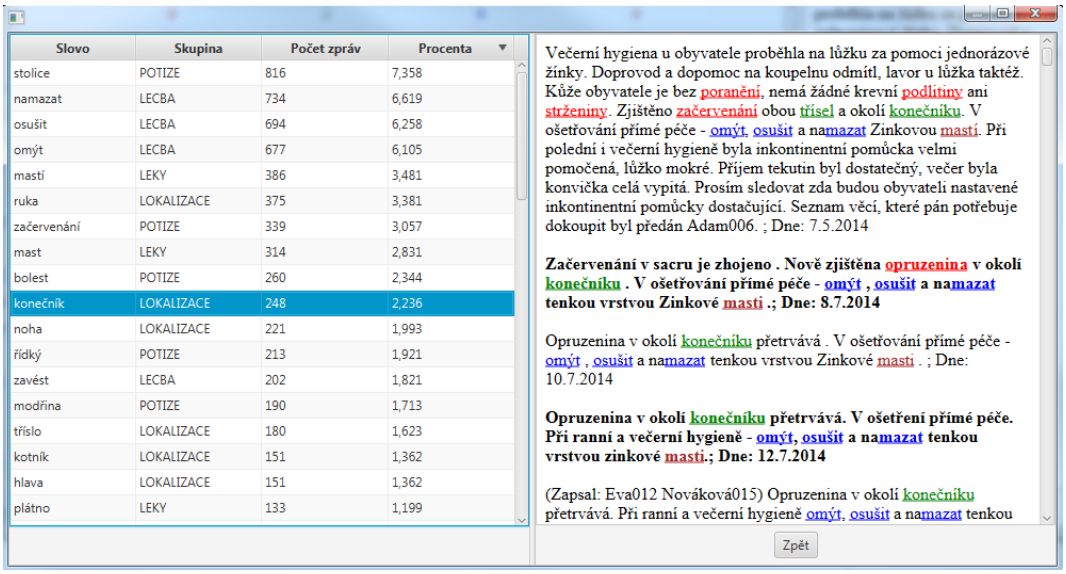
Obrázek 4.5: Přehled zpráv

- **Slovo** - výraz, nalezený v textu
- **Skupina** - pojmová skupina, do které výraz patří
- **Počet zpráv** - počet výskytů výrazu ve všech zprávách
- **Procenta** - procentuálně vyjádřený počet výskytů
- **Zprávy** - všechny zprávy, ve kterých se výraz nachází. Text zpráv je s barevně vyznačenými nalezenými výrazy - vše je zobrazeno napravo od tabulky pro každý aktivní řádek.

Pro získání těchto dat byl použit sql příkaz:

```
select e.entity, e.entityid, e.lexiconname, cnt from
(select  entityid , count(entityid) as cnt
        from results group by (entityid)) as s1
        inner join entities e
                on s1.entityid = e.entityid
order by cnt desc
```

Sloupec s procenty je programově dopočítán z celkového počtu zpráv a aktuálního počtu výskytů.



Obrázek 4.6: Přehled nalezených výrazů

4.4 Funkčnost programu

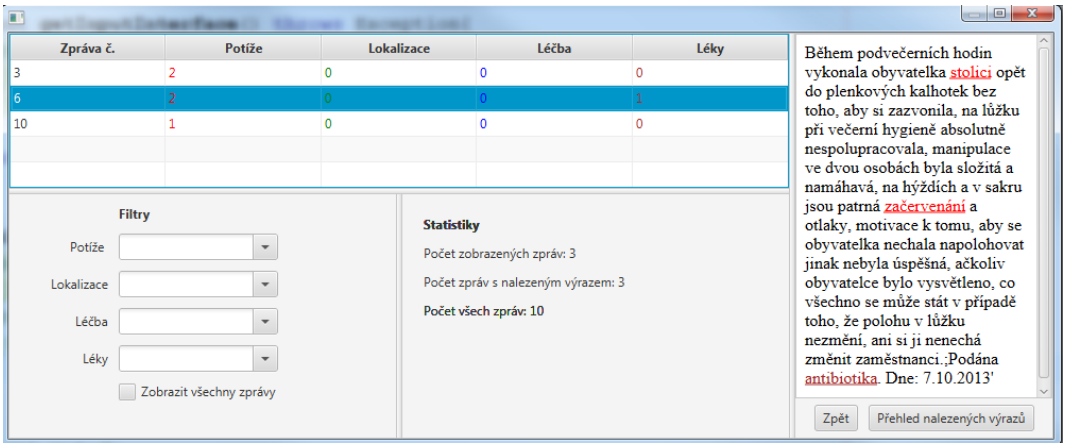
4.4.1 Testovací příklad

Ukážeme si funkčnost programu na malém testovacím příkladu. Vstupem bude obecný textový formát. Všude budeme předpokládat formátování UTF-8.
Mějme následující text: (jde o autentický zdravotní záznam z Domu seniorů, který prošel anonymizací)
Paní Nováková020 odmítla oběd z důvodu nevolnosti. Podány piškoty a čaj.; Dne: 7.10.2013"
Postřeh přímé péče - z důvodu zhoršení mobility, bude u obyvatelky RHB vyzkoušen přesun zvedákem s područkama.; Dne: 7.10.2013"
Při odpolední výměně inkontinentních pomůcek měla obyvatelka 1x řídkou stolici, SZP informována.; Dne: 7.10.2013"
Mluvil jsem s obyvatelkou. V současné době se cítí slabá, ale sama mluvila, že by opět ráda chodila. Bavili jsme se o možnosti instalace závěsu - to razantně odmítla. Uvedla, že se s prvním rozhodnutím unáhlila, pak až jí došlo, že by jí především chybělo madlo, které má nyní na dveřích. Využívá ho při vstávání z WC, při chůzi v chodbičce. To je hlavní důvod odmítnutí. Souhlasila s návrhem, že pokud bychom nějak vyřešili madla a možnost přichycení, závěs

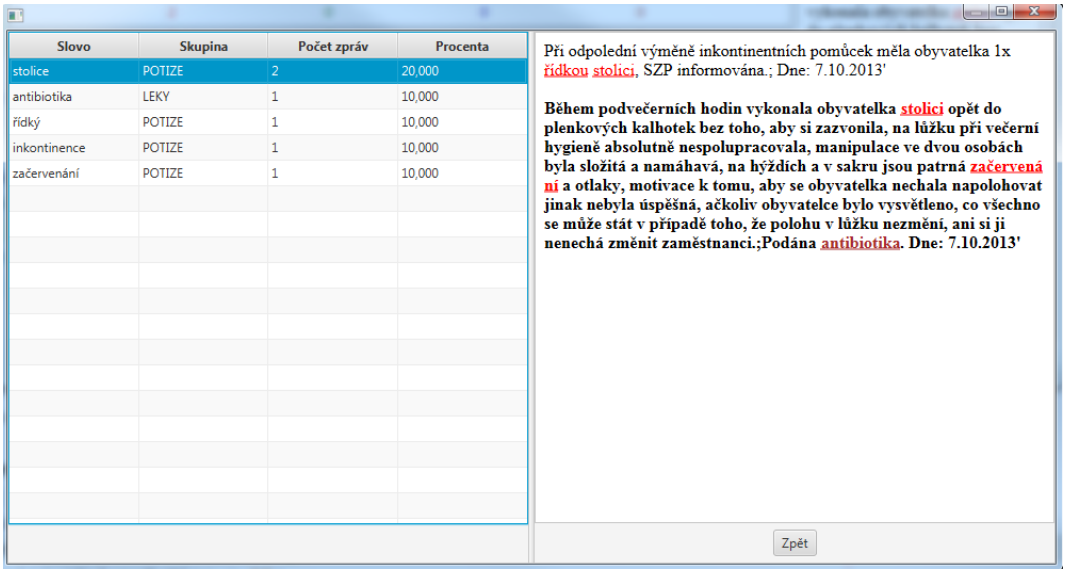
by akceptovala. Obyvatelka mi děkovala mi za návštěvu a rozovor (i na jiná témata), a že hledáme možnosti a řešení jejího docházení na WC.
Přijedou v pátek odpoledne na návštěvu, zůstanou do soboty. Zdá se mu, že obyvatelka ""chřadne"". Myslí si, že je to především z důvodu omezení nabídky aktivit, že je maminka odevzdanější, pasivnější a chybí jí ""elán""a motivace. Během podvečerních hodin vykonala obyvatelka stolici opět do plenkových kalhotek bez toho, aby si zazvonila, na lůžku při večerní hygieně absolutně nespolupracovala, manipulace ve dvou osobách byla složitá a namáhavá, na hýždích a v sakru jsou patrná začervenání a otlaky, motivace k tomu, aby se obyvatelka nechala napolohovat jinak nebyla úspěšná, ačkoliv obyvatelce bylo vysvětleno, co všechno se může stát v případě toho, že polohu v lůžku nezmění, ani si ji nenechá změnit zaměstnanci.;Podána antibiotika. Dne: 7.10.2013"
Obyvatelka byla dnes vysazena do invalidního vozíku a dána na chodbu, kde si povídala s ostatními obyvateli a byla spokojená. Na vozík se obyvatelka přesunula sama jen s mírnou dopomocí. ; Dne: 8.10.2013"
Obyvatelka byla po celý den spavá a téměř nepila a nejedla. Prosím, obyvatelka pravidelně a aktivně nabízet tekutiny a kontrolavat, zda-li jí, popřípadně dopomoct u jídla.; Dne: 9.10.2013"
(Zapsal: Eva019 Nováková022) Obyvatelka z rána nesnědla nic. Při podávání oběda jídlo odmítla. Jako důvod uvedla, že jíst nebude, at jí nic nenutíme.; Dne: 9.10.2013"
Obyvatelka při noční a ranní kontrole komunikovala bez problémů, orientovaná. Při polohování a výměně inkontinence spolupracovala. Nabízené tekutiny neodmítala, pila s lahvičky sama, bez problémů. Signalizaci v noci nepoužila.; Dne: 12.10.2013"

Program vezme jako jednu jednotku zprávy jeden odstavec, který je ukončený znakem konce řádku. Pro ni vytvoří záznam v databázi. Pokud pouštíme program poprve, případně po editaci pojmových slovníků, vytváří se mapa výrazů. Tato operace trvá řádově desítky sekund a je samozřejmě nezávislá na velikosti zpracovávaného souboru. Po vytvoření mapy se provede samotné vyhledávání, které na takto malém souboru trvá do tří sekund.

Byly nalezeny tři zprávy, které obsahují aspoň jeden výraz z pojmových slovníků. Na další tabulce je vidět, že bylo nalezeno celkem pět výrazů.



Obrázek 4.7: Přehled zpráv s nalezeným výrazem



Obrázek 4.8: Přehled nalezených výrazů

4.4.2 Výkonnost programu

Rychlost:
Program byl vyzkoušen na anonymizovaných datech z Domu seniorů. Původně měl záznam 11090 položek zápisů v excelovském souboru. Byl převeden do sqlite databázového souboru SPS_anonymBIG.db. Pokud je již vytvořena mapa hledaných výrazů, trvá prohledání několik vteřin. Pokud stejná vstupní data převedeme do textového formátu, doba zpracování se prakticky nezmění. Menší zdržení při práci s programem nastává pokud v prohlížecí tabulce zpráv zadáme filtr na nějaký výraz. Vykonání složitější sql query pak trvá až do deseti sekund. Největší zdržení tak nastává při tvorbě kešovací mapy hledaných výrazů. Je to dáno velikostí databázové tabulky vocabulary.db s miliony slov.

Přesnost:
Program najde všechny entity, které jsme zadali v konfigurační databázi pojmových slovníků. Případné nepřesnosti mohou nastat, pokud gramatický tvar je společný pro více základních tvarů: např. mastí -> mast, mastit. Ve vstupním textu mohou být samozřejmě překlepy, tyto program explicitně neřeší. Případné časté nepřesnosti mohou být zahrnuty do pojmového slovníku.

5 Závěr

Výsledkem bakalářské práce jsou dva programy pro anonymizaci a strukturalizaci textu. Původní podnět k bakalářské práci přišel z praxe, konkrétně z jednoho Domova pro seniory. Byl jsem v kontaktu s panem ředitelem a on mi popsal jak by si představoval program, který by mu pomohl automatizovat sledování péče o seniory. Měl konkrétní požadavky, co by se mělo zahrnout do vyhledávání. Ale hlavně chtěl, aby měl možnost program konfigurovat a volit, co sledovat a jak výsledky strukturovat. Bohužel pan ředitel byl brzy po zahájení prací na bakalářské práci odvolán. Nové vedení jsem už nekontaktoval. Ale myslím, že programy, které jsou součástí této práce by vyhověly původním požadavkům.

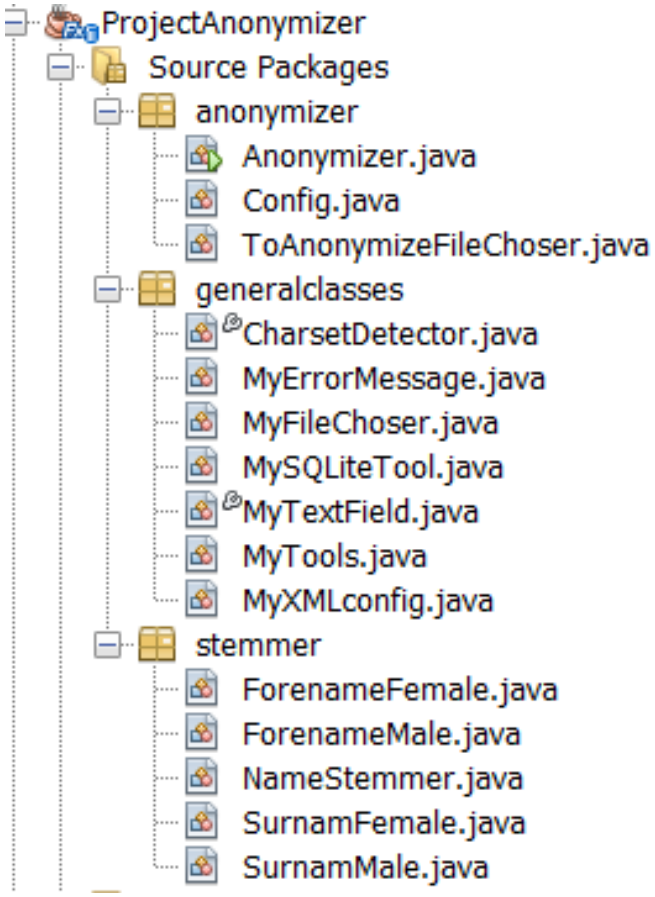
Programy se dají použít i mimo oblast zdravotnictví. Změnou pojmových slovníků lze vyhledávat entity i z jiných oblastí. Program **Anonymizer** je úplně obecný a nepředpokládá předem žádný charakter textu. Pracuje i s formátovanými textovými soubory, např. xml.

Na obou programech je možný další vývoj, u **Anonymizeru** by další zlepšení určitě spočívalo ve skloňování anonymních náhrad podle původního textu. U programu **ReportsInfoMiner** je možné přidávat další vyhledávací sql query příkazy, které by vytěžily další informace. To už by vyplývalo z požadavků praxe.

A Programová dokumentace

Oba programy jsou psány v jazyku Javafx, proto je třeba mít nainstalovánu Javu verze 8 a novější.

A.1 Program Anonymizer



Obrázek A.1: Struktura programu Anonymizer

Struktura programu je na Obr. A.1. Skladá se ze tří balíků:

anonymizer	- hlavní třídy Anonymizeru
generalclasses	- pomocné utility
stemmer	- práce se stemmizační databází

Zdrojový kód hlavní metody třídy Anonymizer pro anonymizaci textu:

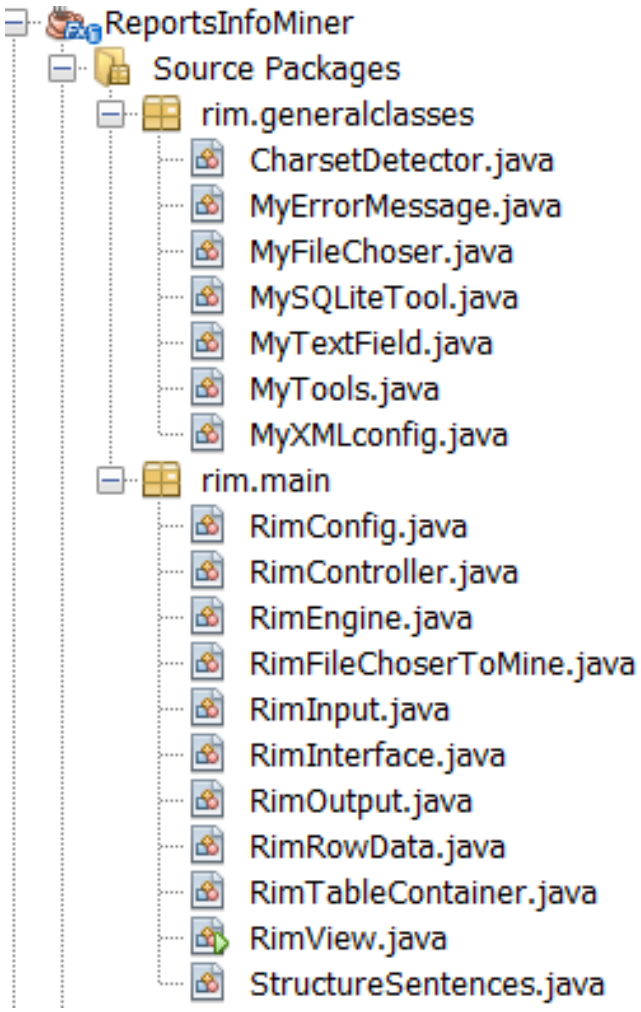
```
//#############################################################################
/**
 * hlavni metoda pro anonymizaci
 * 1. otevre soubor pro anonymizaci
 * 2. rozlozi obsah na jednotlive vety
 * 3. rozlozi vety na jednotliva slova
 * 4. ve slovech najde jmena a prijmeni
 * 5. zjisti, zda jde skutecne o jmena a prijmnei (pomoci
    slovniku korekce)
 * 6. nahradi jmena a prijmeni anonymnimi nahradami
 * 7. vyhleda dalsi kandidaty na nahrazeni (podle upper case)
 * 8. vlozi do slovniku skutecne jmeno (ve vsech vyskytujicich se
    tvarech) a jeho nahradu
 * @throws Exception
 */
void anonymize() throws Exception{

  // nacteni cest k souborum
  updateDataFromView();

  //aktualizuje seznam kandidatu pro nahrazeni - prida ty
     odsouhlasene v minulem behu
  updateCandidates();

  //nacte seznam jmen, ktera se oznacila v minulem behu jako
     nejmena
  getTableWrong();

  //zalozime nove slovniky, kde bude seznam nahrad - vytvareji se
     pri kazdem behu znovu
  m_DictionaryFornameMale = new HashMap<>();
  m_DictionaryFornameFemale = new HashMap<>();
  m_DictionarySurnameMale = new HashMap<>();
  m_DictionarySurnameFemale = new HashMap<>();

  try (
```

```
// soubor pro ulozeni vysledku
BufferedWriter bw = m_Config.getStoreWriter()) {
if (bw == null){
  // vystupni soubor neni v poradku
  return;
}

// rozklad vstupniho souboru na vety
String[] lines =
    MyTools.getLines(m_Config.m_cFileToAnonymize,
    m_Config.m_Charset);

// objekty pro nahrazeni jmen a prijmeni ve vete
SentenceObject soSurnamesMale = new
    SentenceObject(NameType.SURNAME_MALE,
    m_DictionarySurnameMale);
SentenceObject soSurnamesFemale = new
    SentenceObject(NameType.SURNAME_FEMALE,
    m_DictionarySurnameFemale);
SentenceObject soFornamesMale = new
    SentenceObject(NameType.FORENAME_MALE,
    m_DictionaryFornameMale);
SentenceObject soFornamesFemale = new
    SentenceObject(NameType.FORENAME_FEMALE,
    m_DictionaryFornameFemale);

/* cyklus pres vsechny vety */
for (String line : lines){
  // rozklad na slova
  String tokens[] = MyTools.tokenizeSentence(line);
  // vyndame pocatecni a koncove znaky, ktere nejsou
     pravdepodobne soucasti jmen
  tokens = washTokens(tokens);
  // nalezeni kandidatu na jmena a prijmeni
  String[] forenamesFemale = m_Stemmer.digNames(tokens,
     NameType.FORENAME_FEMALE);
  tokens = removeDuplicates(tokens, forenamesFemale);
  String[] forenamesMale = m_Stemmer.digNames(tokens,
     NameType.FORENAME_MALE);
  tokens = removeDuplicates(tokens, forenamesMale);
  String[] surnamesMale = m_Stemmer.digNames(tokens,
     NameType.SURNAME_MALE);
  tokens = removeDuplicates(tokens, surnamesMale);
```

```
String[] surnamesFemale = m_Stemmer.digNames(tokens,
    NameType.SURNAME_FEMALE);
tokens = removeDuplicates(tokens, surnamesFemale);
// v tokens uz zbyly jen "nonamy" - ulozime je pro pripadne
    odsouhlaseni uzivatelem
addFirstCapitalNonames(tokens);
//vyhodime jmena, ktera uz jsou zahrnuta v jinem seznamu
    jmen
forenamesMale = removeDuplicates(forenamesMale,
    forenamesFemale);
surnamesMale = removeDuplicates(surnamesMale,
    forenamesMale);
surnamesMale = removeDuplicates(surnamesMale,
    forenamesFemale);
surnamesFemale = removeDuplicates(surnamesFemale,
    forenamesMale);
surnamesFemale = removeDuplicates(surnamesFemale,
    forenamesFemale);
surnamesFemale = removeDuplicates(surnamesFemale,
    surnamesMale);

// pridame slova, ktera byla drive oznacena jako jmena
List<String[]> ret = addFromCandidates(tokens,
    surnamesMale, surnamesFemale);
surnamesMale = ret.get(0);
surnamesFemale = ret.get(1);

// kontrola spravnosti nalezenych entit podle slovniku
    korekce - vyhodime "nejmena"
forenamesFemale =
    correctListToReplaceFromCorrection(forenamesFemale,
    m_CorrectionForenames);
forenamesMale =
    correctListToReplaceFromCorrection(forenamesMale,
    m_CorrectionForenames);
surnamesFemale =
    correctListToReplaceFromCorrection(surnamesFemale,
    m_CorrectionSurnames);
surnamesMale =
    correctListToReplaceFromCorrection(surnamesMale,
    m_CorrectionSurnames);

// nahrazeni pravych jmen anonymy a pridani do slovniku
```

```
        line = soFornamesMale.replace(line, forenamesMale);
        line = soFornamesFemale.replace(line, forenamesFemale);
        line = soSurnamesMale.replace(line, surnamesMale);
        line = soSurnamesFemale.replace(line, surnamesFemale);
        //zapis zmenene vety do vystupniho souboru
        bw.write(line);
        bw.newLine();
      }
    }
    setTables();
  }
}
```

A.2 Program ReportsInfoMiner



Obrázek A.2: Struktura programu ReportsInfoMiner

Struktura programu je na Obr. A.2. Skladá se ze tří balíků:

rim.main	- hlavní třídy programu
rim.generalclasses	- pomocné utility
rim.sqlbrowsers	- prohlížení databázových tabulek

A.3 Wrapper pro práci s SQLite databází

Zdrojový kód třídy pro práci s SQLite DB.

MySQLiteTool:

```
/**
 * wrapper pro knihovnu databaze sqlite
 *
 * @author A12B0064K
 */
public class MySQLiteTool {

  // aktualni spojeni s databzi
  Connection m_Connection;

  // aktualni tabulka databaze, se kterou se pracuje
  String m_cActiveTable;

  // resultset napriklad pro vysledek sql dotazu
  ResultSet m_ResultSet;

  // seznam jmen sloupcu aktualni tabulky
  List<String> m_ColumnsName;

  /**
   * vytvori aktualni spojeni na databazi v souboru cDBfile
   * zaroven ho dosadi do promenne m_Connection
   * @param cDBfile - soubor s SQlite databazi
   * @return aktualni spojeni
   * @throws Exception
   */
  public Connection createConnectionToSQLite(String cDBfile) throws
      Exception{
    Class.forName("org.sqlite.JDBC");
    m_Connection =
        DriverManager.getConnection("jdbc:sqlite:"+cDBfile);
    return m_Connection;
  }

  /**
```

```
 * vraci aktualni spojeni
 * @return
 */
public Connection getConnection(){
  return m_Connection;
}

/**
 * nastavuje aktualni tabulku v databazi
 * @param cTable
 */
public void setActiveTable(String cTable){
  m_cActiveTable = cTable;
}

/**
 * ze souboru s formatem csv vytvoru SQlite databazi
 * @param cCSVinput - vstupni csv soubor
 * @param cSQLiteOutput - vystupni sqlite soubor s databazi
 */
public static void createSQLiteFromCSV(String cCSVinput, String
    cSQLiteOutput) throws Exception{
  final String SEPARATOR = "#@#";

  String Charset = MyTools.getCharset(cCSVinput);
  if (Charset == null){
    throw new RuntimeException("File "+cCSVinput+" is
        unreadable.");
  }

  File out = new File(cSQLiteOutput);
  if (out.exists()){
    out.delete();
  }

  MySQLiteTool sq = new MySQLiteTool();
  sq.createConnectionToSQLite(cSQLiteOutput);
  sq.getConnection().setAutoCommit(false);

  String[] lines = MyTools.getLines(cCSVinput, Charset);
  if (lines == null || lines.length==0){
    throw new RuntimeException("File "+cCSVinput+" is empty.");
  }
```

```
// nejdriv vytvorime strukturu tabulky
String[] cHeader = lines[0].split(SEPARATOR);
int iPocetSloupcu = cHeader.length;
if (iPocetSloupcu == 0){
  throw new RuntimeException("File "+cCSVinput+" has no
      columns.");
}

String[] cDataType = new String[iPocetSloupcu];
for(int i=0; i<iPocetSloupcu; i++){
  cDataType[i] = "TEXT";
}
cDataType[0] = "INT";
String cTableName = MyTools.getFileName(cSQLiteOutput);
sq.createSQliteTable(cTableName, cHeader,cDataType);

// naplnime tabulku v db daty
String header = lines[0].replace(SEPARATOR, ",");
int iLine = 0;
for (int i=1; i<lines.length; i++){
  iLine++;
  String line = lines[i];
  Statement stmt = sq.getConnection().createStatement();
  //String[] splitLine = MyTools.separateString(line, ",",
      iPocetSloupcu);
  line = "'"+line.replace(SEPARATOR,"','")+"'";
  String sql = "INSERT INTO "+cTableName+" ("+header+") " +
               "VALUES ("+String.valueOf(iLine)+','+line+");";
  try{
    stmt.executeUpdate(sql);
  }
  catch(Exception e){
    System.out.print(sql+"\n");
  }
  stmt.close();
}
sq.getConnection().commit();


sq.getConnection().close();
sq.close();
```

```
}


/**
 * vytvori sqlite tabulku
 *
 * @param cName - nazev tabulky
 * @param cStructure - nazvy jednotlivych sloupcu tabulky
 * @param cDataType - typy promennych ve sloupcich
 * @throws Exception
 */
public void createSQliteTable(String cName, String[] cStructure,
    String[] cDataType) throws Exception {
  if (m_Connection == null || cStructure.length == 0) return;
  Statement stmt = m_Connection.createStatement();
  if (cName == null)
    cName = m_cActiveTable;

  String sql = "CREATE TABLE " + cName +
              " (____ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,";
  for (int i=0; i<cStructure.length-1; i++){
    sql += cStructure[i] + " "+cDataType[i]+",";
  }
  sql += cStructure[cStructure.length-1] + "
      "+cDataType[cStructure.length-1]+")";
  try{
    stmt.executeUpdate(sql);
  }catch (Exception e){
    throw new RuntimeException(e);
  }
  stmt.close();
}

/**
 * vraci prvni record odpovidajici SQL prikazu
 * @param sql SQL prikaz
 * @return resultset, pokud zadny nenalezne, vraci null
 * @throws Exception
 */
public ResultSet getFirstRecord(String sql) throws Exception {
  Statement stmt = getConnection().createStatement();
  ResultSet rs = stmt.executeQuery(sql);
  if (rs.next()){
```

```
    return rs;
  }
  return null;
}

/**
 * vraci nasledujici record odpovidajici SQL prikazu
 * @param rs predchozi resultset
 * @return resultset, pokud zadny nenalezne, vraci null
 * @throws Exception
 */
public ResultSet getNextRecord(ResultSet rs) throws Exception {
  if (rs.next()){
    return rs;
  }
  return null;
}

/**
 * uzavre spojeni s databazi
 * @throws Exception
 */
void close() throws Exception{
  if (m_Connection != null)
    m_Connection.close();
}

/**
 * vraci seznam nazvu sloupcu tabulky
 * @param cTable nazev tabulky, pro kterou chceme seznam sloupcu,
 * pokud je null, vezme se aktualni tabulka
 * @return seznam nazvu sloupcu tabulky
 * @throws Exception
 */
List getColumnsName(String cTable)throws Exception {
  if (cTable == null)
    cTable = m_cActiveTable;
  DatabaseMetaData meta = getConnection().getMetaData();
  ResultSet res = meta.getColumns(null, null, cTable, null);
  List ret = null;
  while (res.next()) {
    if (ret == null)
      ret = new ArrayList<String>();
```

```
    ret.add(res.getString("COLUMN_NAME"));
  }
  return ret;
}

/**
 * vraci prvni radek tabulky
 * je ulozen v HashMap, kde key je nazev sloupce a value je
    hodnota ve sloupci
 * @return prvni radek tabulky
 * @throws Exception
 */
public LinkedHashMap<String, String> getFirstRow()throws
    Exception{
  String sql = "SELECT * FROM '"+ m_cActiveTable+ "'";
  m_ResultSet = getFirstRecord(sql);
  m_ColumnsName = getColumnsName(null);
  return getRow();
}

/**
 * vraci dalsi radek tabulky
 * je ulozen v HashMap, kde key je nazev sloupce a value je
    hodnota ve sloupci
 * @return dasli radek tabulky
 * @throws Exception
 */
public LinkedHashMap<String, String> getNextRow()throws Exception{
  if (m_ResultSet == null)
    return null;
  m_ResultSet = getNextRecord(m_ResultSet);
  return getRow();
}

private LinkedHashMap<String, String> getRow()throws Exception{
  LinkedHashMap<String, String> RowData = new
      LinkedHashMap<String, String>();
  if (m_ResultSet != null){
    for (String cColumn : m_ColumnsName){
      RowData.put(cColumn, m_ResultSet.getString(cColumn));
    }
    return RowData;
  }
```

```
    return null;
  }

  /**
   * vraci radek row, ktery ma polozky oddelene carkami
   * @param row
   * @return
   */
  private String[] getValues(Map<String, String> row){
    String[] ret = {"","",};
    for (Iterator<String> it = row.keySet().iterator();
        it.hasNext();){
      String cKey = it.next();
      ret[0] += (ret[0].isEmpty()?"":",") + cKey;
      ret[1] += (ret[1].isEmpty()?"'":"','") + row.get(cKey);
    }
    if (!ret[1].isEmpty())
      ret[1] += "'";
    return ret;
  }

  /**
   * vlozi do aktualni tabulky radek s daty
   * @param Row data radku
   * @param iRow cislo radku - pouziva se jako primary key pro
       sloupec ____ID
   * @throws Exception
   */
  public void insertRow(Map<String, String> Row, int iRow) throws
      Exception {
    String[] vl = getValues(Row);
    String sql = "INSERT INTO "+m_cActiveTable+" (____ID,"+vl[0]+")
        " +
                "VALUES ("+String.valueOf(iRow)+","+vl[1]+");";
    Statement stmt = m_Connection.createStatement();
    try{
      stmt.executeUpdate(sql);
    }catch(Exception e){
      throw new RuntimeException(e);
    }}

}
```

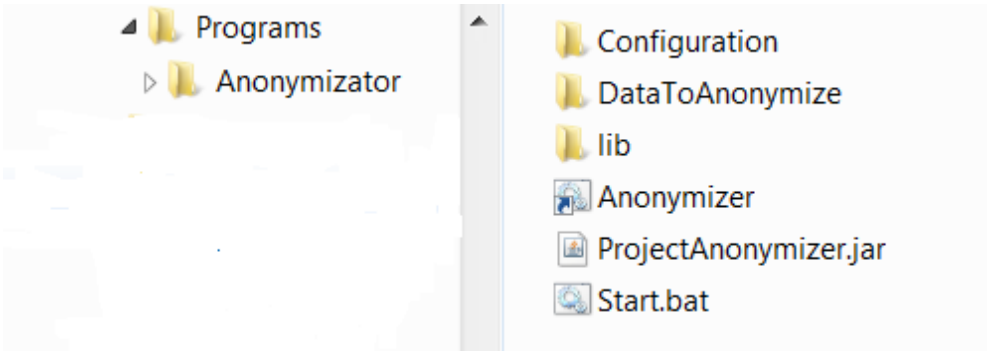
B Uživatelská dokumentace

B.1 Softwarové požadavky

Oba programy vyžadují nainstalovanou Javu verze 8 a novější. Program je odzkoušen na Windows 7 ale měl by fungovat i na vyšších verzích.

B.2 Program Anonymizer

Adresářová struktura programu je na Obr. B.1:

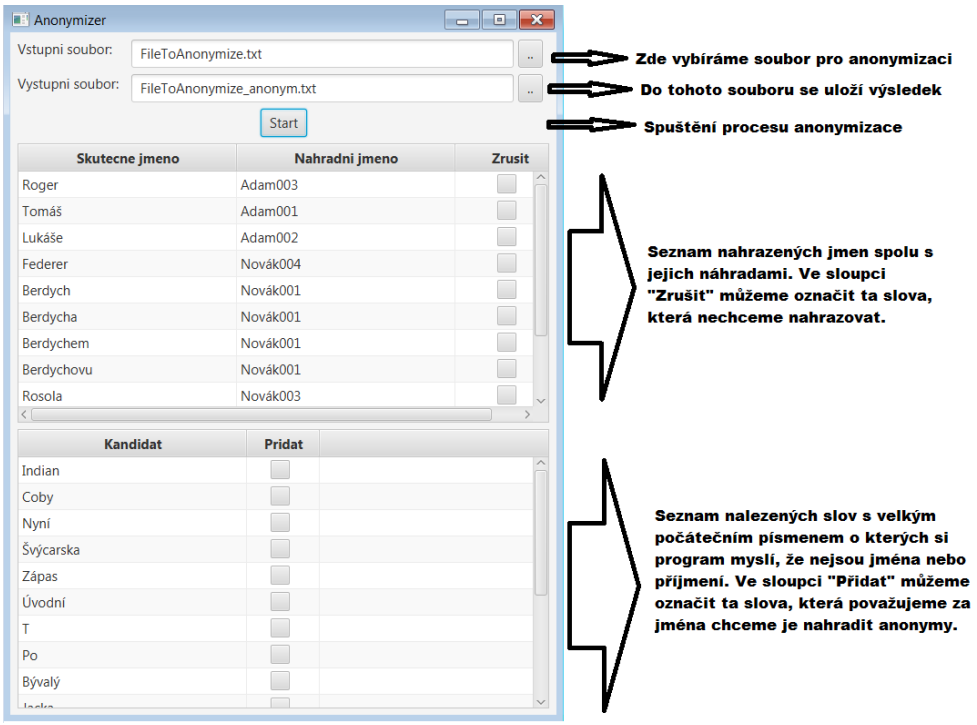


Obrázek B.1: Adresář programu Anonymizer

Popis adresářů:

Configuration	- konfigurační soubory programu
DataToAnonymize	- testovací data v souboru FileToAnonymize.txt
lib	- knihovny projektu

Program se spouští dávkovým souborem start.bat
Ovládací dialog je na Obr. B.2. Ovládání programu je jednoduché a intuitivní. Důležité je, že pokud nějaké slovo označíme za ”nejméno”, nebude se už příště nahrazovat. Program se tak vlastně ”učí”.



Obrázek B.2: Program Anonymizer

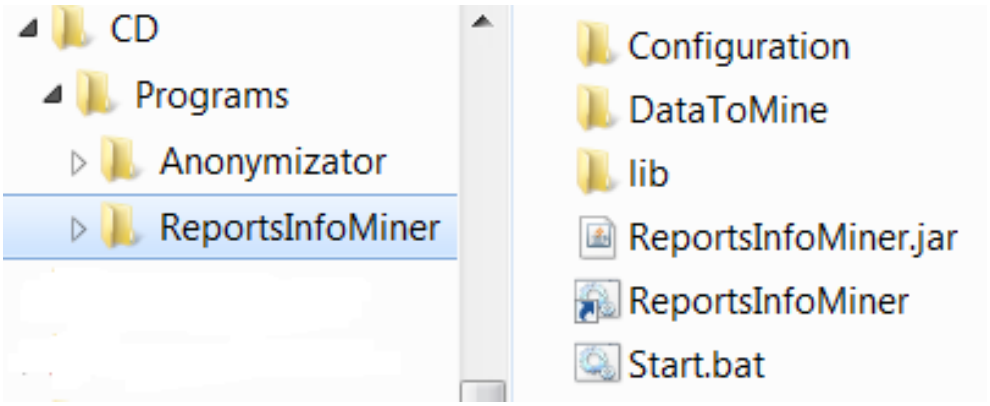
B.3 Program ReportsInfoMiner

Adresářová struktura programu je na Obr. B.3:

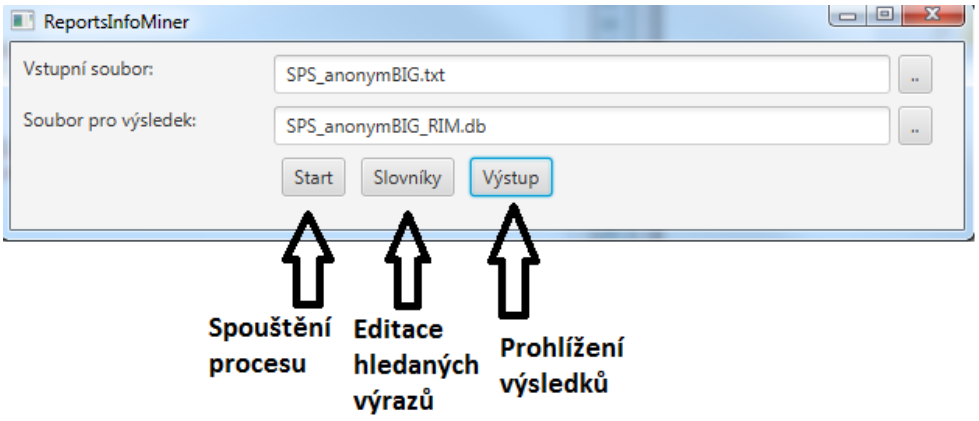
Popis adresářů:

Configuration	- konfigurační soubory programu
DataToMine	- testovací data
lib	- knihovny projektu

Program se spouští dávkovým souborem start.bat
Ovládací dialog je na Obr. B.4.

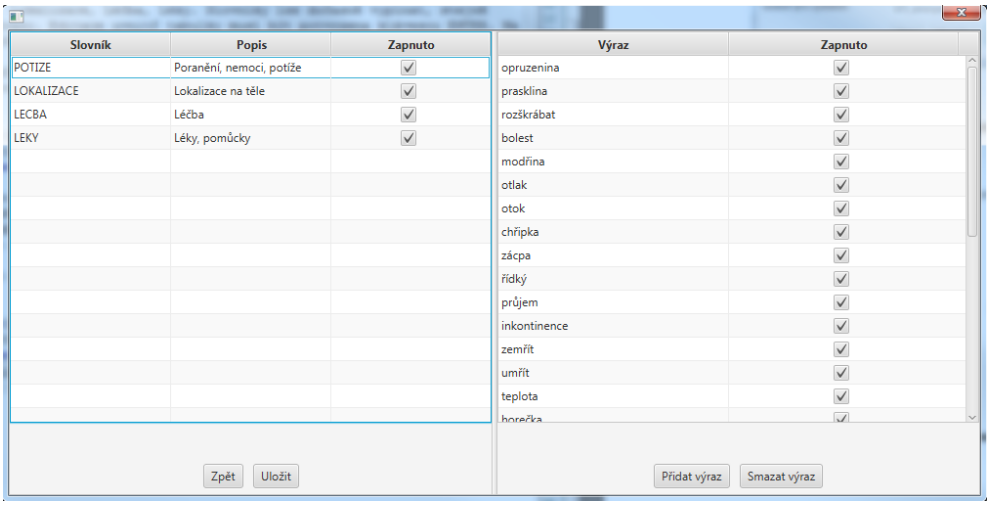


Obrázek B.3: Adresář programu ReportsInfoMiner



Obrázek B.4: Úvodní obrazovka programu ReportsInfoMiner

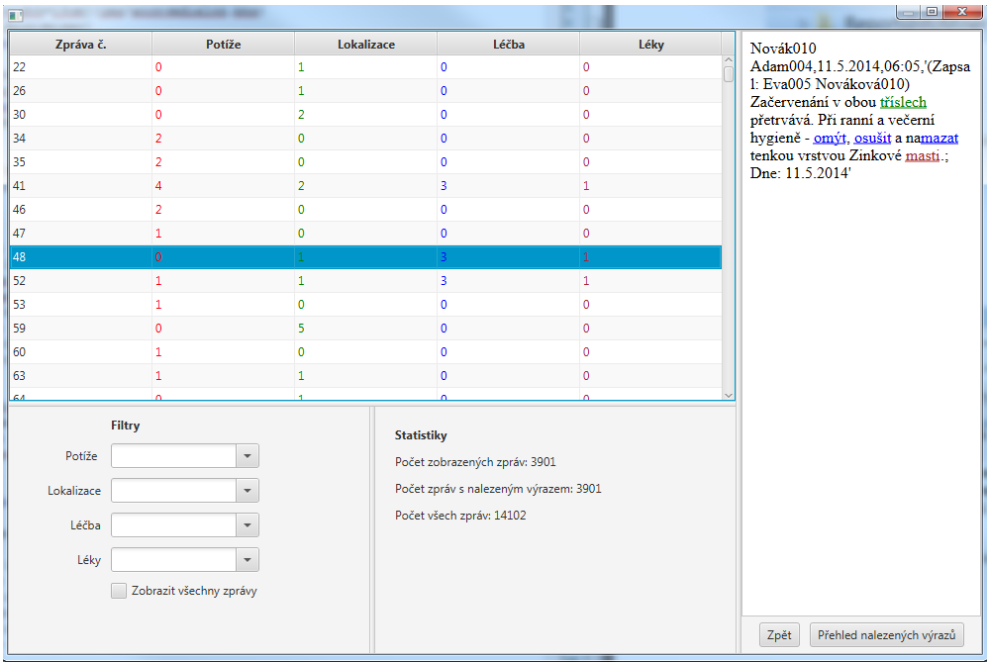
Editace hledaných výrazů - výrazy jsou rozděleny do čtyř tematických slovníků - Potíže, Lokalizace, Léčba, Léky. Slovníky lze dočasně vypínat, stejně jako jednotlivé výrazy. Editace uvnitř tabulky musí být potvrzena klávesou ENTER. Na konci editace je třeba změny potvrdit tlačítkem Uložit.



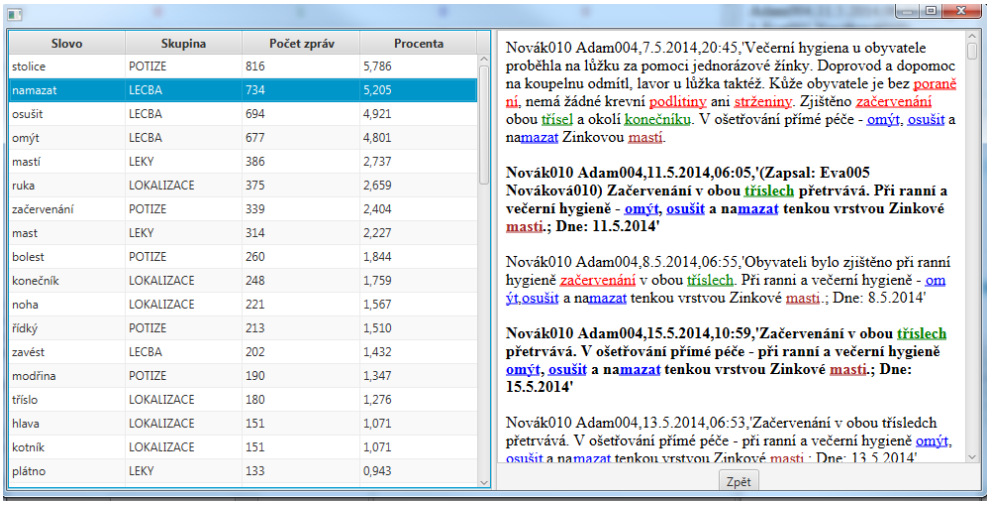
Obrázek B.5: Editace pojmových slovníků

Po **Startu** se vytvoří výsledková databáze a lze prohlížet výsledky. Jsou zobrazeny ve dvou tabulkách. První ukazuje přehled zpráv s nalezenými výrazy: V comboboxech vlevo dole lze zvolit filtr na nalezené výrazy. Filtry jsou spolu spojeny klauzulí AND. Checkbox ”Zobraz všechny zprávy” umožňuje vidět úplně všechny zprávy, tedy i ty, ve kterých nebyl nalezen žádný výraz. V pravém poli je zobrazena zpráva z aktuálního řádku.

Tlačítkem **Přehled nalezených výrazů** přejdeme do druhé výsledkové tabulky: Zde jsou zobrazeny nalezené výrazy spolu s četností a v pravém poli jsou zobrazeny všechny zprávy, ve kterých se výraz vyskytnul.



Obrázek B.6: První výsledková tabulka



Obrázek B.7: Druhá výsledková tabulka

C Obsah přiloženého CD

Na přiloženém CD jsou následující data:

Programs - oba programy spolu s knihovnami a testovacími daty
Sources - zdrojové soubory
Text - Text práce v pdf a latex

Literatura a odkazy

[1] Knihovna OpenNLP, *https://opennlp.apache.org/*

[2] Seznam všech jmen v ČR s doplněným oslovením (5. pádem - vokativem) *http://www.validace.cz*

[3] SQLite JDBC *http://mvnrepository.com/artifact/org.xerial/sqlite-jdbc/*

[4] Charu C. Aggarwal, ChengXiang Zhai *Mining Text Data*

[5] Helena Palátová, Marek Grác *Segmentace textu na věty*

[6] Karásek, Šanda, Burget, Morský *Strojové učení základem pro hybridní lemmatizační algoritmus*

[7] Petr Sgall *Typy jazyků a jejich základní vlastnosti*

[8] Hamish Cunningham, *T*ext Processing with GATE, Gateway Press CA ©2011

[9] Slavíček, Pavel, *GATE plugin pro rozdělování textu na věty - Bakalářská práce, https:// otik.uk.zcu.cz/handle/11025/5527*

[10] Ševčíková, Žabokrtský, Straková, Straka: *Czech Named Entity Corpus 2.0, https://lindat.mff.cuni.cz/repository/xmlui/handle/1858/00-097C-0000-0023-1B22-8*

[11] Anjali Ganesh Jivani: *A Comparative Study of Stemming Algorithms*

[12] Ljiljana Dolamic, Jacques Savoy: *Indexing and stemming approaches for the Czech language*

[13] Silviu Guiasua, Abe Shenitzer: *Princip maxima entropie*

[14] Adam L. Berger, Stephen A. Della Pietra: *A Maximum Entropy Approach to Natural Language Processing*