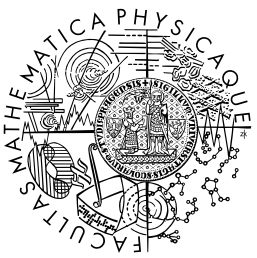


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

DIPLOMOVÁ PRÁCE

Jakub Michalko

Algoritmy detekce obchodních dokumentů podle šablon

Katedra softwarového inženýrství

Vedoucí diplomové práce:	doc. RNDr. Tomáš Skopal, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové systémy

Praha 2016

Chcel by som poďakovať svojmu vedúcemu diplomovej práce doc. RNDr. Tomášovi Skopalovi za rady, pomoc a usmernenie, ktoré mi pomohli pri písaní tejto práce. Ďalej by som chcel poďakovať vojej manželke a rodine za ich podporu. Tiež by som chcel poďakovať pánovi Koláčkovi zo spoločnosti COST CUTTING solutions, ktorý mi predviedol software ReadSoft INVOICES a poskytol inšpiráciu a podnetné informácie týkajúce sa problematiky spracovávania oskenovaných dokumentov. V neposlednom rade by som chcel poďakovať kolegovi Jakubovi Jarošovi za trpezlivosť a pomoc pri zvládaní pracovných povinností.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Algoritmy detekce obchodních dokumentů podle šablon

Autor: Jakub Michalko

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: Tomáš Skopal, Assoc. Prof.

Abstrakt: Diplomová práca sa zaoberá analýzou a návrhom systému pre automatické rozpoznávanie dokumentov. Systém spracuje dokument a prevedie ho do textovej podoby, pričom musí byť zachovaná informácia o pôvodnej polohe slova v dokumente. Tieto dáta budú následne preskúmané a určitým dátam bude pridelený ich význam. Spôsob, akým bude dátam pridelený význam je založený na pravidlách, ktoré môže meniť užívateľ podľa svojej potreby. Následne podľa dát, ich prideleného významu a ich polohy, systém nájde podobný dokument a podľa neho identifikuje aktuálne skúmaný dokument.

Klíčová slova: pološtrukturované dokumenty, annotácia,OCR, vyhľadávanie dokumentov

Title: Algorithms for business document detection using templates

Author: Jakub Michalko

Department: Department of Software Engineering

Supervisor: Tomáš Skopal, Assoc. Prof.

Abstract: Thesis deals with analysis and system design for automatic document recognition. The system explores the document and converts it into text data with information about the position of the word in original document. These data will then be reviewed and some of them will be assigned their importance. The way the data will be assigned is based on rules which may vary according to user needs. According to the data, their assignment and the importance of their position, the system finds a similar document and identifies the current document.

Keywords: semistrucutred documents, annotation, OCR, document search

Obsah

1. Úvod

Dokumenty sa v dnešnej dobe nachádzajú buď v papierovej, alebo elektronickej podobe. Nevýhodou papierových dokumentov je, že prístup k nim je veľmi neefektívny a v medzinárodnej organizácií skoro nemožný, nehovoriac o nákladoch s tým spojených. Prevod do elektronickej podoby tieto problémy odbúrava, avšak v súčasnej dobe nestačí len oskenovať dokument ako obrázok a uložiť ho niekam na zdielané úložisko. Väčšina užívateľov totiž nechce len hľadať informáciu v jednom dokumente, ale chce vyhľadávať informáciu nad niekoľkými dokumentami. To samozrejme nie je možné bez toho, aby dokument nebol prevedený do textovej podoby, respektíve do podoby, v ktorej je vyhľadávanie možne. Korporátna sféra zase vyžaduje prevod papierových dokumentov do softvérového systému, s ktorým pracuje.

Každá organizácia rieši zadávanie dokumentov do interného systému po svojom. Bežnou praxou je najať si pracovnú silu, ktorá prepisuje dokumenty do interného systému. Tento spôsob je najlepší v prípadoch, keď sú dokumenty písane ručne. Iné organizácie zase nechávajú túto činnosť na klientoch. Bežnou praxou je umiestnenie formulára na webové stránky, ktorý vyplňujú klienti. V prípade, kedy sa jedná o dokumenty generované systémom, alebo je nežiadúce, aby klienti zadávali dokumenty do systému, sa ako najlepšie riešenie naskytuje spracovať dokument softvérovo a ten zaniesť do interného systému automaticky.

Softvérové spracovanie tlačených dokumentov je stále sa rozvíjajúce odvetvie. Základom pre spracovanie tlačeného dokumentu je OCR systém, ktorý v obrázku dokumentu identifikuje znaky a tie prevedie na odpovedajúce znaky v elektronickej podobe.

1.1 Motivácia

Na trhu existuje mnoho komerčných riešení (niektoré sú uvedené v kapitole 2.2), no stále nie sú dokonalé. Svoje know-how si prísne strážia. Všetky sú navyše špecializované na konkrétny druh dokumentu ako napríklad faktúra, alebo životopisy. Anotovaniu dokumentov a dolovaniu dát z dokumentov sa venuje celé odvetvie informatiky [1], no predovšetkým je zamerané na anotovanie neštrukturovaných dokumentov.

V tejto práci sa k analýze dokumentu pristupuje ako k analýze ľubovoľného typu dokumentu. Prednostne ale bude vychádzať z analýzy korporátnych dokumentov.

1.2 Cieľ práce

Hlavným cieľom tejto práce je analyzovať dokument, na jej základe následne vyhľadať čo najpodobnejší spracovaný dokument (označovaný ako šablóna) a podľa neho v dokumente identifikovať významné časti textu.

Unikátny spôsob vyhľadávania podobnosti dokumentov tkvie vo vzájomnej polohe blokov textu. Blokom textu sa rozumie skupina slov alebo riadkov, ktoré

[1] respektíve časť odvetvia *information retrieval* zaoberajúce sa dolovaním dát

spolu nejakým spôsobom súvisia. Mnoho dokumentov má podobné rozloženie blokov textu v dokumente (napríklad adresa zákazníka býva v pravej hornej časti dokumentu). Práve tento spôsob vyhľadávania umožňuje priradiť význam jednotlivým blokom textu, určiť typ dokumentu zo šablóny, ktorá stačí že je podobná nejakej spracovanej šablóne.

Anotácia textu znamená priradiť celému textu, alebo jeho častiam význam ako napríklad meno, priezvisko alebo telefónne číslo. Cieľom práce bude návrh a implementácia algoritmov, ktoré sa budú snažiť anotovať časti textu od jednotlivých slov až po bloky textu. Anotácia textu bude prebiehať zdola nahor, tj. od jednotlivých slov až po bloky textov. Podľa typu anotovaných blokov textu, ich vzájomných polôh a podľa už analyzovaných dokumentov (šablón) hľadá aplikácia podobný dokument, podľa ktorého určí typ dokumentu a identifikuje dôležité dáta.

1.3 Štruktúra práce

Táto práca pozostáva z niekoľkých kapitol:

- *Existujúce riešenia* pojednáva o spôsoboch spracovania dokumentov a aplikáciách, ktoré spracovanie automatizujú
- *Model* predstavuje zobecnený a formálne definovaný návrh riešenia
- *Implementácia riešenia* obsahuje popis dátových štruktúr, algoritmov a heuristík, potrebných pre implementáciu modelu
- *Užívateľská dokumentácia* obsahuje inštaláciu aplikácie a jej ovládanie
- *Programátorská dokumentácia* popisuje základné triedy a štruktúry aplikácie
- *Praktické experimenty* obsahuje výsledky meraní a ich zhrnutie
- *Záver* zhrňuje hlavné myšlienky a zistenia tejto práce

2. Existujúce riešenia

Podstatou tejto práce je navrhnúť spôsob, ako dáta v papierovej podobe preniesť do elektronického systému. V tejto kapitole si popíšeme existujúce spôsoby a softwarové riešenia, ktoré to dokážu.

2.1 Ručné zadávanie

2.1.1 Prepis

Medzi najstarší a stále používaný spôsob zadávania dokumentov do systému je ručné zadávanie. V praxi pri zadávaní väčšieho počtu dokumentov do systému sa najme lacná pracovná sila, ktorej zaškolenie vyžaduje krátky čas. Tento spôsob sa uprednostňuje v situáciách, kedy sa do systému zadáva buď malý počet dokumentov a spoločnosti sa neoplatí kupovať si drahé riešenia. Ďalším prípadom je, keď softvér nedokáže správne spracovať dokumenty. Väčšinou sa jedná o ručne písané dokumenty alebo ručne vyplnené dotazníky. Nevýhodou tohoto spôsobu sú veľké časové a finančné náklady pri prepise strojovo generovaných dokumentov, ktoré sú inak softvérovo spracovateľné.

2.1.2 Vyplňovanie elektronického formulára

Iným spôsobom, ako zaviesť dokument do systému je nechať samotného užívateľa (respektíve odosielateľa) vyplniť elektronický formulár, ktorý je jednoducho spracovateľný systémom. Takéto formuláre často predstavujú napríklad dotazníky, registračné formuláre alebo objednávky internetových predajní.

Výhodou tohoto spôsobu je okamžité uloženie dokumentu do systému.

Nevýhodou je rôznorodosť rozhraní a dokumentov. Každý správca systému si to robí takzvane "po svojom". Výsledkom toho je, že jeden typ dokumentu/objektu, napríklad karta kontaktu, má v Google contacts jednu štruktúru, v Microsoft exchange druhú a v Skype opäť inú štruktúru. V ideálnom prípade by existoval jeden štandard popisujúci všetky typy objektov/dokumentov a jedna aplikácia, pomocou ktorej by užívatelia mohli zadávať tieto objekty/dokumenty. Celonárodné a medzinárodné organizácie zaoberajúce sa štandardizovaním typov dokumentov, vytvárajú protokoly a smernice, ktoré umožňujú priamu komunikáciu medzi rôznorodými systémami. V praxi ale nie každý investuje svoje prostriedky do implementácie rozhrania splňujúceho štandardy a to buď kvôli ich neznalosti, alebo kvôli množstvu nákladov, ktoré by musel na ne vynaložiť.

2.2 Automatické rozpoznávanie

2.2.1 Datamolino

Datamolino je online služba, ktorej vstupom sú oskenované dokumenty v podobe PDF súborov alebo obrázkov a výstupom sú extrahované dáta, ktoré je možné buď importovať do účtového systému, alebo stiahnuť v podobe CSV či XSL súboru. Túto službu je možné bezplatne si vyskúšať po dobu 7 dní. Špecializujú

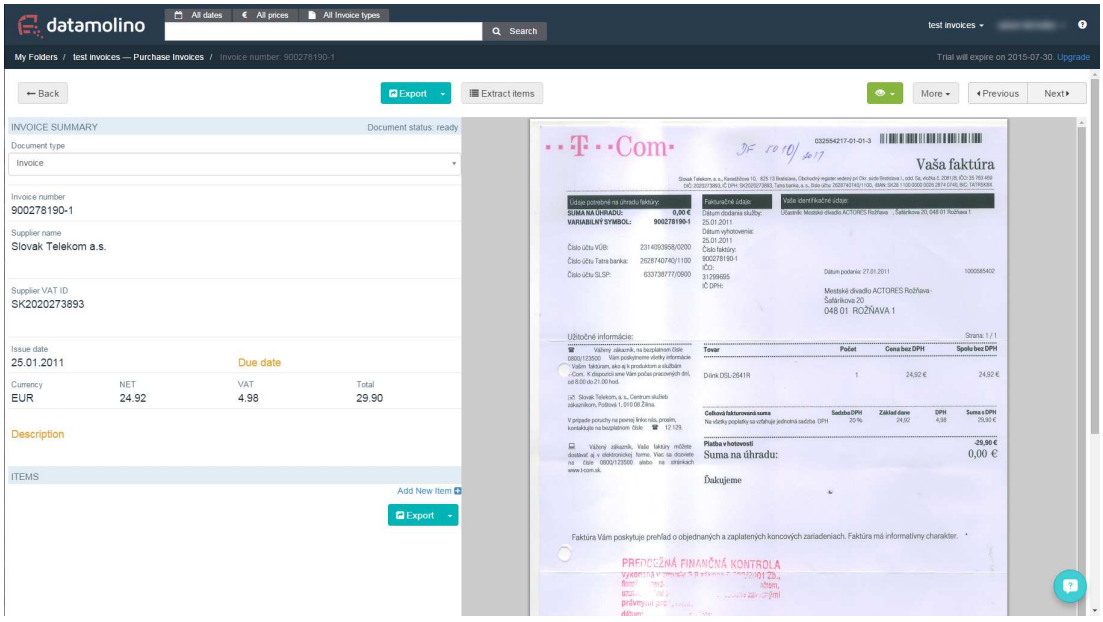
sa na faktúry, proformy faktúr, dobropisy a účtenky. Ich zámerom je oprostiť užívateľa od akejkoľvek nutnosti zásahu do procesu rozoznávania. To vedie k nutnosti čakať 1 až 3 dni na spracovanie. Tento startup projekt sa snaží konkurovať existujúcim službám cenou a odlišným prístupom k spracovaniu dát a možnosťou exportovať dáta do mnohých účtovných systémov.

Po nahraní dokumentu na vzdialený server sa automaticky spustí rozoznávanie. Systém sa snaží automaticky detekovať o aký typ dokumentu sa jedná a vyextrahovať z neho dôležité dáta. Rozoznané dokumenty môžu skončiť v troch stavoch:

- *rozpoznané*
- *1 deň*
- *3 dni*

Na testovacích dátach dokázal systém automaticky [1] vyextrahovať číslo faktúry, IČO odosielateľa a celkovú čiastku faktúry. V ostatných prípadoch bolo nutné počkať, než operátori potvrdia/doplnia údaje na faktúre. Systém automaticky nerozoznáva položky na faktúre. Pri testovaní v niektorých prípadoch namiesto položiek faktúry vyplnil *popis faktúry*, do ktorého vložil text popisujúci prvú položku na faktúre. V jednom prípade dokonca s preklepom a nesprávnym formátom dátumu (na niektorých miestach chýbala medzera). Pre doplnenie položiek musí užívateľ kliknúť na tlačítko rozoznať položky. Tým sa opäť faktúra dostala do stavu *1 deň* alebo *3 dni*. Extrahované položky mali správne vyplnené meno položky, ale chýbali údaje o cene a DPH. Ďalšou nevýhodou je, že nezistíte, z ktorého miesta v obrázku dáta pochádzajú.

Po technickej stránke aplikácia nie je na špičkovej úrovni, ale pre potreby zákazníkov postačuje.



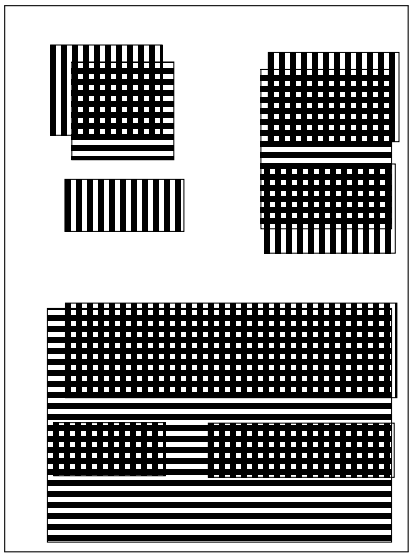
Obrázok 2.1: Datamolino

[1] Dokument neskončil v stave 1 deň alebo 3 dni.

2.2.2 LANA

LANA aplikácia bola vyvinutá na Matematicko-Fyzikálnej fakulte Karlovej Univerzite ako softvérový projekt, kde som bol členom vývojového tímu. Aplikácia najprv skonvertovala oskenovaný dokument do obrázku [2], ktorý následne vyrovnala a pomocou aplikácie Tesseract extrahovala text s jeho umiestnením v dokumente. Slová z extrahovaného textu postupne spájala do riadkov a riadky do blokov textu podľa toho, aká veľká medzera ich oddeľovala.

Hlavnou myšlienkou celého algoritmu rozoznávania bolo, porovnávanie aktuálneho dokumentu s dokumentami, ktoré boli správne spracované a v nich boli identifikované dôležité dáta ako číslo faktúry, adresa odosielateľa a pod. Zhoda dvoch dokumentov záležala na tom, ako sa prekrývajú jednotlivé bloky textov, ako je zobrazené na obrázku 2.2. Bloky textu jedného dokumentu sú zobrazené obdĺžnikmi s horizontálnym šrafovaním, bloky textu druhého sú zobrazené vertikálnym šrafovaním. Prienik je následne vyjadrený percentuálne vzhľadom k prvému a k druhému bloku. Výsledná podobnosť je rátaná ako priemer týchto percentuálnych zhôd. Napríklad prekrytie vzhľadom k dokumentu A je 80%, vzhľadom k dokumentu B je to 60%. Výsledná podobnosť oboch dokumentov je 70%.



Obrázok 2.2: Prekrytie dvoch dokumentov

Aby bolo možné dokumenty porovnávať, bolo potrebné normovať všetky dokumenty na jednotnú šírku, pretože prekrytia blokov boli počítané na základe ich súradníc. Ak by totiž boli dokumenty naskenované v rôznych rozlíšeniach, tie isté bloky by mali iné súradnice a prekrývali by sa len čiastočne, alebo vôbec.

LANA podľa nájdeného podobného dokumentu dokázala následne určiť, o aký typ dokumentu sa jedná a identifikovať miesta, kde sa dôležité dáta nachádzajú. V porovnaní s Datamolino je LANA na rovnakej úrovni. Lana umožňuje klientom opraviť dáta, zatiaľčo Datamolino má na to interných operátorov.

V tejto práci je pre vyhľadávanie použitý odlišný spôsob vyhľadávania počítania podobnosti dvoch dokumentov (viď kapitola Model). Aplikácia, ktorá implementuje algoritmy bola vyvinutá nezávisle na LANA aplikácii.

[2] V prípade, že vstupným dokumentom bol pdf dokument.

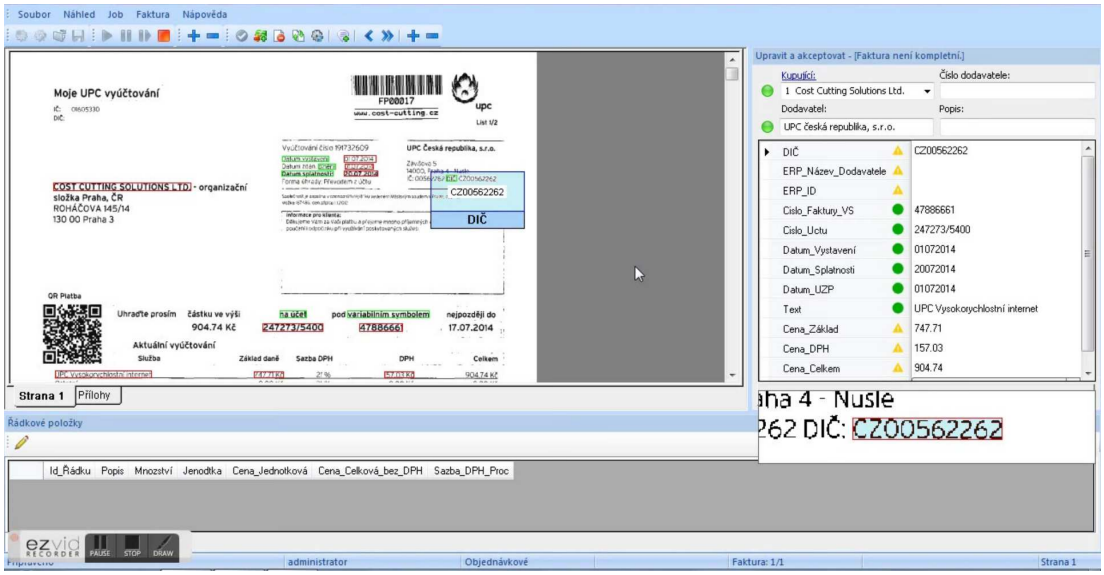
2.2.3 ReadSoft INVOICES

Spoločnosť ReadSoft sa vo všeobecnosti zaoberá všetkým, čo súvisí so spracovaním dokumentov a ich workflow. Jedným z ich produktov je služba ReadSoftInvoices, ktorá spracováva naskenované dokumenty.

Umožňuje prepojenie s inými účtovnými systémami. Dokáže spracovávať akýkoľvek typ dokumentu. Pre čo najlepšie extrahovanie textu z dokumentu používa až 3 ICR enginy[3]. V prípade nejednoznačnosti rozoznaného znaku, porovnáva výsledok s ďalšími dvoma. ICR umožňuje rozoznať nielen tlačené, ale aj ručne písané znaky.

Nové (nenaučené) typy dokumentov je možné ho naučiť, čím sa vytvorí takzvaná šablóna dokumentu. Podľa týchto šablón následne systém dokáže určiť miesta, kde sú ktoré informácie umiestnené.

Okrem toho sa v nenaučených dokumentov snaží systém nájsť dôležité informácie podľa kľúčových výrazov, ako napríklad *DIČ*, alebo *Daňové identifikační číslo*. V prípade, že dané kľúčové slovo nenájde, snaží sa nájsť v jeho bezprostrednom okolí jeho hodnotu (väčšinou napravo od, alebo pod kľúčovým slovom), ktorá odpovedá významu kľúčového slova. Príkladom je DIČ českej spoločnosti začína znakmi CZ a pokračuje číslami (alebo aj znakmi), rodnému číslu odpovedá 6 čísel, znak / a 4 čísla atp. .



Obrázok 2.3: Readsoft

Aplikácia je platená a nie je možné si ju bezplatne vyskúšať. Pri prezentácií aplikácie bola zdôraznená požiadavka na kvalitné skenovacie zariadenie pre čo najlepšie výsledky. Aplikácia bola nastavená na český jazyk. Keďže naše testovacie faktúry boli v slovenčine a v trochu horšej kvalite [4], bolo nám povedané, že na takýchto dokumentoch to nebude pracovať správne a teda nevieme, aké výsledky by sme dostali nad týmito dátami. Ale podľa popisu a prezentačných videí, je ReadSoft INVOICES technologicky najvyspelejšou z pomedzi analyzovaných aplikácií v tejto práci.

[3]ICR je skratka *Intelligent character recognition*, čo je pokrokovejšia verzia OCR

[4]testovacie dáta sú súčasťou priloženého CD

Porovnanie obecnejších vlastností uvedených aplikácií je možné nájsť v tabuľke 7.1.

3. Model

Táto kapitola popisuje model riešenia, ktorý je rozdelený na niekoľko častí:

- *Monomorfizmus dokumentov* - model dát a definícia podobnosti layoutov dvoch dokumentov
- *Miera podobnosti dokumentov* - definícia metriky podobnosti dvoch dokumentov
- *Šablóna* - definícia šablóny dokumentu
- *Nájdenie šablóny* - popis heuristiky pre nájdenie šablóny dokumentu

3.1 Monomorfizmus dokumentov

Vyhľadávanie dokumentov je postavené na myšlienke, že dva dokumenty (respektíve dokument a šablóna dokumentu) sú si podobné ak majú podobné rozloženie layoutu a jednotlivé časti layoutu (bloky textu) obsahujú rovnaký typ informácie (adresa, hlavička a pod.). Je potrebné zohľadniť aj to, že jednotlivé bloky textu môžu byť rôzne vysoké[1]. Vzájomná poloha jednotlivých blokov textu je však v podobných dokumentoch zachovaná (napr. tabuľka je pod adresou, alebo hlavička sa nachádza nad všetkými ostatnými blokmi textu).

Prevedením dokumentu na graf tak, že:

- blok textu predstavuje vrchol v grafe
- typ informácie v bloku textu predstavuje ofarbenie vrcholu
- vzájomná poloha častí layoutu predstavuje hranu medzi vrcholmi, ktorá má priradený uhol (zvierajúci s osou x)

je možné model dokumentu formalizovať podľa definície 1. Obrázok 3.1 tento prevod ilustruje.

Definícia 1. *Majme konečnú množinu farieb C a graf G(V,E). Potom zobrazenie $c : V \to C$, ktoré je na, sa označuje ako* ***farbenie vrcholov grafu G*** *a zobrazenie $u : E \to [0, 2\pi]$, pre ktoré platí $\forall e(v_1, v_2) \in E : u(v_1, v_2) = (u(v_2, v_1) + \pi) \bmod 2\pi$, sa označuje ako* ***uhly hrán grafu G****. Štvorica* $\boldsymbol{G(V, E, c, u)}$*, kde c je farbenie vrcholov a u sú uhly hrán G, sa označuje ako* ***graf layout****.*

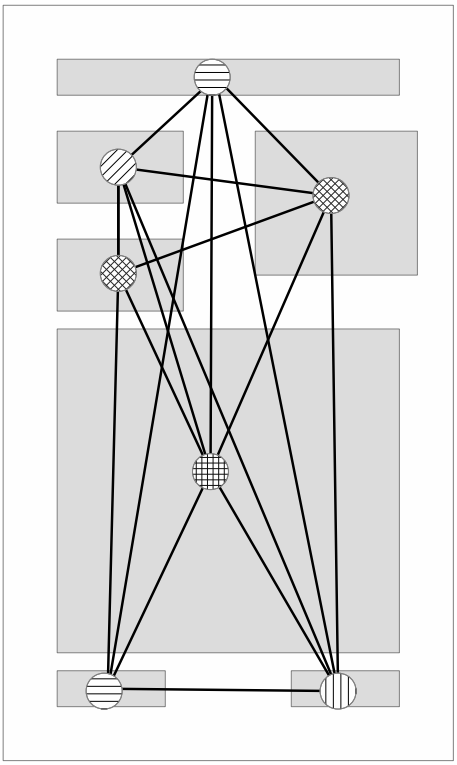
Hľadanie podobných dokumentov znamená nájsť zobrazenie layoutu jedného dokumentu na druhý, teda zobrazenie vrcholov medzi dvoma grafmi. Nájdené zobrazenie naviac musí spĺňať, že farby vrcholov vzoru a obrazu sú rovnaké.

Definícia 2. *Nech $G(V_G, E_G, c_G, u_G)$ a $H(V_H, E_H, c_H, u_H)$ sú dva grafy layoutov. Nech c_G a c_H sú ofarbenia grafov G a H a h je monomorfizmus grafu G na H. Ak pre h naviac platí, že*

$$\forall v_G \in V_G, \exists v_H \in V_H : h(v_G) = v_H \Rightarrow c_G(v_G) = c_H(v_H)$$

je toto zobrazenie označené ako ***farebný monomorfizmus****.*

[1] Napríklad tabuľky mávajú výšku závislú na počte riadkov



Obrázok 3.1: Model layoutu dokumentu

K farebnému monomorfizmu dvoch grafov je potrebné pridať podmienku, že aj hrany musia mať podobný sklon. Inak by nebolo možné nájsť dva dokumenty s podobným rozložením layoutu. Farebne monomorfné grafy, reprezentujúce dokumenty by sa obsahovo podobali (obsahovali by text podobného typu), ich layout by však mohol byť úplne odlišný. Definícia 3 popisuje vlastnosti funkcie počítajúcej podobnosť uhlov hrán.

Definícia 3. *Nech h je farebný monomorfizmus grafov $G(V_G, E_G, c_G, u_G)$ a $H(V_H, E_H, c_H, u_H)$. Funkcia* ***sim*** $: [0, 2\pi] \times [0, 2\pi] \to \Re^+$ *počítajúcu podobnosť uhlov dvoch hrán, pre ktorú platí*

$$\forall u \in [0, 2\pi] : \lim_{x \to 0} sim(u, u + x) = 0$$

sa označuje ako ***miera zhody uhlov****.*

Z definície funkcie *sim* vyplýva, že pre dva rovnaké uhly má funkcia hodnotu 0. Obrázok 3.2 ilustruje funkciu *sim*, ktorá porovnáva hrany z dvoch grafov (dokumentov). Funkcia *sim* môže byť definovaná napríklad ako veľkosť rozdielu dvoch čísel, teda :

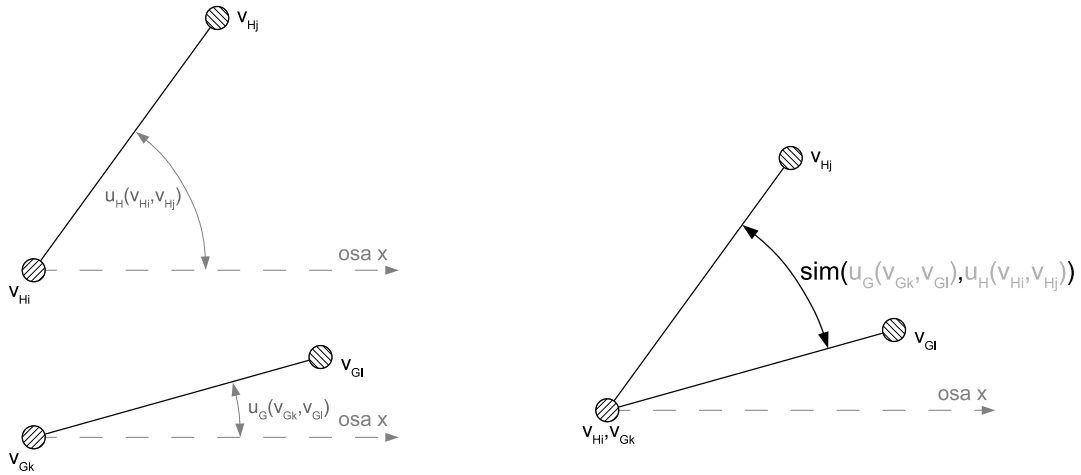
$$sim(u_1, u_2) = |u_1 - u_2|$$

Vďaka *sim* je možné definovať maximálny rozdiel v uhloch (ϵ), ktorý určuje, či porovnávané hrany sú si ešte podobné.

Na jeho základe je definovaný monomorfizmus dvoch dokumentov, ktorého hrany môžu byť odlišné, no v nejakej tolerancii:

Definícia 4. *Ak pre farebný monomorfizmus h, funkciu sim a konštantu ϵ platí:*

$$\forall e_G(v_{G,1}, v_{G,2}) \in E_G : sim(u_G(v_{G,1}, v_{G,2}), u_H(h(v_{G,1}), h(v_{G,2})) \leq \epsilon)$$

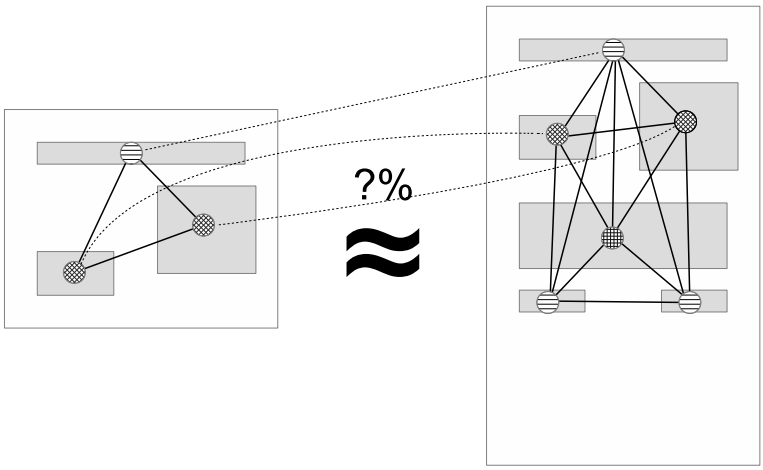


Obrázok 3.2: Podobnosť uhlov hrán v dokumentoch

potom monomorfizmus hrán $\boldsymbol{h_{sim,\epsilon}} \subset E_G \times E_H$ *grafov G a H, sa označuje ako* ***monomorfizmus layoutov****.*

3.2 Miera podobnosti dokumentov

Nájdenie grafu H, ktorý je layoutovo monomorfný s grafom G ešte nemusí znamenať, že sú si grafy podobné, ako ilustruje obrázok 3.3. Je vidieť, že graf vpravo má viac vrcholov, než graf vľavo. Miera podobnosti teda musí zohľadniť aj veľkosť oboch grafov (z hľadiska počtu vrcholov).



Obrázok 3.3: Miera podobnosti dvoch dokumentov

Definícia 5. *Nech G a H sú grafy layoutu. Nech pre nejaké* $\epsilon > 0$ *existuje podfgraf G' grafu G, pre ktorý exituje* $h_{sim,\epsilon}, \epsilon$ *s H. Potom funkcia laysim* : $E_G \times E_H \rightarrow [0,1]$*, počítajúca podobnosť podgrafu G' s H, je definovaná rovnosťou:*

$$laysim(G, H, h_{sim,\epsilon}) = \sqrt{\frac{|h_{sim,\epsilon}|}{Max(|E_G|, |E_H|)}}$$

Teda ak $h_{sim,\epsilon}$ je zobrazenie niektorých hrán G na hrany grafu H, potom

$$laysim(G, H, h_{sim,\epsilon}) = \sqrt{\frac{počet\, podobných\, hrán\, medzi\, grafom\, G\, a\, H}{Max(počet\, hrán\, v\, G,\ počet\, hrán\, v\, H)}}$$

.

Vzorec vychádza z predpokladu, že graf je typu K_n, ktorý má $n \times n - 1/2$ hrán. Odmocnenie výsledku dáva "lineárnejšiu"závislosť výsledku k počtu zhodných vrcholov v grafoch.

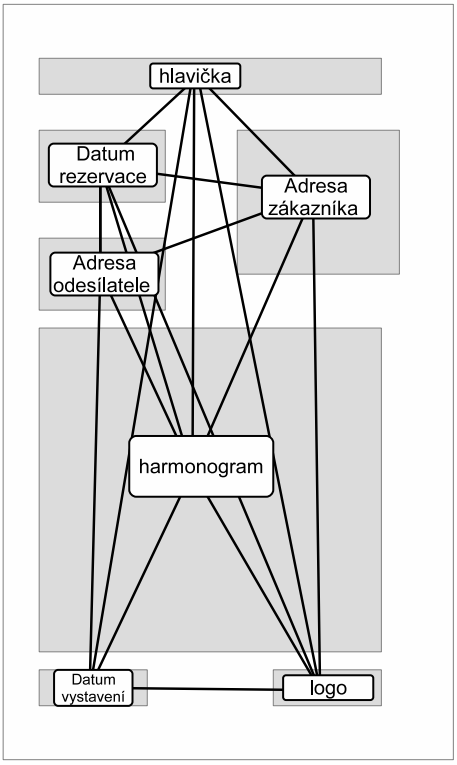
Definícia 6. *Nech M je množina grafov layoutov (šablóny), G je graf layoutu a $\epsilon > 0$. Nech G^{MAX} je indukovaný podgraf G a $H^{MAX} \in M$, pre ktoré existuje monomorfizmus layoutov $h^{MAX}_{sim,\epsilon}$. Ak platí*

$$\forall G' \subset G, \forall H' \in M, \forall {h'}_{sim,\epsilon} : laysim(G', H', h'_{sim,\epsilon}) \leq laysim(G^{MAX}, H^{MAX}, h^{MAX}_{sim,\epsilon})$$

, potom $\boldsymbol{H^{MAX}}$*, sa označuje ako* ***najpodobnejší layout ku G z množiny*** M*.*

Týmto je teda možné definovať, akú maximálnu odchýlku môže graf mať, aby bol akceptovaný ako rovnaký. Napríklad ak by maximálna odchýlka podobnosti grafov bola 10%, potom graf H, pre ktorý platí $laysim(G, H, h_{sim,\epsilon}) \geq 0.9$ je nájdeným podobným layoutom k layoutu G.

3.3 Šablóna



Obrázok 3.4: Šablóna

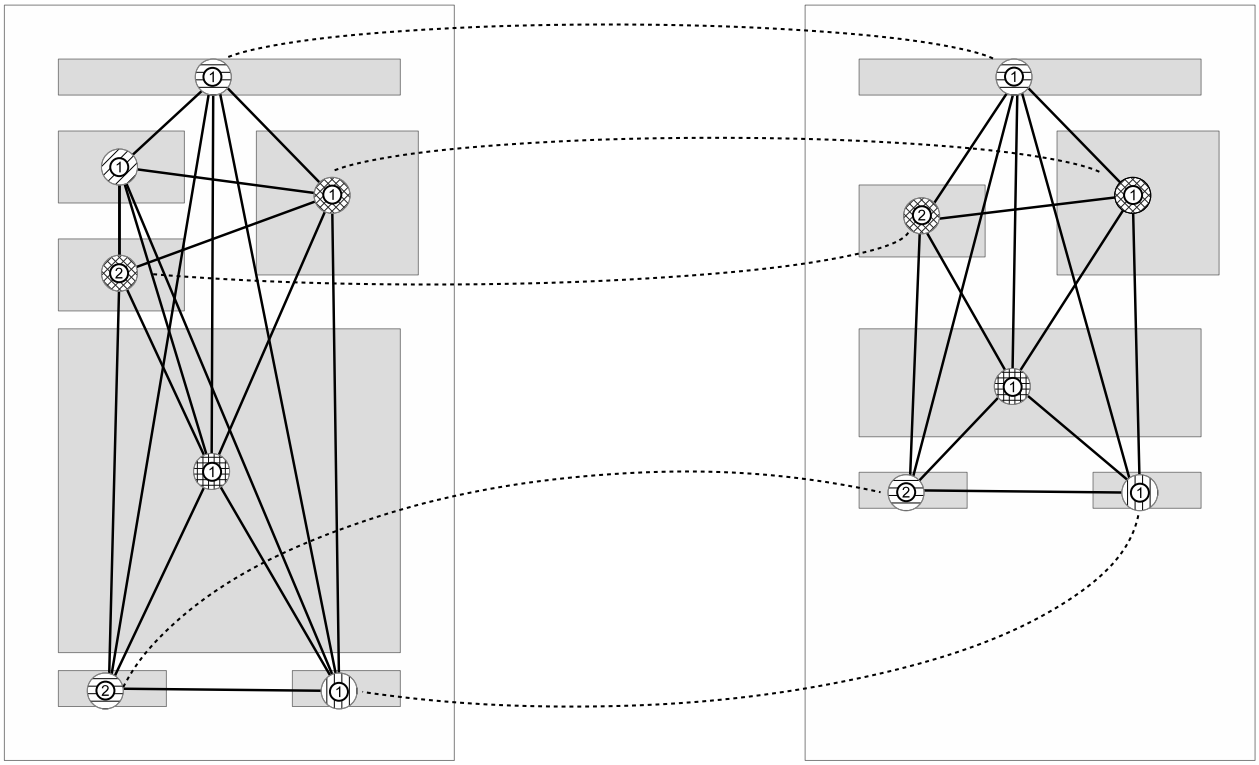
Šablóna dokumentu je rozšírením grafu layoutu (teda dokumentu) o informáciu, ktorý vrchol má aký význam. Pre dokument typu telegram to môže byť napríklad

meno odosielateľa, alebo meno prijímateľa. Obe tieto informácie sú rovnakého typu (tj. meno), avšak každá má iný význam.

Definícia 7. *Nech Meaning je množina "významov"a $G(V_G, E_G, c_G, u_G)$ je graf layoutu. Nech funkcia* $\boldsymbol{m_G : V_G \rightarrow Meaning}$ *priraďuje význam vrcholom grafu G. Potom sa pätica* $\boldsymbol{G(V_G, E_G, c_G, u_G, m_G)}$ *označuje ako šablóna dokumentu.*

3.4 Nájdenie šablóny

Hľadanie šablóny k dokumentu G znamená nájsť graf H z množiny šablón S a monomorfizmus layoutov $h_{sim,\epsilon}$ pre tieto dva grafy tak, aby podobnosť grafov $laysim(G, H, h_{sim,\epsilon})$ bola aspoň μ. Bez ujmi na obecnosti, nech μ je 0.9 (teda podobnosť musí byť aspoň 90%), a nech tolerancia uhlov hrán $\epsilon = \pi/8$ (45°).



Obrázok 3.5: Číslovanie vrcholov

Pri hľadaní homomorfizmu medzi dokumentom a šablónou sa využíva toho, že vrcholy sú ofarbené. Jediným problémom je správne priradiť vrcholy s farbou, ktorá sa v grafe vyskytuje viac krát. Riešenie, ktoré sa ponúka je využiť dvojrozmerné usporiadanie vrcholov a očíslovať vrcholy najprv podľa osy y a potom podľa osy x, ako na obrázku 3.5. Po očíslovaní má každý vrchol jedinečnú kombináciu: $(farba, poradie)$.

Ak teda pre vrchol v grafe G existuje vrhol v grafe H, ktorý má rovnakú farbu a číslo, je táto dvojica vrcholov pridaná do výslednej množiny, ktorá predstavuje hľadaný farebný monomorfizmus h. Skonštruovanie h je preveditelné v čase

$$O(max(|V_G| .log(|V_G|), |V_H| .log(|V_H|)))$$

pretože očíslovanie vrcholov v grafe G a trvá najviac $O(V_G)$, v grafe H $O(V_H)$. Nájdenie odpovedajúcich si vrcholov trvá $O(max(|V_G|\,.log(|V_G|), |V_H|\,.log(|V_H|)))$.

Ak pre $\epsilon = \pi/8$ (45°) a *sim* platí, že $h_{sim,\epsilon}(E_G, E_H)$, potom h je hľadaný monomorfizmus. Ak naviac $laysim(G, H, h_{sim,\epsilon}) > 0.9$, potom H je podobný dokument ku G. Pre overenie $h_{sim,\epsilon}(E_G, E_H)$ je potrebné prehľadať všetky hrany, čo zaberie čas $O(|E_G| + |E_H|)$. Výpočet hodnoty $laysim(G, H, h_{sim,\epsilon})$ sa výberom vhodných dátových štruktúr dá vypočítať v čase $O(1)$. Časová zložitosť pre zistenie, či dokumenty sú podobné trvá:

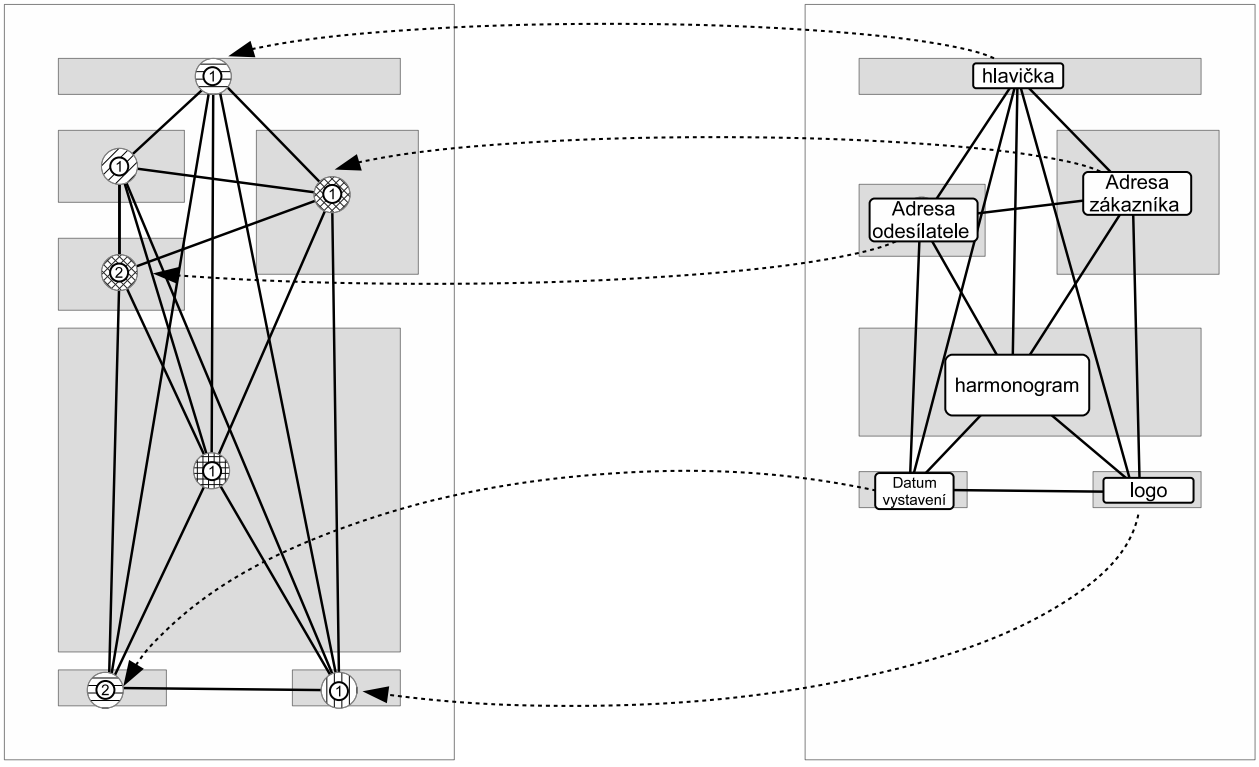
$$O(|E_G| + |E_H|)$$

Pretože $E_G >> V_G$, celková zložitosť pre zistenie podobnosti dokumentov je

$$O(|E_G| + |E_H|)$$

Ak je nájdený viac ako jeden podobný dokument, čiže šablóna, vyberie sa tá, ktorá má najväčšiu hodnotu $laysim(G, H, h_{sim,\epsilon})$. Z $h_{sim,\epsilon}$ je vytvorené inverzné zobrazenie h^{-1} pre vrcholy grafov. Pomocou neho je vrcholom v G priradený význam, čím sa z G stáva šablóna. Priradenie významu je preveditelné v čase :

$$O(|V_G|)$$



Obrázok 3.6: Priradenie významov zo šablóny.

4. Implementácia riešenia

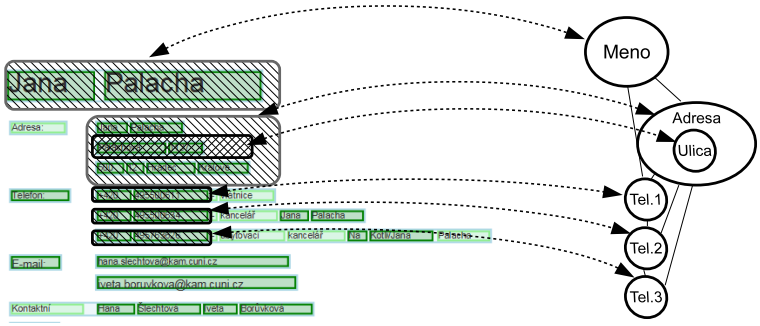
Táto kapitola pozostáva z niekoľkých častí:

- *Reprezentácia modelu* - popis základných dátových štruktúr použitých v implementácii modelu
- *Algoritmy* - zjednodušený popis algoritmov pre ofarbenie vrcholov, nájdenie monomorfizmu grafov a výpočet podobnosti layoutov
- *Parser dokumentov* - popis parsovania dokumentov, kde veľká časť sa venuje problematike vylepšeniu rozoznávania dokumentov
- *Anotátor dokumentov* - obsahuje popis pravidiel a spôsobu ich aplikácie pre anotovanie fráz v dokumente, teda ofarbovanie vrcholov
- *Vyhľadávač dokumentov* - obsahuje už popis vyhľadávania dokumentov

4.1 Reprezentácia modelu

4.1.1 AnnotatedPhrase

Anotovaná fráza označená ako **AnnotatedPhrase**, reprezentuje ofarbený vrchol v modeli. Frázu tvorí skupina slov *SpatialWord*, ktorých MOO (viď sekcia Minimálny Ohraničujúci Obdĺžnik) definuje MOO frázy. Grafický stred MOO určuje umiestnenie vrcholu v grafe layoutu a typ anotácie (teda význam fráze) definuje farbu vrcholu, viď obrázok 4.1. Ako je vidieť na obrázku, môže sa jedna *AnnotatedPhrase* nachádzať v inej *AnnotatedPhrase*. To samozrejme ničomu nevadí. Tieto dve *AnnotatedPhrase* sa reprezentujú tak, že sa sa tá vonkajšia fráza nachádza všade okolo vnútornej - tj. vo všetkých smeroch. Dôležité ale je ich umiestnenie voči ostatným *AnnotatedPhrase*, ktoré sa zachováva.



Obrázok 4.1: Anotovaná fráza a vrchol v grafe layoutu

AnnotatedPhrase je teda základným prvkom, ktorý je použitý pre vyhľadávanie podobných dokumentov. Základnými atribútmi *AnnotatedPhrase* sú:

- *AnnotationType* - typ anotácie, ako napríklad mesto, číslo, priezvisko a pod. V modeli predstavuje **ofarbenie vrcholu**.
- *Words* - slová, ktorá tvoria frázu.

- *Probability* - pravdepodobnosť, že slová skutočne odpovedajú typu anotácie.
- *TypeOccurence* - poradové číslo pre tento typ fráze. Je používané pre rozlíšenie, keď sa v dokumente nachádza viac fráz rovnakého typu (napríklad dva dátumy vedľa seba). V modeli predstavuje **poradie** ofarbeného vrcholu.
- *LeftTop, RightDown, CenterPoint* - MOO a stred MOO. V grafe layoutu predstavuje **umiestnenie vrcholu**

Ďalšími parametrami, ktoré pomáhajú pri implementácií algoritmov, sú *childAnnotations* a *parentAnnotations*. Anotácia sa môže skladať z iných *AnnotatedPhrase*, ako napríklad adresa, ktorá sa skladá z ulice, čísla domu, mesta a PSČ. Odkaz na rodičovskú anotáciu slúži pre hierarchické zoradenie vytvorených fráz.

Spôsob, akým sa vytvárajú fráze *AnnotatedPhrase* je popísaný v sekcii Anotovanie fráz.

4.1.2 Minimálny Ohraničujúci Obdĺžnik

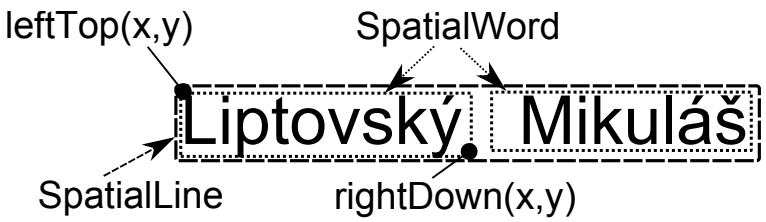
Minimálny ohraničujúci obdĺžnik (skratka **MOO**) MOO je najmenší obdĺžnik obaľujúci svoj obsah, ktorého hrany sú rovnobežné s osami. Obsahom takého MOO môže byť napríklad množina bodov, kriviek, alebo množina iných MOO. V tejto práci je obsahom MOO slovo *SpatialWord*, či *SpatialLine* a je reprezentovaný jeho dvoma rohovými bodmi:

- **leftTop** - bod so súradnicami X a Y, ktorý reprezentuje ľavý horný roh MOO
- **rightDown** - bod so súradnicami X a Y, ktorý reprezentuje pravý dolný roh MOO
- **centerPoint** - stredový bod odvodený z *leftTop* a *rightDown*

4.1.3 SpatialWord

SpatialWord predstavuje slovo a jeho MOO. Príklad *SpatialWord*u je zobrazený na obrázku 4.2. Jeho základnými atribútmi sú:

- **text** - slovo
- **leftTop, rightDown, centerPoint** - MOO a jeho stred



Obrázok 4.2: Príklad SpatialWord a SpatialLine

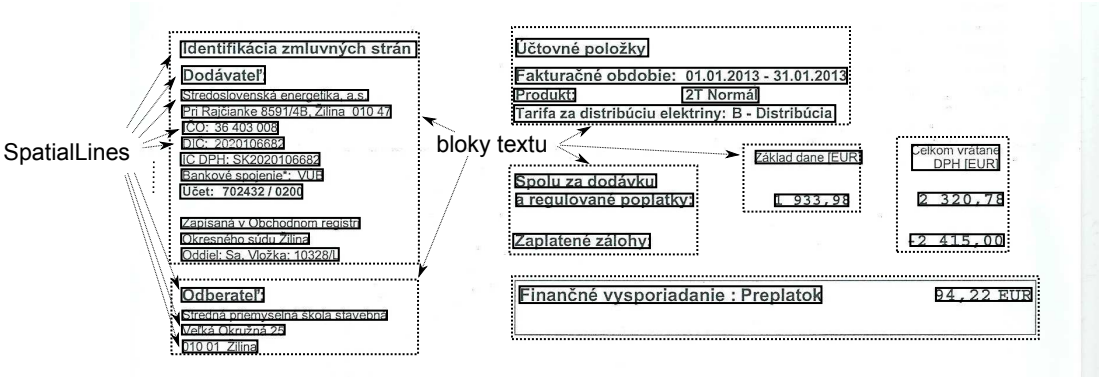
Implementácia objektu naviac obsahuje odkazy na *AnnotatedPhrase*,v ktorých sa slovo nachádza.

4.1.4 SpatialLine

Ako obrázok 4.2 ilustruje, **SpatialLine** zoskupuje *SpatialWord* do väčších celkov. Podmienkou ale je, že slová musia byť z rovnakého bloku textu. *Blok textu* je abstrakciou, ktorá priraďuje textu význam, ako napríklad adresa odberateľa, tabuľka, kontaktné údaje a tak podobne. Z toho plynie, že slová *SpatialWord*, ktoré patria napríklad adrese odosielateľa, nemôžu byť v jednej *SpatialLine* so slovami, ktoré určujú napríklad dátum splatnosti faktúry.

Postačujúcou podmienkou pre **SpatialLine** je, že jej slová oddeľuje malá medzera, ktorá nepresahuje dvojnásobok [1] výšky písma. Vo väčšine prípadov totiž medzera väčšia ako dvojnásobok výšky písma oddeľuje jednotlivé bloky textu.

Obrázok 4.3 ilustruje jednotlivé *SpatialLine*s, ktoré sú rozdelené do blokov textu. Blok textu je len abstrakciou.



Obrázok 4.3: Príklad SpatialLines

SpatialLine je priestorový objekt a teda má svoj *MOO*. Ten predstavuje MOO slov *SpatialWord*, ktoré patria do *SatialLine*. Základnými atribútmi sú:

- **words** - slová patriace do *SpatialLine*
- **leftTop, rightDown** - MOO, ktorý je vypočítaný ako MOO slov

Implementácia kvôli vyhľadávaniu v dokumente, obsahuje naviac odkaz na predchádzajúcu a následujúcu *SpatialLine* a číslo riadku, do ktorého patrí.

4.1.5 Document Item

DocumentItem predstavuje anotovanú frázu v dokumente a jej význam vzhľadom k typu dokumentu. Napríklad ak dokument obsahuje viac adries, kde jedna je adresa odosielateľa a druhá adresa prijímateľa. V modeli predstavuje funkciu m_G priradzujúcu **význam** vrcholu.

Document item teda obsahuje:

- **prhase** - odkaz na *AnnotatedPhrase*
- **meaning** - význam

Implementácia v skutočnosti obsahuje len hešovaciu mapu, reťazca a odkazu na *AnnotatedPhrase*, preto nemá samostatnú triedu.

[1] postačilo by, ak medzera by bola menšia ako trojnásobok výšky písma, ale už pri testovaní sa vyskytovali dokumenty, kde bola táto medzera menšia

4.1.6 SpatialDocument

SpatialDocument je implementáciou *grafu layoutu* v modeli. Obsahuje:

- zoznam *AnnotatedPhrase* - predstavujú ofarbené vrcholy grafu
- zoznam *SpatialWord*s - slová v dokumente
- zoznam *SpatialLine*s - riadky jenotlivých blokov
- zoznam *DocumentItem*s - význam jednotlivých fráz

SpatialDocument predstavuje jednu stránku v zdrojovom súbore, z ktorého boli vyextrahované slová do *SpatialWord*. Predpokladom je to, že dokument je jednostránkový. Spracovanie viac-stránkových dokumentov je vhodné ako ďalšie možné rozšírenie tejto práce.

SpatialDocument obsahuje okrem zoznamu všetkých jeho *SpatialLine*(a teda všetkých jeho slov *SpatialWord*), aj zoznam *anotovaných fráz* (viď 4.1.1).

Po rozoznaní obsahuje dokument naviac *typ dokumentu*(napríklad faktúra, letenka) a zoznam *dokumentových položiek* (napr. adresa odosielateľa, adresa prijímateľa, celková čiastka faktúry a pod.).

4.2 Algoritmy

4.2.1 Anotovanie fráz

Anotovanie fráz je komplexná problematika. V tejto práci sa anotovanie fráz vykonáva na základe užívateľom definovaných pravidiel, ktoré vychádzajú z:

- číselníku - zoznam konštánt, ktoré definujú význam, pravdepodobnosť a spôsob porovnania (matchCase, ignoreCase)
- regulárnych výrazov - reťazcu odpovedajúci regulárnemu výrazu je priradený význam a pravdepodobnosť
- anotovaných fráz - ak je konkrétna skupina anotovaných fráz vedľa seba (v konkrétnom poradí), vytvorí sa z nich nová *AnnotatedPhrase*.

Anotované frázy nemusia vychádzať len z užívateľom definovaných pravidiel. Zdrojom môže byť aj usporiadanie textu do tabuľky. [2] Podrobnejší popis spôsobu anotovania fráz je popísaný v sekcii implementácie Anotátor dokumentov a ich definácia sa nachádza v súbore *config/annotate.xml*.

4.2.2 Vyhľadávanie šablón

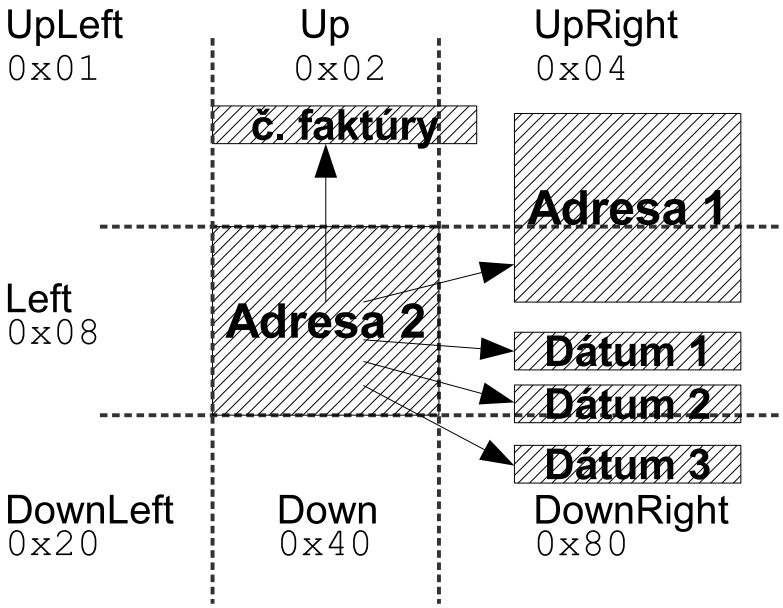
Ak k funkcii $\boldsymbol{sim(e_2, e_2)}$, počítajúcej podobnosť uhlov dvoch hrán, sa nastaví tolerancia ϵ na (90°), umožní to reprezentovať uhly zjednodušene. Celé okolie vrcholu (teda anotovanej fráze) sa rozdelí na 8 sektorov (viď obrázok 4.4). Uhly, ktoré

[2] Ako možné rozšírenie anotácie môže byť určenie, záhlavia dokumentu, popisu obrázku alebo obrázka samotného (v tomto prípade by sa jednalo o anotovanú frázu bez textu).

sú v rovnakom sektore sa teda líšia najviac o (90°). Uhol je tak reprezentovaný číslom sektoru, do ktorého spadá.

Ak každému sektoru je pridelená mocnina čísla 2, je možné bitovou mapou reprezentovať **intervalový rozsah okolia vrcholu**. Ten je potrebný pre prípady, keď jedna fráza sa nachádza vo viacerých sektoroch (viď fráza Adresa v obrázku 4.4). V implementácií to znamená, že ak bitové prekrytie dvoch bitmapových vektorov (bitový AND) je nenulové číslo, referované frázy sa nachádzajú v podobných oblastiach a rozdiel uhlov je najviac (90°).

Obrázok 4.4 zobrazuje príklad vektorov, kde východzím bodom je *Adresa 2*. Hodnota *Area* pre Adresu 1 a Adresu 2 bude pozostávať z dvoch vektorov, pretože Adresa 2 spadá do oblastí *UP RIGHT* a *RIGHT*, čomu odpovedajú vektory 0x04 a 0x10. Výsledná bitmapa vektorov bude mať hodnotu: *0x04 + 0x10 = 0x14*, čo v desiatkovej sústave predstavuje hodnotu 20.



Obrázok 4.4: Vektory fráz

Vyhľadávací záznam pre určenie podobnosti dokumentov je definovaný ako trojica *(fráza1, area, fráza2). Fráza 1* a *fráza 2* sú anotované frázy dokumentu a *area* je *intervalový rozsah okolia vrcholu*, do ktorého spadá *fráza 2*. Pre urýchlenie vyhľadávania sa odfiltrujú frázy, ktoré majú menšiu pravdepodobnosť (napríklad menšiu ako 0.9). Frázy, ktoré ostali po odfiltrovaní sa označujú ako **ImportantPhrase** a len tie tvoria *Vyhľadávacie záznamy*.

Výpočet zhody vyhľadávacích záznamov:
Vyhľadávací záznam A dokumentu X je zhodný s vyhľadávacím záznamom B v dokumente Y, ak sú splnené všetky rovnosti:

- A.fráza1.annotationType = B.fráza1.annotationType
- A.fráza1.typeOccurence = B.fráza1.typeOccurence
- A.area & B.area != 0 (& je bitový operátor)
- A.fráza2.annotationType = B.fráza2.annotationType

- A.fráza2.typeOccurence = B.fráza2.typeOccurence

Teda dva vyhľadávacie záznamy sú zhodné, pokiaľ sú odpovedajúce si fráze rovnakého typu, v dokumente majú rovnaké poradie a ich vzájomná poloha je rovnaká (alebo podobná).

Výsledná **miera zhody** skúmaného dokumentu A a dokumentu B, je rovnako ako v modeli, daná podielom:

$$similarity(A,B) = \sqrt{\frac{pocet\,zhodnych\,zaznamov\,v\,A\,a\,B}{Max(pocet\,vyhladavacich\,zaznamov\,v\,A\,a\,B)}} \tag{4.1}$$

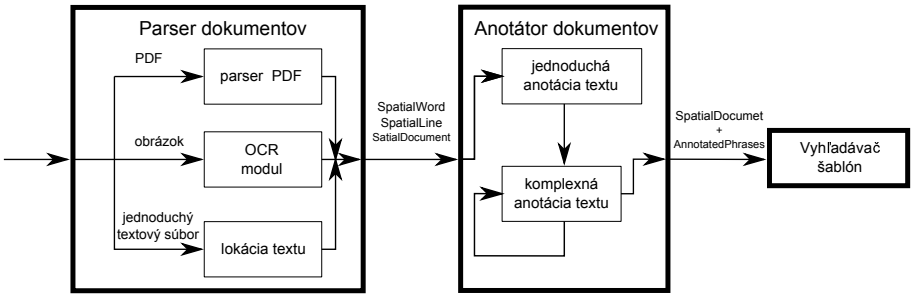
4.3 Schéma aplikácie

Celá aplikácia pozostáva z troch častí:

- **Parser dokumentov** zo vstupného dokumentu extrahuje text a spolu s jeho lokáciou, z ktorého sú vytvárané slová *SpatialWord*. Z nich sa následne vytvorí dokument **SpatialDocument**, ktorý si slová interne porozdeľuje na *SpatialLine*.

- **Anotátor dokumentov** v *SpatialDocumente* vyhľadá podľa anotačných pravidiel slová, ktorým pridelí význam a pravdepodobnosť.

- **Vyhľadávač dokumentov** následne podľa anotovaných fráz vyhľadá podobný dokument a vypočíta zhodu, na koľko sú tieto dokumenty podobné. Z nájdeného podobného dokumentu potom vyhľadávač určí, o aký typ dokumentu sa jedná a ktoré anotované frázy sú dôležité.



Obrázok 4.5: Schéma aplikácie

4.4 Parser dokumentov

Hlavným účelom tohoto modulu je extrahovať z dokumentu text spolu s jeho súradnicami. Súradnice sa priraďujú jednotlivým slovám, riadkom a celkovému dokumentu.

Pre prevod z rôznych typov dokumentov (txt, pdf, jpg, png alebo bmp) sú použité rôzne moduly. Napríklad pre extrakciu textu z obrázku sa používa OCR[3]

[3]OCR je zkratka Optical character recognition. Je to metóda, ktorá z obrázku extrahuje text

modul, pre PDF dokument sa použije modul pre prevod PDF do obrázku, ktorý je následne poslaný do OCR modulu[4], alebo textový súbor, je odoslaný do jednoduchého modulu, ktorý len pridá informáciu o umiestnení slov, či riadkov v dokumente.

4.4.1 Prevod z textového súboru

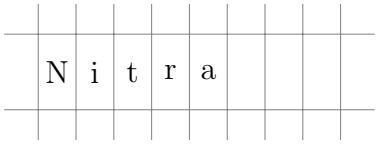
Modul pre prevod textového súboru do *SpatialDocument*u má niekoľko dôvodov.

- takmer každý dokument (rtf, doc, docx, xml) sa ľahšie prevádza do textového súboru než do obrázku
- väčšina OCR aplikácií umožňuje prevod obrázka do textového súboru, čo dáva možnosť používať vlastné OCR namiesto *Tesseract OCR*, ktorý je zabudovaný do aplikácie.
- vytvorený textový súbor sa veľmi rýchlo prevádza do *SpatialDocument*u, čo pri vývoji a testovaní aplikácie ušetrilo veľa času

Modul rozdelí celý obsah textového dokumentu na jednotlivé slová a riadky, ktorým následne priradí súradnice, čím z nich vytvorí *SpatialWord* a *SpatialLine*.

Vytvorenie SpatialWord a SpatialLine

Ak by bol celý textový dokument rozdelený po znakoch na tabuľku, kde jedena bunka tabuľky obsahuje jeden znak textového súboru, potom súradnice slova sú určené riadkom a stĺpcami tabuľky, v ktorých sa nachádzajú jednotlivé znaky slova. Príklad slova je ilustrovaný na obrázku 4.6, kde slovo je tvorené piatimi znakmi v piatich bunkách tabuľky.



Obrázok 4.6: Lokácia slova

Nech každý znak (a teda každá bunka tabulky) je 24 pixelov vysoký a 12 pixelov široký [5], potom dokument je možne virtuálne previesť do tlačenej podoby a slovám dať adekvátnejšie súradnice. Samozrejme tieto rozmery je možné zmeniť ľubovolne, pretože na funkčnosť to nebude mať vplyv, ale dať znakom takýto rozmer je lepšie pre predstavu, než nechávať znaky (respektíve bunky tabuľky) o veľkosti 1x1 pixel.

Slová *SpatialWord* sa teda jednoducho vytvoria tak, celý dokument je prehľadávaný po riadkoch a tie po slovách. Každému slovu je vytvorené *SpatialWord* so súradnicami

[4]existujú knižnice, ktoré dokážu extrahovať text s PDF dokumentu, ale nedokážu to ak je tvorený výhradne obrázkami. V tom prípade je opäť potrebné extrahovať obrázky do súborov, nad ktorými sa následne spustí OCR aplikácia

[5]rozlíšenie znaku 12x24 pixelov je často používaným v ihličkových, alebo termo-tlačiarniach, preto práve tento rozmer

vypočítanými podľa vzorca v kóde 4.1, kde *wordStart* je pozícia, kde slovo začína a *wordSize* je počet znakov v slove:

Kód 4.1: Zmiešaná definícia

```
1  leftTop.x = (wordStart) * 12
2  leftTop.y = row * 24
3  rightDown.x = (wordStart+wordSize) * 12
4  rightDown.y = (row+1) * 24
```

Tie *SpatialWord*, ktoré majú medzi sebou medzeru menšiu ako 3 bunky, sa spoja do jedného riadku *SpatialLine*.

4.4.2 Parser PDF dokumentov

PDF dokumenty môžu obsahovať obrázky, ktoré často bývajú výstupom skenerov, avšak parser PDF dokumentov sa zameriava len na extrakciu textovej časti. Pre extrakciu je použitá knižnica PDFBox [6] od Apache. Tá poskytuje okrem textu slov aj ich súradnice. Z nich sú vytvorené *SpatialWord* a následne *SpatialDocument*, ktorý je výstupom parseru.

4.4.3 OCR modul

OCR modul má za úlohu zo vstupných obrázkov vyextrahovať text a určiť jeho umiestnenie v dokumente. Pre extrakciu textu z obrázka bolo ako najlepšie riešenie zvolený open-source engine Teseract OCR. Okrem podpory viacerých jazykov je možné naučiť Tesseract OCR tiež nove znakové sady a hlavne pri extrahovanom texte tiež uvádza súradnice v obrázku, odkiaľ text pochádza. Výstupom tohoto modulu je množina slov *SpatialWord*, z ktorých je vytvorený *SpatialDocument*.

Je dôležité, aby text, ktorý je vyextrahovaný z obrázka mal čo najmenšiu chybovosť. Anotátor by totiž chybné slovo nerozpoznal a dôležitá fráza by sa tak nemusela v dokumente nájsť. Preto sa nasledujúce sekcie venujú vylepšeniu rozpoznania textu predspracovaním obrázka a rozdelením obrázka na menšie časti.

4.4.3.1 Predspracovanie pred OCR

Keďže v praxi nie sú všetky dokumenty dokonale oskenované, dochádza najčastejšie k pootočeniu dokumentu. Tesseract nie je úplne spoľahlivý a sami autori doporučujú, aby bol obrázok predspracovaný, tj. aby bol dokument vyrovnaný, aby bol odfiltrovaný šum a tak podobne. Pokusy naviac ukázali, že keď tesseract zpracováva menšie časti obrázku (napríklad pár riadkov), má lepšie výsledky než pri rozoznávaní celého dokumentu. Táto časť diplomovej práce sa preto zameriava aj na to, ako najviac pomôcť Tesseractu pri extrahovaní textu z obrázkov. Vylepšenia spracovania obrázku zahrňuje vyrovnanie (takzvane *descew* obrázka), rozsekanie na menšie časti a následné zloženie obrázku.

[6] voľne dostupná v MAVEN repozitári, ako groupID: org.apache.pdfbox, artifactId: pdfbox, version: 2.0.2

4.4.3.2 Vyrovnanie obrázka

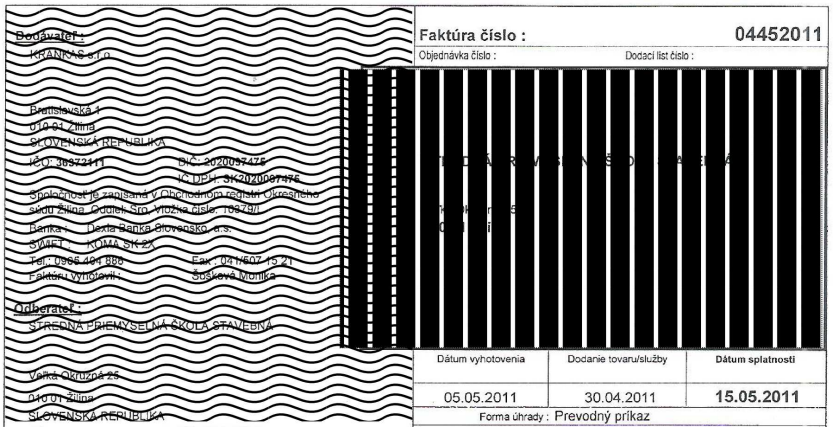
Oskenované dokumenty bývajú často pootočené o pár stupňov. Pre vyrovnanie obrázka bola použitá Houghova transformácia [1]. Základná Houghova transformácia bola určená pre detekciu priamok v obrázku. Rozšírená verzia dokáže na obrázku detekovať ľubovolné tvary, ako kruhy alebo elipsy. Pre účely tejto práce úplne postačila jednoduchá verzia. Pri implementovaní bola použitá existujúca implementácia [2] vyvinutá v jazyku Java, ktorej autor je ale neznámy (bol uvedený len jeho pseudonym *Anydoby*). Implementácia vyrovnania obrázka pozostáva z dvoch fáz. V prvej sa na obrázku detekuje uhol, o ktorý je oskenovaný dokument pootočený a v druhej fáze je pôvodný obrázok otočený naspäť o zistený uhol. Centrom rotácie je samozrejme stred obrázka.

4.4.3.3 Rozsekanie obrázka

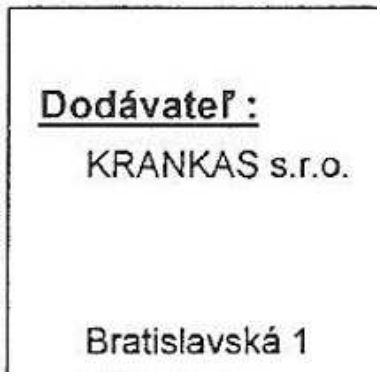
Tesseract OCR má problém pri extrahovaní textu z veľkých obrázkov. Pri pokusoch vracal oveľa lepšie výsledky v prípadoch, keď extrahoval text len z častí obrázka. Tieto boli ručne vybrané a uložené do separátneho súbora. Tieto časti väčšinou obsahovali bloky textu.

Pre segmentáciu dokumentu existuje mnoho spôsobov. Medzi najčastejšie používané algoritmy patria X-Y cutting, Voronoi, Whitespace alebo Docstrum. V práci [3] sú tieto algoritmy porovnávané. I keď Voronoi a Docstrum dávajú dobré výsledky a dokážu spracovať aj zakrivené dokumenty, pre účely rozsekania obrázka najlepšie poslúži X-Y cutting algoritmus v kombinácii s detekovaním hrán.

Výsledkom X-Y cutting algoritmu sú obdĺžniky, ktoré sa dajú z obrázka jednoducho vybrať do samostatného súboru. Iné algoritmy môžu mať za výsledok výrezy v tvare písmena L, s čím sa oveľa horšie pracuje, pretože pri vyrezávaní (kde sa vyrezáva obdĺžnik) je nutné prekryť oblasti z iných výrezov. Príklad je uvedený na obrázku 4.7, kde je problémová oblasť označená vlnkami. Do nej zasahuje pruhovaná oblasť a pri výreze by ju bolo nutné prekresliť, aby sa nezpracovávala *Tesseract*om. Detekcia hrán v dokumente nám umožní lepšie detekovať *biele miesta* a zároveň umožní detekciu čiar, ktoré môžme použiť ako oddeľovače blokov textu.



Obrázok 4.7: Problém s vyrezávaním



Obrázok 4.8: Originánly obrázok

Obrázok 4.9: Detekované hrany

Detekcia hrán

Pre detekovanie hrany sme použili Canny Edge detector[4]. Algoritmus väčšinou pozostáva zo 4. krokov:

- Eliminácia šumu pomocou Gaussového filtra
- Určenie gradientu (veľkosť a smer) z prvej derivácie
- Nájdenie lokálnych maxím
- eliminácia nevýznamných hrán pomocou dvojitého tresholdu (s hornou a dolnou hranicou)

Na obrázku 4.8 je zobrazený pôvodný dokument. Jeho detekované hrany sú zobrazené na obrázku 4.9.

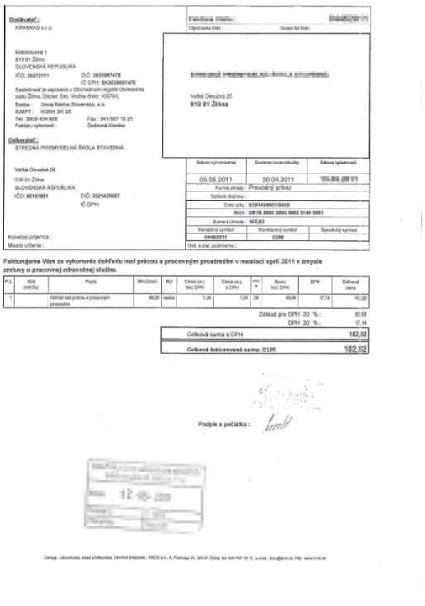
Jeho implementáciu vytvoril Tom Gibara[5], ktorá bola s menšími úpravami použitá v tejto práci. Bolo nutné pridať podporu formátu obrázka 4-bytového ABRG. Pre dvojitý treshlold bola nastavená dolná hranica na 0.9 (spodný treshold) a hornú na 1.0 (horný treshold)[7]. Táto kombinácia pri testovaní na oskenovaných dokumentoch vracala najlepšie výsledky.

Detekcia deliacich čiar

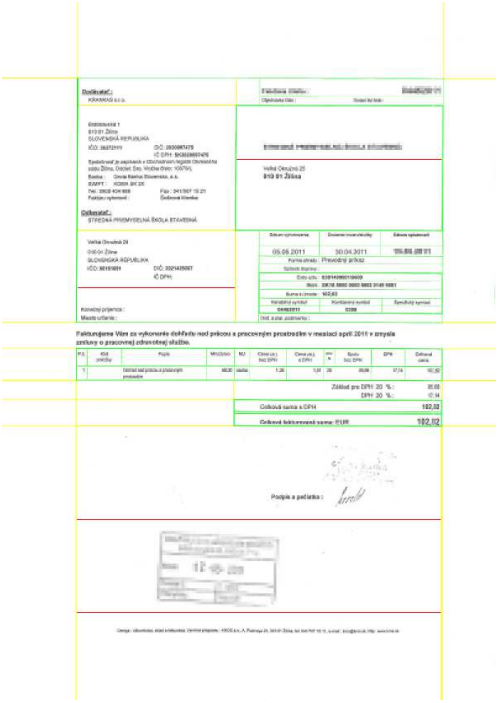
Deliacimi čiarami sa označujú horizontálne alebo vertikálne čiary, ktoré dokument rozdeľujú do blokov. Je ale dôležité, aby nenarušili súvislosť textu. Napríklad aby nerozdeľovali slovo, alebo jeden riadok na dve časti.

Deliace čiary vychádzajú z dvoch zdrojov. Jedným zdrojom sú čiary priamo vytlačené v dokumente, ktoré sa označujú ako **Pritned Dividers**. Väčšinou tvoria tabuľky alebo horizontálne a vertikálne čiary, ktoré oddeľujú časti dokumentu (napríklad hlavičku dokumentu od zbytku).

[7]Ak hodnota daného pixelu je väčšia ako horný treshold, je pixel označený priamo ako hranový. Ak je graditent medzi horným a dolným tresholdom, potom je uznaný ako hranový, pokiaľ susedí s bodom, ktorý už je označený ako hranový.



Obrázok 4.10: Dokument



Obrázok 4.11: Dokument a jeho deliace čiary

Predĺžením *Pritned Divider*, až po okraj dokumentu, alebo po najbližšiu *Pritned Divider*, by vznikla ďalšia deliaca čiara. Takéto deliace čiary sa označujú ako **Extended Printed Divider**. Podmienkou je, aby žiadna *Extended Printed Divider* nekrižovala inú *Pritned Divider* alebo text.

Druhým zdrojom deliacich čiar sú biele miesta v dokumente. Biele miesta v dokumente tvoria prázdnu oblasť dokumentu, kde nie je nič "vytlačené". Deliace čiary, ktoré vychádzajú z bielych miest v dokumente, budú označené ako **White Space Divider**.

Na obrázku 4.10 je zobrazený dokument. Obsahuje tabuľky, polo-štrukturovaný text, ktorý je naviac orámovaný. Obrázok 4.11 ilustruje, ako by takýto dokument mohol byť rozdelený podľa deliacich čiar. Zelené čiary predstavujú *Pritned Dividers*. Ich predĺžením sme dostali *Extended Printed Dividers*, ktoré sú zobrazené žltou farbou. *White Space Dividers* sú zobrazené červenou.

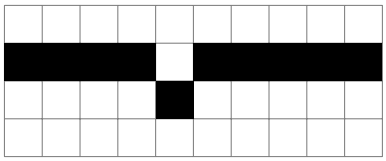
Pri hľadaní deliacich čiar budeme v prvom kroku hľadať *Pritned Dividers*.Takéto čiary ale nie vždy končia na okraji. Preto v druhom kroku sa pokúsime nájdené hrany predĺžiť skrz biele miesta na maximum, čím dostaneme *Extended Printed Dividers*. V treťom kroku sa budeme snažiť rozdeliť vzniknuté obdĺžniky skrz ich biele miesta, čím vytvoríme *White Space Dividers*.

Detekcia čiar na obrázku

Obrázok s detekovanými hranami obsahuje len biele a čierne body. Biele body predstavujú pozadie (respektíve oblasť, kde sa farba nemení), čierne nájdené

zmeny farby, to jest miesta, kde sa mení farba pozadia a farba textu/čiar[8]. Tento obrázok bude v tejto práci používaný ako základný obrázok pre detekciu všetkých deliacich čiar a označuje sa ako **Základný obrázok**.

Algoritmus pre nájdenie *Pritned Dividers* prehľadáva všetky čierne body v *základnom obrázku* a skúša, akú najdlhšiu horizontálnu a vertikálnu čiaru je možne z daného bodu nájsť. Pretože obrázok nie je dokonalý a obsahuje nerovnosti, je potrebné predpokladať, že rovná čiara ma občas posunutý pixel, ako napríklad na obrázku 4.12.



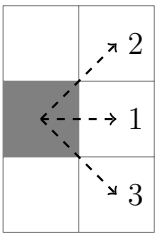
Obrázok 4.12: Nerovnosť hrany

Preto pri hľadaní horizontálnej čiary zľava doprava, najprv algoritmus skontroluje, či od aktuálneho pixelu je vpravo tiež čierny bod. Ak áno, pridá ho do aktuálnej horizontálnej čiary a pokračuje, inak skontroluje diagonálne (v smere doprava) susediace body a ak aspoň jeden je čierny, pridá ho do aktuálnej horizontálnej čiary a pokračuje v hľadaní. Priorita pixelu označuje susedný pixel, ktorý pri prehľadávaní horizontálnej/vertikálnej čiary má prednosť. Priority susedných pixelov sú zobrazené na obrázkoch 4.13 a 4.14. Pri hľadaní vertikálnej čiary postupuje obdobne ako pri hľadaní horizontálnej čiary s tým, že sa prehľadáva zhora nadol.

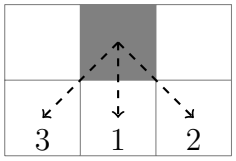
Hrany, ktoré majú dĺžku 1 pixel sú odstránené. Obrázok 4.15 zobrazuje detekované hrany, ktoré sme rozdelili na vertikálne a horizontálne. Na obrázku 4.16 sú nájdené horizontálne hrany zobrazené červenou a vertikálne zelenou farbou. Žlté pixely zobrazujú prieniky vertikálnych a horizontálnych hrán.

Je vidieť, že aj znaky textu obsahujú horizontálne a vertikálne čiary. Pre rozlíšenie, ktoré hrany sú deliacimi a ktoré sú čiarami v znakoch je použitá minimálna dĺžka v danom smere. Minimálna dĺžka v horizontálnom smere je na-

[8]V zdrojových kódoch je to opačne. Čierny pixel predstavuje bod, kde sa farba nemení, a biely, resp. nečierny bod označuje zmenu farby. Pretože biele miesto v pôvodnom obrázku a čierne miesto v obrázku s rozoznanými hranami je tá istá oblasť, bolo by pri výklade mätúce používať obe tieto označenia naraz. Preto pre potreby tejto práce boli v obrázku s rozoznanými hranami prehodené biely za čierny pixel, aby *biela oblasť* bolo spoločné označenie v oboch obrázkoch.



Obrázok 4.13: Poradie pri hľadaní ďalšieho pixelu pre horizontálnu čiaru



Obrázok 4.14: Poradie pri hľadaní ďalšieho pixelu pre vertikálnu čiaru

Obrázok 4.15: Detekované hrany

Obrázok 4.16: Detekované hrany rozdelené na horizontálne a vertikálne

stavená na 1/10 z šírky dokumentu a minimálna dĺžka vo vertikálnom je 1/10 z výšky dokumentu. Predpokladom je, že dokument nebude obsahovať znak, ktorý by zaberal priestor aspoň 1/10 šírky, alebo výšky dokumentu. Na druhú stranu čiary, ktoré by boli kratšie ako 1/10 šírky, alebo výšky, zrejme oddeľujú veľmi malý blok textu.

Na obrázkoch 4.8 a 4.9 je vidieť, že *Printed Dividers* sa po detekcií hrán prejavujú ako dve čiary [9], respektíve ako krivka, ktorá obaľuje pôvodnú čiaru. Tieto čiary je potrebné zjednotiť, aby sa dokument nerozdeľoval podla dvoch, tesno susediacich čiar, ktoré by navyše v ďalších krokoch[10] mohli spôsobiť problémy.

Získanie čiar *Printed Dividers* pozostáva z niekoľkých krokov:

1. prefiltrovanie hrán: hranu obalíme do čo najmenšieho obdĺžnika. Obdĺžnik musí byť široký minimálne 1/10 šírky dokumentu, alebo vysoký 1/10 z výšky dokumentu, pričom menší z pomerov šírka:výška a výška:šírka nesmie má byť menší ako 1:10 [11].

2. roztriedenie hrán do dvoch množín: na vertikálne a horizontálne

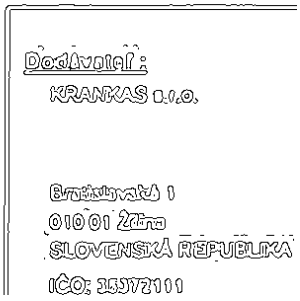
3. zjednotenie susediacich hrán: Zvlášť pre vertikálne a zvlášť pre horizontálne čiary. Zjednotenie vertikálnych čiar sa spočíta (pre horizontálne čiary obdobne):

   - zoradenie čiar podľa x-ovej súradnice
   - meranie vzdialeností: postupne pre každú vertikálnu čiaru sa preskúmajú, v smere doprava, susedné vertikálne čiary (maximálne do vzdialenosti

[9] detekcia hrán hľadá, kde sa mení farba, takže každá čierna krivka a čiara je teda ohraničená z oboch strán

[10] Napríklad pri predlžovaní čiar, keď sa dve kolmé čiary križujú v tvare ⊨ a horizontálna čiara by mohla byť predĺžená. V prípade, keby bola vertikálna čiara ponechaná ako dve, vonkajšia čiara (v tomto prípade tá vľavo) by zastavila pokus o predĺženie horizontálnych čiar smerom doľava.

[11] pomer 1:10 odfiltruje hrany s väčším sklonom a ponecháva hrany, ktoré sú mierne zakrivené, ako napríklad hrany, zaobleného obdĺžnika, ktoré sú na koncoch zakrivené



Obrázok 4.17: Pôvodné hrany

Obrázok 4.18: Zjednotené hrany

1/10 šírky dokumentu), ktoré prekrývajú vyšetrovanú čiaru vo vertikálnom smere. Namerané horizontálne vzdialenosti sa uložia. Vzdialenosť dvoch čiar sa počíta ako vzdialenosť obdĺžnikov, ktoré čiary obaľujú.

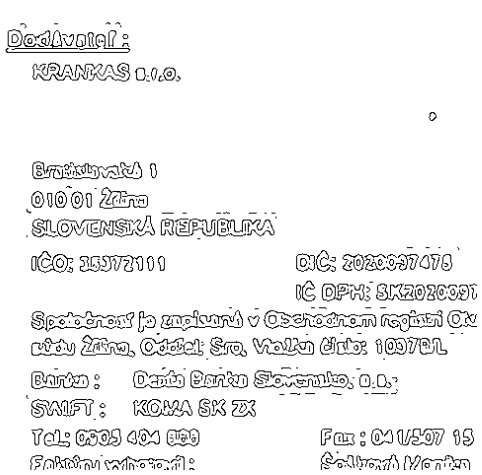
- z nameraných vzdialeností sa zistí maximálna používaná vzdialenosť, ktorá sa označuje ako **Susedská vzdialenosť**. V tomto prípade by ako maximálna vzdialenosť bola braná tá, ktorá by bola aspoň 3x zaznamenaná.
- zjednotenie susedných čiar podľa nájdenej maximálnej vzdialenosti: dve susediace čiary, ktoré majú horizontálnu vzdialenosť menšiu alebo rovnú ako *Susedská vzdialenosť* (tj. vzdialenosť obdĺžnikov, ktoré čiary obaľujú), sa spoja do jednej čiary: čiaru obaľujúci obdĺžnik je spočítaný ako minimálny obdĺžnik, ktorý obaľuje obe čiary (tj. z ich obdĺžnikov), z neho sa vytvoria body, ktoré vedú stredom obdĺžnika od horného okraju až po spodný okraj.

4. nahradenie pôvodných čiar zjednotenými čiarami: z obrázka sa odstránia body, ktoré tvorili pôvodnú čiaru, a nahradia ich body, ktoré tvoria zjednotené čiary.

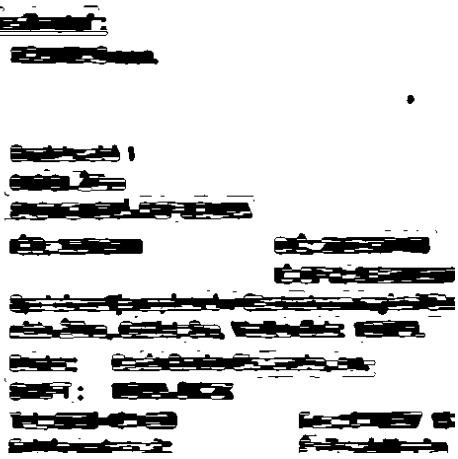
Na obrázku 4.17 a 4.18 sú zobrazené čiary pred zjednotením a po zjednotení.

Predĺžením takto nájdených čiar by bolo možne dostať adekvátne rozdelenie dokumentu na menšie bloky. Predĺženie v danom smere prebieha po bielych miestach dokumentu, dokiaľ nenarazí buď na znak, alebo na inú deliacu čiaru. Takáto čiara ale môže preseknúť jedno slovo na dve časti, pretože medzi jednotlivými znakmi sú biele miesta. To je samozrejme nechcené a preto ešte pred predlžovaním čiar je nutné vyplniť prázdne miesta medzi znakmi jedného slova.

Základný obrázok sa preto prehľadáva pixel po pixeli a počítajú sa dĺžky bielych horizontálnych čiar, ktoré sú menšie ako 1/10 šírky dokumentu. Najčastejšie sa opakujúca dĺžka určuje najbežnejšiu vzdialenosť medzi čiernymi pixelmi, čo v pôvodnom dokumente predstavuje šírku čiary znakov. Táto hodnota sa zväčší o



Obrázok 4.19: Pôvodné hrany znakov



Obrázok 4.20: Vyplnenie medzier medzi znakmi

2 pixely [12] a označí ako **Hrúbka písma**.

Pre zistenie najčastejšej vzdialenosti medzi znakmi je potrebné zistiť druhú najbežnejšiu vzdialenosť. Preto sa z nameraných vzdialeností odstránia tie, ktoré sú menšie ako *Hrúbky písma* a z týchto hodnôt sa vypočíta priemer. 1,5 násobok tejto hodnoty (pri pokusoch sa ukázalo, že 1,5 násobok spočítanej strednej hodnoty pokrýva väčšinu medzier medzi znakmi) bude označovaný ako **Znaková medzera**.

Základný obrázok bude opäť postupne prechádzaný pixel po pixeli a všetky nájdené horizontálne biele čiary, ktoré sú kratšie ako *Znaková medzera*, sa nahradia čiernymi pixelmi (ďalšie spracovanie bude tým pádom pokladať tieto pixely ako pixely patriace znaku). Tým sa vyplnia biele miesta nie len v znakoch, ale aj medzi jednotlivými znakmi z jedného slova. Obrázok 4.19 zobrazuje obrázok pred a obrázok 4.20 po vyplnení medzier medzi znakmi.

Predĺženie nájdených čiar

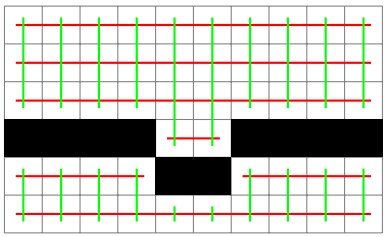
Predĺžením doposiaľ nájdených rozdeľovacích čiar *Printed Dividers* vznikajú *Extended Printed Dividers*, ktoré adekvátnejšie rozdeľujú dokument než čiary *White Space Divider*. Príkladom môže byť horizontálna čiara, ktorá končí 3 cm od okraja dokumentu (tj. nepretína dokument úplne), alebo tabuľka s neviditeľnými hranami, ktorá má orámovaný len posledný riadok.

Čiara *Printed Divider* predlžovaná dovtedy, pokiaľ algoritmus nenarazí na okraj dokumentu, alebo na *Printed Divider* čiaru. Čiara nie je predĺžená, ak by v danom smere pretínala znak. Môže ale pretínať iné *Extended Printed Dividers*, ako je vidieť na obrázku 4.11, kde *Extended Printed Dividers* sú zobrazené žltou farbou.

[12] predĺženie o 2 pixely sa pri pokusoch ukázalo ako dostatočné, aby bola obsiahnutá väčšina medzier v znakoch

Nájdenie rozdeľovačov v bielych miestach

Zo všetkých bielych pixelov v *Základnom obrázku* sa vytvoria vertikálne a horizontálne biele čiary. Teda v každom stĺpci bude toľko vertikálnych bielych čiar, koľko spojitých úsekov z bielych pixelov sa v ňom nachádza. Obdobne s riadkami a horizontálnymi čiarami. Príkladom je obrázok 4.21, kde sa nachádza 7 horizontálnych a 20 vertikálnych bielych čiar. Pre úsporu pamäte a ušetrenia výpočtu v následujúcich krokoch, odfiltruje z horizontálnych tie, ktoré sú kratšie ako 1/10 šírky dokumentu a z vertikálnych tie, ktoré sú kratšie ako 1/10 výšky dokumentu.



Obrázok 4.21: Čiary v bielych miestach

Rozsekanie podľa deliacich čiar

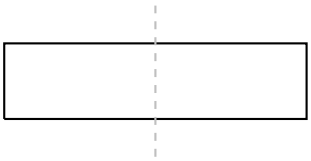
Dokument sa rekurzívne rozseká na menšie bloky pomocou získaných deliacich čiar *Printed Dividers*, *Extended Printed Dividers* a *White Space Divider*. Najprv sa z nich vyberie jeden kandidát (čiara) pre čo najlepšie rozdelenie bloku. Táto čiara je označovaná ako čiara **Divider Line**, rozdelí blok na dva a delí tiež všetky deliace čiary patriace bloku. Pri delení sa opäť odfiltrujú tie čiary, ktoré sú kratšie ako 1/10 rozmeru dokumentu. Rozmer 1/10 dokumentu udáva rozdelenie jedného dokumentu na maximálne 100 blokov. Pri hľadaní kandidáta z deliacich čiar má väčšiu prioritu typ čiary než to, či sú horizontálne, alebo vertikálne. Vyhľadáva sa najprv v čiarach *Printed Dividers* (označené skratkou PD), potom *Extended Printed Dividers*(označené skratkou ED) a nakoniec *White Space Divider*(označené skratkou WD).

Ak čiary, v ktorých sa hľadá *Divider Line*, sú horizontálne budú sa značiť s prefixom H, ak vertikálne, tak s prefixom V. Napríklad horizontálne čiary *Extended Printed Divider* sa budú označovať ako HED (ak by boli vertikálne, tak VED).

Druhou prioritou, ako by sa mal blok deliť, je rozmer bloku. V prípade, ak je blok širší než vyšší, mal by sa rozseknúť vertikálne, ako je na obrázku 4.22. Naopak ak je vyšší než širší, mal by sa rozseknúť horizontálne, ako na obrázku 4.23.

Preto ak je blok širší, bude sa *Divider Line* hľadať v deliacich čiarach v tomto poradí: VPD, HPD, VED, HED, VWD, HWD. Ak je blok vyšší, tak v tomto poradí : HPD, VPD, HED, VED, HWD, VWD.

Pri prehľadávaní každej skupiny má väčšiu prioritu tá čiara, ktorá je najbližšie stredu:



Obrázok 4.22: Vertikálne rozdelenie

Obrázok 4.23: Horizontálne rozdelenie

Dodávateľ :
KRANKAS s.r.o.

Bratislavská 1
010 01 Žilina
SLOVENSKÁ REPUBLIKA
IČO: **36372111** DIČ: **2020097475**
IČ DPH: **SK2020097475**
Spoločnosť je zapísaná v Obchodnom registri Okresného súdu Žilina, Oddiel: Sro, Vložka číslo: 10379/L
Banka : Dexia Banka Slovensko, a.s.
SWIFT : KOMA SK 2X
Tel.: 0905 404 888 Fax : 041/507 15 21
Faktúru vyhotovil : Šošková Monika

Odberateľ :
STREDNÁ PRIEMYSELNÁ ŠKOLA STAVEBNÁ

Obrázok 4.24: Pôvodný obrázok

Dodávateľ :
KRANKAS s.r.o.

Bratislavská 1
010 01 Žilina
SLOVENSKÁ REPUBLIKA
IČO: **36372111** DIČ: **2020097475**
IČ DPH: **SK2020097475**
Spoločnosť je zapísaná v Obchodnom registri Okresného súdu Žilina, Oddiel: Sro, Vložka číslo: 10379/L
Banka : Dexia Banka Slovensko, a.s.
SWIFT : KOMA SK 2X
Tel.: 0905 404 888 Fax : 041/507 15 21
Faktúru vyhotovil : Šošková Monika

Odberateľ :
STREDNÁ PRIEMYSELNÁ ŠKOLA STAVEBNÁ

Obrázok 4.25: Nájdené rozdelenie obrázka

1. všetky čiary sa zoradia podľa vzdialenosti od stredu bloku (tj. čiara, ktorá by viedla stredom, by bola prvá)

2. ak výsledný zoznam nie je prázdny, vyberie sa prvá a tá sa stane čiarou *Divider line*

Ak v danom bloku bola nájdená *Divider line*, rozdelíme ho podľa nej na dva a zároveň aj všetky ich deliace čiary (tj. PD, ED a WD) . Ak sú vzniknuté bloky dostatočne veľké a obsahujú aspoň nejakú deliacu čiaru, rekurzívne sa rozdelia podľa ich deliacich čiar (tj. nájsť v nich *Divider line* a podľa nich ich rozdeliť).

Obrázok 4.24 ukazuje, ako vyzeral pôvodný dokument a obrázok 4.25 zobrazuje, ako bol dokument rozdelený. Je vidieť, že deliace čiary sa pridržujú tých vytlačených, a čo je dôležité, nerozdeľujú slová na dve časti.

4.4.3.4 Výsledné poskladanie

Časti obrázka, na ktoré bol dokument rozsekaný, sa uložia ako jednotlivé obrázkové súbory, pričom musia byť uložené súradnice obdĺžnika, z ktorých bol obrázok vyrezaný. Nad týmito obrázkami je následne spustený Tesseract OCR, ktorý rozoznaný text uloží ako HTML súbor. Ten obsahuje slová spolu s ich súradnicami. Z neho sa vytvoria slová *SpatialWord*, ktoré sa musia odsadiť o súradnice obdĺžnika vyrezaného z pôvodného obrázka.

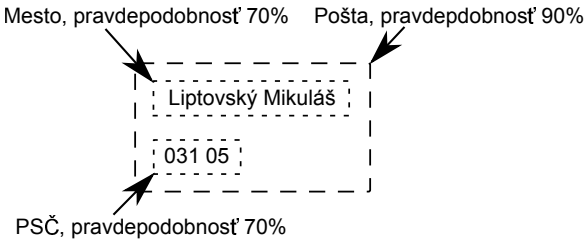
Z týchto slov sa následne vytvorí *SpatialDocument*, ktorý je výsledkom parsovania.

4.5 Anotátor dokumentov

Základným prvkom, s ktorým pracuje anotátor dokumentov je **fráza**. Tvoria ju slová, ktoré sa môžu nachádzať na rôznych miestach v dokumente. Väčšinou sa ale jedná o prípady, kedy sa slová nachádzajú vedľa seba, alebo pod sebou. Anotátor podľa užívateľom definovaných pravidiel nachádza v dokumente fráze a následne im priraďuje význam a pravdepodobnosť, že daná fráza skutočne odpovedá priradenému významu. Spôsob, ako užívateľ definuje pravidlá a ako sa počíta pridelená pravdepodobnosť je popísané v kapitole Definícia anotácie.

Príkladom môže byť číslo *031 05*. To podľa užívateľom nadefinovaného pravidla, by malo byť *PSČ* s pravdepodobnosťou 20% . Číslo ale môže byť súčasťou telefónneho čísla napríklad *00421 55 031 05*. To, či je číslo skutočne *PSČ* závisí od text v jeho okolí. Ak by sa naľavo, alebo nad číslom *031 05* nachádzalo mesto, pravdepodobnosť, že by sa jednalo skutočne o *PSČ*, by bola väčšia. To ale závisí od nadefinovaných pravidiel.

Fráza, ktorá má priradený význam a pravdepodobnosť sa označuje ako **anotovaná fráza**. Obrázok 4.26 zobrazuje hneď niekoľko anotovaných fráz. Jednou je *Liptovský Mikuláš*, ktorá bola anotovaná ako typ *Mesto* s pravdepodobnosťou 70%. Druhou je číslo *031 05*, ktorá bola anotovaná ako typ *PSČ* s pravdepodobnosťou 70%. Tretia fráza sa skladá z týchto dvoch fáz, je označená ako typ *Pošta* s pravdepodobnosťou 90%.



Obrázok 4.26: Anotované fráze

V súčasnosti existuje mnoho anotačných nástrojov, ale väčšina sa sústreďuje na anotáciu textu bežného jazyka a prevod textov do RDF dát[6], alebo sú súčasťou komerčných produktov. Preto bolo nutné vyvinúť vlastný engine pre anotovanie polo-štrukturovaných dokumentov.

***Polo-štrukturovaný dokument**, alebo dáta, je forma štrukturovaných údajov, ktoré nie sú v súlade s formálnou štruktúrou dátových modelov, spojených s relačnou databázou, alebo s inou formou dátových tabuliek, ale napriek tomu obsahujú značky, alebo iné prvky, ktoré vytvárajú hierarchiu záznamov a dátových polí*[7].

Zjednodušene sú to dáta, ktorých polia (v angličtine *fields*, v tejto práci označované ako *fráze*) sú popísané nejakou značkou. Príkladom je *"tel.: +420 111 222 445"*, kde fráza je telefónne číslo a značkou je *"tel:"*. Značkou môže ale byť aj poloha frázy v dokumente, respektíve to, vedľa akých ďalších fráz sa nachádza.

4.5.1 Anotácia fráz

Anotácia frázy znamená priradenie významu jednému, alebo viacerým slovám, ktoré spolu nejakým spôsobom súvisia. Význam frázy sa môže meniť v závislosti

na kontexte, umiestnení alebo jeho zvýraznení [13]. Automatickému anotovaniu fráz a textu sa venuje celé vedecké odvetvie a je veľmi komplikované.

V tejto práci sa využíva toho, že dáta sú polo-štrukturované a kontext nie je potrebné pochopiť úplne. Navyše v takýchto dokumentoch má každá významná fráza svoj pomerne jednoducho definovateľný kontext, alebo pre určenie významu frázy nie je potrebný kontext.

Anotovanie fráz v tejto práci, je založené na pravidlách, ktoré majú v sebe 3 základné položky:

- *pattern* - jednoznačne definovaný výraz pre nájdenie slova/skupiny slov
- *type,probability* - priradenie typu a pravdepodobnosti nájdenému slovu/skupine slov
- *context*- dodatočné zvýšenie pravdepodobnosti v prípade, že sa v okolí slova/slov nachádza fráza konkrétneho typu

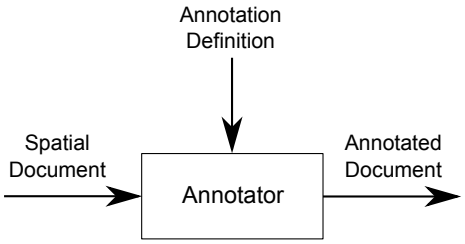
Konkrétnym definíciám a príkladom sa venuje kapitola Definícia anotácie.

V implementácii sa ako *pattern* používajú:

- konštanty - môžu byť menovito definované, alebo ako stĺpec v tabuľke. Naviac je možné definovať či porovnávanie je *Case Sensitive*[14], *Case Insensitive*[15], alebo len veľkosť prvého znaku musí byť v oboch prípadoch zhodná[16].
- regulárne výrazy
- nájdené frázy - používa sa pri komplexných anotáciách, kde výsledná fráza pozostáva s niekoľkých anotovaných fráz[17].

Context definuje podmienku: ak sa v nejakom okolí od aktuálnej frázy nachádza fráza typu X, potom zvýši pravdepodobnosť aktuálnej frázy o Y. Viac o contexte je popísané v sekcii Kontext.

O nájdenie výrazov v textovom dokumente sa stará trieda **Annotator**.



Obrázok 4.27: Annotator

[13]Napríklad text je veľkým tučným písmom

[14]dva odpovedajúce si reťazce musia mať naviac rovnaké veľké a malé písmena

[15]pri porovnávaní na veľkosti znakov nezáleží

[16]často sa nedopatrením užívateľov píšu dvojslovné názvy s veľkými, alebo malými znakmi. V dokumente sa potom objavuje napríklad názov ulice *Veľká Okružná* s veľkým *O*, a v inom dokumente *Veľká okružná* s malým *o*.

[17]príkladom je adresa, ktorá pozostáva z mesta, PSČ a ulice s číslom domu

4.5.2 Definícia anotácie

V súbore *config/annotate.xml* je možné definovať vlastné anotovanie fráz. Tento súbor sa označuje ako **Anotačný súbor**. Definície anotácií sa delia na dva typy: **Jednoduchá anotácia** a **Komplexná anotácia**. Princíp anotovania je postavený na:

- hľadaní vzorov v texte
- hľadaní už anotovaných fráz v nejakom okolí skúmaného textu
- na postupnom zoskupovaní malých anotovaných fráz do väčších celkov

Napríklad ak je v texte meno ulice a naľavo od neho je číslo, spolu tvoria adresu v ulici. Ak sa pod týmito dvoma frázami naviac nachádza mesto a PSČ, zoskupením týchto fráz je možné vytvoriť konkrétnu adresu.

4.5.2.1 Jednoduchá anotácia

Jednoduchá anotácia je základným prvkom a v podstate je nezávislá na ostatných anotáciách. Musí obsahovať pattern[18], alebo textovú konštantu, typ a pravdepodobnosť, s akou nájdený text odpovedá typu anotácie. Príklad, ako definovať jednoduchý typ je zobrazený v kóde 4.2.

```
1  <simpleType type="city">
2    <const prob="0.5" value="Žilina" matchCase="false"/>
3    <const prob="0.5" value="Bratislava">
4        <context type="zip" prob="0.4" area="row" distance="next" />
5    </const>
6  </simpleType>
```

Kód 4.2: Príkad jednoduchej anotácie

Na riadku 1. je typ anotácie **type** pomenovaný ako *city*. Na riadku 2. a 3. je definované, čo a ako sa ma vyhľadávať a v prípade nájdenia frázy v texte, aká pravdepodobnosť sa má fráze pridadiť. V tomto prípade, ak by anotátor našiel v texte slovo *"Bratislava"*, z odpovedajúceho slova vytvorí frázu typu "city"s pravdepodobnostnou hodnotou 0.5.

Definácia kontextu, ako je na riadku 4., zvyšuje pravdepodobnosť nájdenej frázy. Ak anotátor v jej okolí nájde frázu typu *zip* definovanú v okolí slova *"Bratislava"*, ktorá je najďalej vo vzdialenosti *next* (čo predstavuje susedné slovo/riadok), bola by pravdepodobnosť toho, že slovo "Bratislava"je skutočne typu *city*, zväčšená o 0.4 podľa vzorca 4.2. Teda 0.5+(1-0.5)*0.4 = 0.7 . Viac o atribútoch a hodnotách kontextu je popísané v sekcii Kontext.

Definície typov sa medzi sebou neprepisujú, ale pridávajú. Ekvivalentnou definíciou kódu 4.2 je kód 4.3.

```
1  <simpleType type="city">
2     <const prob="0.5" value="Žilina" matchCase="true"/>
3  </simpleType>
4  <simpleType type="city">
5     <const prob="0.5" value="Bratislava">
6         <context type="zip" prob="0.4" area="l,d" distance="next" />
```

[18]Vzor, ktorý definuje množinu reťazcov, ako napríklad regulárny výraz.

```
7       </const>
8     </simpleType>
```

Kód 4.3: Príkad jednoduchej anotácie

4.5.2.2 Konštanty

Konštanta definuje ako presne vyzerá slovo, ktorému priradí kontext. Príklad definície je uvedený v kóde 4.2. Atribútmi konštanty sú:

- prob - pravdepodobnosť, že nájdené slovo odpovedá typu anotácie. Obsahuje čísla v rozsahu od 0.0 po 1.0
- value - hodnota konštanty
- matchCase - nepovinný atribút - indikuje, či v nájdenom slove musia byť zhodné aj malé a veľké písmená. Môže mať hodnotu *true* a *false*. False je východzia hodnota. V niektorých prípadoch je potrebné, aby záležalo na veľkosti len u prvého znaku. V tom prípade sa nastavuje hodnota *firstletter*.

4.5.2.3 Zoznam z databázy

Pre prehľadnosť konfiguračného súbora a jednoduchú prácu s ním, sa konštanty môžu ukladať aj do databázy[19]. Príklad definície je uvedený v kóde 4.4

```
1 <simpleType type="city">
2   <db table="city_names" where="name␣like␣'A%'" valuecol="name"
3       probCol="prob" matchCase="true"/>
4   <db table="all_cities" probcol="prob" valuecol="name"  />
5 </simpleType>
```

Kód 4.4: Konštanty z databázy

Atribútmi sú:

- table - tabuľka, z ktorej sa budú vyberať konštanty
- where - nepovinný atribút - filtračná podimenka v SQL dotaze. stĺpec
- valuecol - stĺpec, ktorý obsahuje hodnoty konštánt
- probCol - stĺpec, ktorý obsahuje pravdepodobnosť, že nájdené slovo odpovedá typu anotácie. Môže nadobúdať hodnoty od 0.0 po 1.0
- matchCase - nepovinný atribút - indikuje, či v nájdenom slove musia byť zhodné aj malé a veľké písmená. Môže mať hodnotu true a false. False je východzia hodnota.

[19] V konfiguračnom súbore config/config.xml je možné zadať prihlasovacie údaje do databázy.

4.5.2.4 Regulárne výrazy

Ďalším spôsobom, ako určiť, či daná fráza odpovedá nejakému textu je pomocou patternu. Ten v tomto prípade sa zapisuje pomocou regulárneho výrazu. Príklad definície je uvedený v kóde 4.5.

```
<simpleType type="email">
  <regex pattern="[0-9a-z]{3,}@[a-z0-9]+\.[a-z]{2,3}"  prob="0.9" />
  <regex pattern="[0-9a-z]{3,}@[a-z0-9.]+\.[a-z]{2,3}" prob="0.95" />
</simpleType>
```

Kód 4.5: Regulárny výraz

Atribútmi sú:

- pattern - regulárny výraz
- prob - pravdepodobnosť, že nájdené slovo odpovedá typu anotácie. Obsahuje čísla v rozsahu od 0.0 po 1.0

4.5.2.5 Zmiešaná definícia

Jednotlivé spôsoby definovania anotácie možno kombinovať. To znamená, že je možné v jednom XML elemente definovať anotačný typ pomocou konštanty, zoznamu z databázy aj pomocou regulárneho výrazu, ako je ukázané na príklade 4.6.

```
<simpleType type="phone">
  <const prob="0.8" value="112" />
  <const prob="0.8" value="150" />
  <const prob="0.8" value="155" />
  <const prob="0.8" value="158" />
  <const prob="0.8" value="911" />
  <db table="commercial_numers" probcol="prob" valuecol="number"  />
  <regex pattern="\+420([␣]*[0-9]){7,10}" prob="0.99" />
</simpleType>
```

Kód 4.6: Zmiešaná definícia

4.5.2.6 Kontext

Kontext, v ktorom sa vyšetrovaná fráza nachádza, upresní jej pravdepodobnostné ohodnotenie. Príkladom je Žilina. Ak by sa v okolí nachádzalo krstné meno, bude Žilina s veľkou pravdepodobnosťou priezvisko. Ak by sa v okolí nachádzalo PSČ, bude Žilina s veľkou pravdepodobnosťou mestom. Pre definíciu kontextu je potrebné definovať polohu, vzdialenosť a typ frázy (túto frázu budeme označovať ako *kontextová fráza*), ktorá tvorí kontext.

```
<simpleType type="city">
  <db table="cities" probcol="prob" valuecol="name"  />
    <context type="PSC" prob="0.9" area="ROW,U,D"
      distance="close" minprob="0.5" />
  </db>
  <const prob="0.8" value="Žilina">
    <context .../>
  </const>
  <regex pattern="\[A-Z][a-z]{2,20}" prob="0.1" />
```

```
10      <context .../>
11    </regex>
12  </simpleType>
```

Kód 4.7: Príklad definície kontextu

Kontext má tieto atribúty:

- *type* - typ kontextovej frázy, ktorá sa bude hľadať v okolí
- *prob* - kontextová pravdepodobnosť, ktorá ovplyvňuje pravdepodobnosť aktuálne vyšetrovanej frázy. Podrobne je popísaná v sekcii Pravdepodobnosť.
- *area* (nepovinný atribút) - oblasť, v ktorej sa kontextová fráza nachádza. Podrobne je popísaná v sekcii Oblasť. Východzia hodnota je *"all"*.
- *distance* (nepovinný atribút) - vzdialenosť, v akej sa kontextová fráza nachádza. Podrobne je popísaná v sekcii Vzdialenosť. Východzia hodnota je *"ignore"*
- *minprob* (nepovinný atribút) - ak je pravdepodobnosť kontextovej fráze menšia, ako minprob, bude ignorovaná.

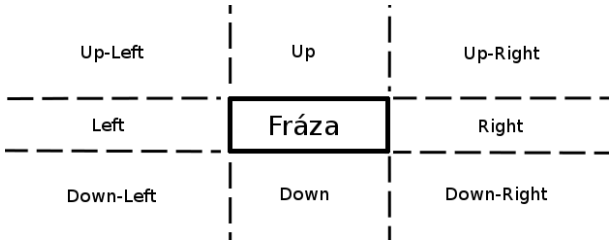
4.5.2.7 Pravdepodobnosť

Kontext v prípade pozitívneho nálezu pridáva pravdepodobnosť vyšetrovanej fráze. Hodnotu, ktorá ovplyvňuje pravdepodobnosť vyšetrovanej frázy, budeme označovať ako *Kontextovú pravdepodobnosť*. Tá môže nadobúdať hodnoty od 0.0 po 1.0. Predpokladajme, že aktuálna pravdepodobnosť vyšetrovanej frázy je p_x. Ak je hodnota kontextovej pravdepodobnosti p_k, potom výsledná pravdepodobnosť $\bar{p}_x$ bude vypočítaná podľa rovnice 4.2

$$\bar{p}_x = p_x + (1 - p_x) * p_k \tag{4.2}$$

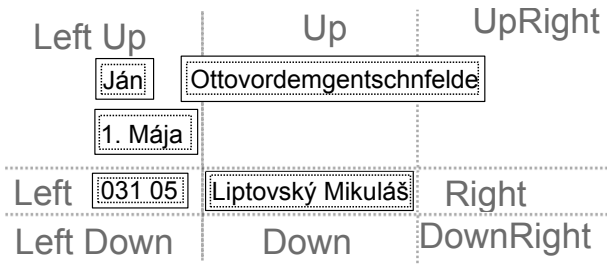
4.5.2.8 Oblasť

Každá fráza rozdeľuje svoje okolie na 8 oblastí podľa smeru: hore, dole, vľavo, vpravo a ich kombinácie, ako je zobrazené na obrázku 4.28.



Obrázok 4.28: Relatívne oblasti

Pri definovaní oblasti bude anotátor klásť dôraz tiež na to v akom smere sa nachádza kontextová fráza. Ak si zabalíme každú frázu do obdĺžnika, ako na obrázku 4.29, zistíme, že frázy môžu zasahovať do dvoch, alebo aj troch oblastí naraz. Pre vyšetrovanú frázu *Liptovský Mikuláš* by jej kontextová fráza *Ottovordemgentschnfelde* nachádzala v jej Up-Left, Up a Up-Right oblasti. Naopak pre



Obrázok 4.29: Príklad vzájomnej polohy

vyšetrovnú frázu *Ottovordemgentschnfelde* by sa jej kontextová fráza *Liptovský Mikuláš* nachádzdala len v jej Down oblasti.

Pri definícií, sa pre označovanie oblastí používajú skratky:

- *u* - Up oblasť
- *ul* - Up-Left oblasť
- *ur* - Up-Right oblasť
- *r* - Right oblasť
- *l* - Left oblasť
- *d* - Down oblasť
- *dl* - Down-Left oblasť
- *dr* - Down-Right oblasť
- *row* - Left a Right oblasť
- *col* - Up a Down oblasť
- *all* - všetky oblasti

Pri definícií, je možné oblasti kombinovať. Skratky *ROW* a *COL* len nahradzujú kombináciu *L,R* a *U,D*.

4.5.2.9 Vzdialenosť

Okrem smeru, respektíve oblasti, kde sa kontextová fráza nachádza, je často dôležitá aj vzdialenosť:

- *next* - kontextová fráza sa nachádza v bezprostrednej blízkosti vyšetrovanej fráze (tj. do vzdilenosti 1. slova).
- *near* - kontextová fráza sa nachádza maximálne do vzdialenosti 3 slov/riadkov od vyšetrovanej fráze.
- *close* - kontextová fráza sa nachádza maximálne do vzdialenosti 5 slov/riadkov od vyšetrovanej fráze.
- *ignore* - východzia hodnota. Udáva, že na vzdialenosti nezáleží.

4.5.2.10 Komplexná anotácia

Komplexná anotácia má predovšetkým zoskupovať nájdené anotované frázy. Principiálne *Komplexná anotácia* vychádza z už nájdenej anotovanej frázy. V jej okolí hľadá anotácie, ktoré do seba *komplexná anotácia* zahrnie. Rovnako ako *jednoduché anotácie*, používa *komplexná anotácia* pre zväčšenie svojej pravdepodobnosti *kontextové anotácie*. Naviac dokáže vo svojom okolí anotovať text. Napríklad ak *komplexnou anotáciou* je adresa, potom riadok nad adresou anotuje ako meno prijímateľa.

```
<complexType type="address">
  <phrase from="street" minprob="0.4">
    <contains type="city" minprob="0.4" area="d" distance="close" />
    <contains type="PSC" minprob="0.4" area="row" distance="close"
        relativeTo="city" />
    <define type="contactName" direction="U" size="phrase"/>
  </phrase>
  <phrase from="streetAddr" minProb="0.4" incProb="0.8">
    <contains type="cityzip" minProb="0.4" area="d" distance="next"
       ↪  />
    <contains type="companyForm" minProb="0.4" area="u" distance="
       ↪ near"
        relativeTo="all"/>
    <define type="contactName" direction="l" size="phrase" include="
       ↪ true"
        relativeTo="companyForm"/>
  </phrase>
  ...
</complexType>
```

Kód 4.8: Príklad definície komplexného typu

Príklad definície je zobrazený v kóde 4.8. Komplexná anotácia sa definuje elementom *complexType* s pomenovaním typu, ktorý sa udáva do atributu *type*. Pravidlá pre tvorbu komplexnej anotácie sa zadávajú do elementu *phrase*. Do neho sa zadávajú dva atibúty:

- *from* - typ východzej anotovanej frázy
- *minprob* (nepovinný atribút) - minimálne pravdepodobnosť východzej anotovanej frázy
- *incProb* (nepovinný atribút) - výsledná pravdepodobnosť sa počíta ako priemer nájdených fráz (tj. východzej frázy a fráz z *"contains"*). Táto hodnota výslednú pravdepodobnosť zväčší podľa vzorca 4.2.

Podelementom *"contains"* sa definuje, aké ďalšie anotované frázy má do seba komplexná anotácia zahrnúť. Definícia je podobná ako definícia kontextu (viď. sekcia Kontext). Chýba jej síce atribút *prob*, zato je naviac možné definovať referenčnú frázu. To jest frázu, ktorá slúži ako východzí bod pre prehľadávanie okolia.

Element "contains"má tieto atribúty:

- *type* - typ frázy, ktorá sa bude hľadať v okolí

- *minprob* (nepovinný atribút) - nájdená fráza musí mať väčšiu alebo rovnakú pravdepodobnosť, ako minprob

- *area* (nepovinný atribút) - oblasť, v ktorej sa fráza nachádza. Podrobne je popísaná v sekcii Oblasť. Východzia hodnota je *"all"*.

- *distance* (nepovinný atribút) - vzdialenosť, v akej sa fráza má nachádzať. Podrobne je popísaná v sekcii Vzdialenosť. Východzia hodnota je *"ignore"*

- *relativeTo* (nepovinný atribút) - ktorá fráza z už zahrnutých má slúžiť ako východzí bod pre prehľadávanie. Je tiež možné zadať hodnotu *all*. Z doposiaľ zahrnutých fráz sa vytvorí jedna dočasná a ta slúži ako vychodzia fráza pre prehľadávanie.

V prípade, že je komplexná fráza nájdená, je možné definovať v okolí novú frázu. Napríklad v prípade nájdenia adresy je možné definovať meno prijímateľa, ktoré môže byť ťažké definovať (napríklad ak je to spoločnosť, alebo inštitúcia).

- *type* - nájdená fráza bude anotovaná týmto typom

- *direction* - smer, v ktorej sa fráza nachádza. Hodnoty smeru sú rovnaké, ako *area*, ktorá je popísana v sekcii Oblasť. Ale *direction* môže označovať len jednu oblasť. Teda len *up*, *up-left*, *up-right* a tak podobne, ale nie *col* alebo *row*.

- *size* - akú veľkú frázu má anotátor anotovať. Môže nadobúdať hodnoty:

  - word - vyhľadá a anotuje len najbližšie *SpatialWord* v danom smere
  - phrase- su to všetky slová v najbližšej *SpatialLine*, zoradené od najbližšieho, až po slovo s oddeľovacím znakom. Oddeľovací znak sa definuje v *anotačnom súbore* a sú to štandardne znaky ";,:". Hlavným cieľom je nájsť v riadku frázu, ktorá sa nachádza medzi čiarkami, ako napríklad: "Liptovský Mikuláš, 031 05, Slovensko"
  - line - vyhľadá a anotuje len najbližšiu *SpatialLine* v danom smere

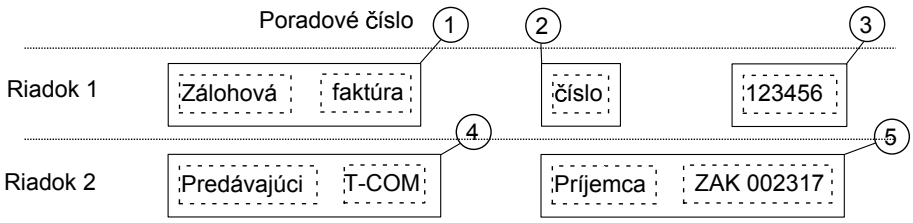
- *relativeTo* (nepovinný atribút) - od ktorej fráze sa má vyhľadávať. Tiež môže nadobúdať hodnotu *all*, čiže dočasne sa vytvorí fráza z doposiaľ nájdených fráz a od nej sa bude vyhľadávať. Východzia hodnota je *"all"*

- *include* (nepovinný atribút) - môže nadobúdať hodnoty *"true"* alebo *"false"*. Určuje, či novo definovaná fráza sa má zahrnúť do súčasnej komplexnej anotácie. Východzia hodnota je *"true"*.

4.5.3 Vyhľadávanie a anotovanie fráz

4.5.3.1 Usporiadanie a očíslovanie SpatialLine

Aby bolo možné priestorovo vyhľadávať text v dokumente, je nutné očíslovať *SpatialLine*. Jednak je potrebné priradiť im číslo riadku v dokumente a poradové číslo. Číslom riadku v dokumente je myslené, na koľkom riadku sa *SpatialLine* nachádza (viď obrázok 4.30).



Obrázok 4.30: Očíslovanie *SpatialLine*

Pri prevode textového súboru do *SpatialDocument*u je určenie čísla riadku jednoduché, nakoľko sú všetky riadky a slová rovnako vysoké a Y-ové súradnice sú v každom riadku rovnaké. Ak však bol *SpatialDocument* vytvorený z obrázku, tak nielenže v jednom riadku sú slová, ktoré majú rôznu výšku, ale dokonca hrozí že posledné slovo v riadku môže pri miernom sklone byť o "riadok vyššie"než slovo na začiatku dokumentu[20]. Pre určenie, či *SpatialWord* alebo *SpatialLine* sa nachádzajú na tom istom riadku, bolo vytvorené pravidlo:

Ak vertikálny stred jedného SpatialWord sa nachádza medzi hornou a dolnou hranicou druhého SpatialWord a takiež ak vertikálny stred druhého SpatialWord sa nachádza medzi hornou a dolnou hranicou prvého SpatialWord, potom obe SpatialWord sú na rovnakom riadku.

Toto pravidlo sa označuje ako **Pravidlo rovnakého riadku**. Ak by sme prvé slovo označili ako word1 a druhé ako word2, potom musí platiť:

$$word1.top \leq word2.verticalCenter \leq word1.bottom$$

a

$$word2.top \leq word1.verticalCenter \leq word2.bottom$$

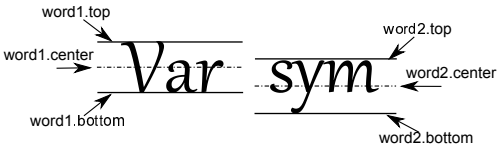
Obrázok 4.31 ilustruje pravidlo, ktoré rozhoduje, či dve slová sú na tom istom riadku. Je na ňom vidieť, že vertikálne stredy sa nachádzajú medzi hornou a dolnou hranicou druhého slova. Obrázok 4.32 zas ilustruje extrémnejší prípad. Z *pravidla rovnakého riadku* vyplýva, že len slovo *def* a písmeno *P* sú na rovankom riadku. Slová *abc* a *ghi* s písmenom *P* spĺňujú len prvú polovicu pravidla, ale nie druhú(to jest:*"stred druhého SpatialWord sa nachádza medzi hornou a dolnou hranicou prvého"*).

V očíslovaných riadkoch je teraz jednoduché prehľadávať riadok "nad", alebo "pod", alebo prehľadávať susedné *SpatialLine*.

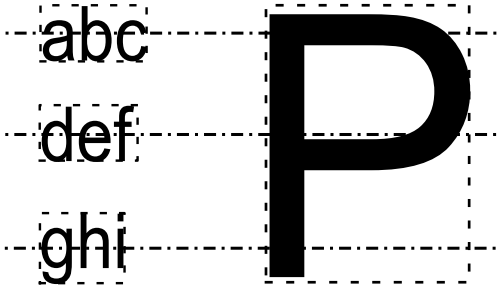
Kebyže na vyhľadávanie použijeme R-strom, bolo by oveľa ťažšie a časovo náročnejšie takéto slová nájsť. Napríklad ak by sme chceli nájsť slová z predchádzajúceho riadku, museli by sme určiť oblasť z ktorej chceme vyhľadať slová podľa veľkosti písma, medzery medzi riadkami a predpokladať, že písmo je rovnako veľké. Problémom sú prípady, keď najbližší riadok je takzvane ob-jedno[21]. Iným spôsobom využitia R-stromov by bolo vybrať slová z väčšej oblasti, a zoradiť ich podľa vzdialenosti. Tu je problém s blízkymi slovami a určením, ktoré slová sú skutočne z predchádzajúceho riadku.

[20]ak na papieri formátu A4, je vytlačený text s veľkosťou písma 10pt, stačí, aby bol dokument pootočený o 1 stupeň a slovo na konci a začiatku riadku sa po Y-ose vôbec neprekrývajú a algoritmus vyvodzuje, že sú na rozdielnych riadkoch

[21]často sa uvádza meno prijímateľa, prázdny riadok, adresa prijímateľa



Obrázok 4.31: Slová rovnakého riadku



Obrázok 4.32: Slová rôznych riadkov

Po očíslovaní *SpatialLine* sa všetky zoradia podľa poradového čísla. V každom *SpatialLine* naviac treba zabezpečiť, že *SpatialWord* budú zoradené podľa x-ovej súradnice. Teraz je možne každému *SpatialWord*u priradiť nasledujúce a predchádzajúce *SpatialWord*. Tieto ukazovatele sa používajú v algoritmoch, ktoré potrebujú prehľadávať len susedné slová.

4.5.3.2 Predfiltrovanie slov

SpatialDocument ktorý bol vytvorený pomocou OCR modulu, často obsahuje reťazce, ktoré nie sú slovami. OCR aplikácia sa snaží rozoznať slovo z čiar kriviek, ktoré sú obrázkami, logom, pečiatkou, alebo je špina na papieri, pokrčenie oskenovaného dokumentu[22] alebo dokonca kancelárska spinka ponechaná na papieri. Výsledkom je napríklad reťazec "—_._;!___—". Slová, ktoré neobsahujú abecedný, alebo číselný znak sa do výsledného dokumentu nepridávajú.

4.5.3.3 Inicializácia anotácií

Trieda *AnnotateBuilder* z *Anotačného súboru config/annotate.xml* načíta všetky anotačné pravidlá do inštancie triedy *AnnotationDefinition*. Tá funguje ako datakeeper[23] anotačných pravidiel. *AnnotateBuilder* následne vytvára inštanciu triedy *Annotator*, čo je hlavná trieda, ktorá anotuje text. *AnnotateBuilder* postupne z *AnnotationDefinition* vyberá definované pravidlá, generuje anotácie[24] a pridáva ich do *Annotator*u.

4.5.3.4 Vyhľadávanie konštánt

Konštanty sa v *Anotačnom súbore* definujú buď ako:

- *const* - vytvorí sa jedna konštanta
- *db* - z databázy sa z definovaného stĺpca vyberú riadky a z nich sa vytvoria konštanty

Pri rozhodovaní, akú štruktúru použiť pre čo najrýchlejšie vyhľadávanie konštánt v texte, respektíve v *SpatialDocument*e, vyšla ako najneefektívnejšia štruktúra

[22]pokrčenie spôsobuje tieň, ktorý je tmavší než papier a OCR si teda myslí, že to môže byť znak

[23]Datakeeper je trieda, ktorej hlavným cieľom je udržiavať dáta a zjednodušovať prístup k nim

[24]anotácie sú inštancie tried, pomocou ktorých *Annotator* anotuje text

hešovacia tabuľka, respektíve hešovacia mapa[25]. Vyplývalo to z výsledkov uvedených v sekcii 7.3.1 Vyhľadávanie slov v texte.

Annotator preto pre vyhľadávanie konštánt v *SpatialWord*e používa hešovaciu mapu. Hešovacia mapa používa ako kľúč objekty typu String a ako hodnoty používa pole anotácií. Napríklad kľúč *Žilina*"bude odkazovať na pole, ktoré obsahuje anotácie *[(mesto,40%),(priezvisko,20%)]*. Teda ak hešovacia mapa obsahuje slovo z *SpatialWord*u, priradí mu zoznam odpovedajúcich anotácií.

Annotator prehľadáva *SpatialDocument* po slovách: z každého *SpatialLine* sa vyberú slová a tie sa vyhľadávajú v hešovacej mape. Nájdenie odpovedajúcich anotácií je jedno-slovné konštanty je teda jasné.

Pre vyhľadávanie konštanty, ktorá pozostáva z dvoch a viacerých slov, napríklad "Liptovský Mikuláš", je potrebné rozšíriť spôsob vyhľadávania. Konštanta sa rozdelí na slová. Prvé slovo sa použije ako vyhľadávací kľúč do hešovacej mapy a ostatné slová sa uložia do anotácie. Pri vyhľadávaní sa potom dodatočne overuje každá anotácia, či je konštanta anotácie jedno, alebo viacslovná. Ak je viacslovná, potom sa musia prehľadať aj susedné slová vyšetrovaného *SpatialWord*. Vďaka tomu, že každé *SpatialWord* má ukazateľ na následujúce slovo (viď sekcia Usporiadanie a očíslovanie SpatialLine) stačí vziať odpovedajúci počet susedných slov a tie porovnať so slovami, ktoré sú uložené v anotácii (jej konštante).

V definícií anotácie konštanty je možné definovať, či pri porovnávaní záleží na veľkosti znakov. *Annotator* v skutočnosti obsahuje tri hešovacie mapy, kde každá odpovedá jednému zo spôsobov porovnávaní:

- *Match case* - záleží na veľkosti znakov
- *Ignore case* - nezáleží na veľkosti znakov
- *First letter* - len pri prvom znaku záleží na veľkosti

Porovnávanie typu *Match case* je popísané v predchádzajúcom odstavci.

Pri porovnávaní typu *Ignore case* je konštanta v anotácilí prevedená na malé písmená a následne uložená do hešovacej mapy. Táto hešovacia mapa je označená ako *constantsIgnoreCase*[26]. Pri vyhľadávaní je slovo zo *SpatialWord*u tiež prevedené na malé písmená a následne hľadané v hešovacej mape *constantsIgnoreCase*.

Pri porovnávaní typu *First letter* je to podobné, ale len u prvého znaku je zachovaná veľkosť a hešovacia mapa je označená ako *constantsFirstLetterMatch*.

4.5.3.5 Vyhľadávanie regulárnych výrazov

Pri vyhľadávaní podľa regulárnych výrazov sa všetky *SpatialWord*y prevedú do jedného reťazca, pričom je zachované ich poradie. Pri prevode sa tiež zaznačujú poradové čísla *SpatialWord* slov v reťazci, aby sa dali spätne dohľadať. Obrázok 4.5.3.5 ilustruje reťazec s uloženými poradovými číslami *SpatialWord*[27].

Annotator potom postupne prechádza všetky anotácie s regulárnymi výrazmi a v reťazci hľadá ich výskyty. V prípade nálezu dohľadá podľa pozície *SpatialWord*y a priradí im anotáciu.

[25]HashMap je implementácia hešovacej tabuľky v jazyku Java

[26]V triede *Annotator* má táto hešovacia mapa názov *constantsIgnoreCase*

[27]v skutočnosti sa ukladajú do intervalového poľa kôli úspore pamäte

		L	i	p	t	o	v	s	k	ý		M	i	k	u	l	á	š	
		1	1	1	1	1	1	1	1	1		2	2	2	2	2	2	2	

Obrázok 4.33: Indexácia *PhraseWords* v reťazci

4.5.3.6 Nájdenie kontextu

Annotator v prvom kroku hľadá v dokumente jednoduché anotácie. Anotované frázy ukladá priamo do *spatialdocument*u.Tie potom prechádza a hľadá, či majú definovaný aj kontext. V prípade, že áno, začne v okolí vyšetrovanej frázy hľadať frázu odpovedajúcu kontextu. Fráza, ku ktorej *Annotator* hľadá kontext sa označuje ako **vyšetrovaná fráza**. Príkladom môže byť vyšetrovaná fáza *Liptovský Mikuláš* a v *anotačnom súbore* je definované pravidlo uvedené na obrázku 4.9. Podľa tohto príkladu by *Annotator* naľavo, napravo a pod frázou *Liptovský Mikuláš* hľadal frázu anotovanú ako *zip* (teda PSČ).

```
...
<simpleType type="city">
  <db table="cities" probcol="prob" valuecol="name"  />
    <context type="zip" prob="0.9" area="L,R,D"
       distance="next" minprob="0.5" />
  </db>
</simpleType>
...
```

Kód 4.9: Príklad kontextu

Prehľadávanie susedných fráz prebieha v dvoch krokoch. Najprv sa prehľadávajú frázy smerom k začiatku dokumentu a v druhom kroku smerom ku koncu dokumentu. Maximálna vzdialenosť, do ktorej sa prehľadáva je daná hodnotou *distance* v definícií pravidla. Hodnota *distance* je popísaná v sekcii Vzdialenosť. Tá môže určovať, že maximálna vzdialenosť kontextovej fráze je 1,3,5 alebo na vzdialenosti nezáleží (tá je reprezentovaná hodnotou 10 000, pretože sa predpokladá, že dokument nebude mať viac ako 10 000 riadkov). Odpočítaním/pripočítaním tejto hodnoty k číslu riadku *vyšetrovanej fráze* je definovaná hodnota **minRowNumber** a **maxRowNumber**.

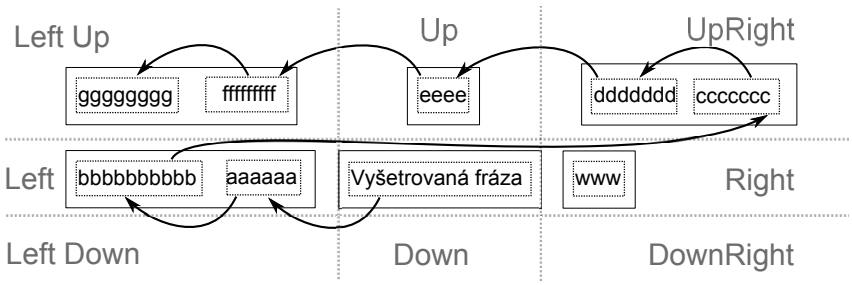
Pseudo kód algoritmu prehľadávania je založený na syntaxi jazyka Java. Premenná oblasť je bitová maska, ktorá z definície pravidla kontextu určuje oblasti, v ktorých sa má kontextová fráza vyhľadávať, preto operátor {& je braný ako bitový operátor a operátor *&&* je braný ako logický AND. Je tiež dôležité si uvedomiť, že vyšetrovaná fráza môže byť na niekoľkých riadkoch. Pseudokód algoritmu je uvedený v kóde 4.10.

```
line = actual_phrase.previous
while(line.line_number >= minRowNumber && line.line_number>=0) {
   rozdel slova riadku line do: laveSlova, stlpcoveSlova,
      ↪ praveSlova
   if (riadok >= horny riadok vysetrovanej fraze)
      //prehladavanie na urovni vysterovanej fraze
      if (oblast & left > 0)  najdi frazu v laveSlova
      if (oblast & right > 0) najdi frazu v praveSlova
   else
```

```
9         //prehladavanie nad urovnou vysetrovanej fraze
10        if (oblast & leftUp > 0 ) najdi frazu v laveSlova
11        if (oblast & up > 0 )     najdi frazu v stlpcoveSlova
12        if (oblast & rightUP > 0) najdi frazu v praveSlova
13  }
```

Kód 4.10: Algoritmus vyhľadávania kontextových fráz

Na obrázku 4.34 je ilustorvaná poradie, v akom sa kontextové fráze vyhľadávajú oblasti, do ktorých slová spadajú.



Obrázok 4.34: Poradie prehľadávania kontextových fráz

Prehľadávanie smerom ku koncu dokumentu sa vykonáva obdobne.

V prípade nájdenia kontextovej frázy sa overí, či spĺňa aj podmienku minimálnej pravdepodobnosti (tá je definovaná parametrom *minprob*). Podľa vzorca pre výpočet pravdepodobnosti, uvedenom v sekcii Pravdepodobnosť, sa pre aktuálne nájdenú kontextovú frázu vypočíta nová pravdepodobnosť *vyšetrovanej frázy*.

Ak má jedna fráza v anotačnom pravidle definovaných viac kontextových pravidiel, potom vyhľadávanie a výpočet výslednej pravdepodobnosti *vyšetrovanej frázy* sa robí nezávisle od ostatných kontextových pravidiel. Teda ak prvá kontextová fráza zvýšila výslednú pravdepodobnosť napríklad z 0.4 na 0.7, potom druhá kontextová fráza zvýši pravdepodobnosť z 0.7 napríklad na 0,85 (nie z 0.5 na 0.75). Z toho jasne vyplýva, čim viac kontextových pravidiel má vyšetrovaná fráza, tým vyššiu pravdepodobnosť môže mať.

4.5.3.7 Vyhľadávanie komplexných anotácií

Komplexné anotácie na rozdiel od jednoduchých nemajú definované vzory (patterny), ktoré by vyhľadávali v texte dokumentu. Komplexné anotácie sú závislé na nájdených jednoduchých anotáciách. Ich zjednodušený princíp je následovný:

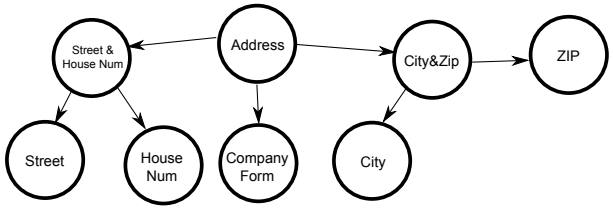
1. nájdi frázu konkrétneho typu
2. v jej okolí nájdi ďalšie anotované frázy (podobne ako kontext) a zahrň ich do tejto anotácie
3. (ak je definované) nájdi kontext v okolí a zvýš pravdepodobnosť tejto frázy[28]
4. (ak je definované) v určenom smere vyber slovo/riadok a ten definuj ako novú frázu

[28] rovnako ako kontext funguje o jednoduchej anotácie

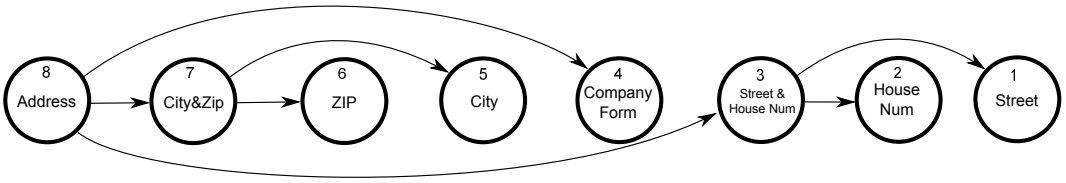
Príprava

Z definície *komplexných anotácií* (viď sekcia Komplexná anotácia) plynie závislosť (z definície *"from"* a definície *"contains"*), kde jedna komplexná anotácia môže byť závislá buď na *jednoduchej anotácií*, alebo na inej *komplexnej anotácií*. V momente hľadania komplexnej anotácie je teda potrebné mať už nájdené anotácie, na ktorých aktuálne vyšetrovaná anotácia zavisí. Preto je potrebné určiť poradie, v akom sa budú komplexné anotácie v dokumente vyhľadávať. Problémom by bolo, ak by komplexné anotácie boli závislé cyklicky. V tom prípade by sa nedalo určiť poradie, v akom by sa mali komplexné anotácie vyhľadávať. Preto cyklické závislosti budú zakázané.

Pre vyriešenie usporiadania, je potrebné previesť závislosti komplexných anotácií do grafu. Obrázok 4.35 ilustruje príklad, ako by taký graf vyzeral. Graf pomocou topologického usporiadania[8] zoradíme tak, aby všetky závislosti (šípky) smerovali vpravo. Tie najkomplexnejšie anotácie sa budú tým pádom nachádzať vľavo a tie nezávislé vpravo. Pri usporiadaní sa zároveň zistí, či graf obsahuje, alebo neobsahuje cyklus. V prípade, ak by sa cyklus našiel, aplikácia zobrazí upozornenie o chybnej konfigurácií *anotačného súboru* a skončí. Usporiadaný graf je zobrazený na obrázku 4.36.



Obrázok 4.35: Príklad neusporianých závislostí prevedených do grafu



Obrázok 4.36: Príklad usporiadaných závislostí prevedených do grafu

Hľadanie

Výsledný zoznam je prechádzaný odzadu (teda od najjednoduchšieho typu), pričom sa vyhľadávajú len komplexné anotácie (jednoduché anotácie boli už nájdené v predchádzajúcich krokoch, viď kapitoly Vyhľadávanie konštánt a Vyhľadávanie regulárny V algoritme je použitá funkcia *"nájdi anotáciu typu X v okolí"*. Pre nájdenie anotácie sa používa algoritmus pre nájdenie kontextu v okolí, popísaný v kóde 4.10 (viď sekcia Nájdenie kontextu).

Pseudokód algoritmu je uvedený v kóde 4.11. Ten hľadá frázu v dokumente pre konkrétnu komplexnú anotáciu. Vhľadávanie sa spúšťa zavolaním funkcie *findPhraseForAnnoation()* (pseudokód vychádza z jazyku Java) :

```
1
2  function findPhraseForAnnoation(komplexAnotacia){
3    F = komplexAnotacia.from //typ vzchodzej anotovanej fraze
4    najdeneKomplexneFraze= new Array()
5    for each( basePhrase : najdi v dokumente fraze typu F){
6        najdeneFraze=getRelativePhrases(komplexAnotacia,
7                        [basePhrase],
8                        komplexAnotacia.contains)
9        najdeneKomplexneFraze.append(najdeneFraye)
10   }
11   for each( foundPhrase: najdeneKomplexneFraze){
12      najdi a vytvor frazu komplexAnotacia.define v okoli foundPhrase
13   }
14 }
15
16 function getRelativePhrases(komplexAnotacia,
17                             foundPhrases,
18                             phrasesToSearch){
19   //ak netreba vyhadavat dalsiu anotovanu frazu,
20   if (phrasesToSearch.isEmpty==true){
21     // vytvor novu komplexnu anotovanu frazu a vrat ju v poli
22     return [new ComplexPhrase(foundPhrases,komplexAnotacia)]
23   }
24
25   if (phrasesToSearch[0].relativeTo == 'all'){
26     //vytvorenie docasenej fraze
27     relativnaFraza = new ComplexPhrase(foundPhrases,komplexAnotacia)
28   } else {
29     relativnaFraza = najdi v foundPhrases frazu
30                      typu phrasesToSearch[0].relativeTo
31   }
32
33   X = phrasesToSearch[0].type
34   trebaNajstFraze= odober 0-ty prvok pola phrasesToSearch
35
36   najdeneKomplexneFraze = new array
37   for each(najdenaFraza: najdi anotacie typu X v okoli
38                                  fraze relativnaFraza){
39     doposialNajdeneFraze= foundPhrases.clone()
40     doposialNajdeneFraze.append(najdenaFraza)
41     // rekurzivne
42     najdeneFraze= getRelativePhrases(komplexAnotacia,
43                                      doposialNajdeneFraze,
44                                      trebaNajstFraze);
45
46     najdeneVyslendeKomplexneFraze.append(najdeneFraze)
47   }
48   return najdeneKomplexneFraze
49 }
```

Kód 4.11: Algoritmus pre vyhľadávanie komplexných fráz

Na začiatku sa v dokumente vyhľadajú všetky anotované frázy typu, ktorý je definovaný v komplexnej anotácií v atribúte *"from"*. Pre nájdenú frázu sa

následne zavolá funkcia *"getRelativePhrases()"*, ktorá vráti pole nájdených komplexne anotovaných fráz.

Ako je v kóde 4.11 vidieť, vyhľadávanie *"contains"* fráz (teda ďalšich fráz, ktoré tvoria komplexne anotovanú frázu) je použitá rekurzívna funkcia. Pretože prehľadávanie fráz definovaných v *"contains"* je závislé od doposiaľ nájdených fráz a oblasť, v ktorej sa bude fáza vyhľadávať (definovaná parametrom *"relativeTo"*) závisí práve na doposiaľ nájdených fráz, vedie spôsob hľadania k prehľadávaniu typu *"do hĺbky"*. To z dôvodu, že v každom bode je nutné si udržiavať informáciu, "odkiaľ sme prišli", čo v tomto prípade je pole fráz, ktoré sa doposiaľ našli.

Rekurzívna funkcia *"getRelativePhrases()"* najprv skontroluje pole *"phrasesToSearch"*, teda aké ďalšie anotácie treba hľadať:

- Ak je pole prázdne, je jasné, už žiadnu frázu netreba hľadať a teda stačí vytvoriť komplexne anotovanú frázu a tú vrátiť. Pretože ale celkovým výsledkom funkcie je pole anotovaných fráz, je nutné túto frázu zabaliť do pola.
- Ak ale pole prázdne nie je:
  - vyberie z poľa *"phrasesToSearch"* prvú definíciu (ktorá sa definuje v *anotačnom súbore* ako element *"contains"*) a nájde podľa tejto definície všetky anotované frázy v dokumente (anotované frázy teda musia odpovedať oblasti, v ktorej sa majú nachádzať, musia mať odpovedajúci typ a tiež pravdepodobnosť)
  - potom pre každú nájdenú frázu : vytvorí nove pole z doposiaľ nájdených fráz a prídá k nim nájdenú frázu, z poľa *"phrasesToSearch"* odoberie prvý prvok a rekurzívne sa zavolá funkcia *"phrasesToSearch"*.
  - všetky výsledky z rekurzívneho volania sa uložia do poľa, ktoré funkcia vráti ako výsledok

Pretože s každým volaním je odobraný jeden prvok z poľa *"phrasesToSearch"*, je rekurzívne volanie deterministické.

Po vyhľadaní

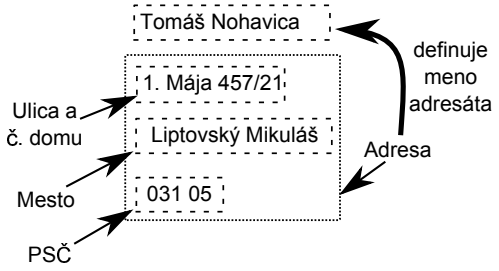
Potom, ako je komplexná fráza nájdená, *Annotator* skontroluje či definícia komplexnej anotácie obsahuje element *"define"*. Ten totiž umožňuje anotovať frázu, ktorá sa nachádza v okolí komplexne anotovanej frázy.

Na obrázku 4.37 je príklad, ako komplexne anotovaná fráza môže anotovať text vo svojom okolí. Nájdená komplexne anotovanán fráza je *adresa*. Tá pozostáva z ulice, čísla domu, mesta a PSČ. Ak je definícia "define"rovnaká, ako v príklade uvedenom v kóde 4.12, potom *Annotator* nad komplexne anotovanou frázou (pretože parameter *"direction"* je nastavená na *"u"*, čo je *"UP"* a *relativeTo="all"*, čo označuje celú komplexnú frázu) vyberie celý riadok (kvôli parametru *size="line"*). Z tohoto riadku vytvorí novú frázu typu *"contactName"* a začlení ho do seba (kvôli parametru *include="true"*).

Grafické znázornenie príkladu je na obrázku 4.37. Je tam vidieť, že komplexná fráza typu "adresa"definuje z riadku nad ňou novú frázu.

```
<complexType type="address">
  <phrase from="streetAddr" minProb="0.4" incProb="0.8">
     ...
   <define type="contactName" direction="u" size="line" include="
      ↪ true"
        relativeTo="all"/>
  </phrase>
  ...
</complexType>
```

Kód 4.12: Príklad definície "define"komplexného typu



Obrázok 4.37: Príklad ako komplexná anotácia identifikuje novú frázu

4.5.3.8 Detekcia tabuliek

Anotátor detekuje tabuľky z bielych miest, ktoré oddeľujú *SpatialLine* v dokumente. Podmienkou pre detekciu tabuľky je, aby tabuľka mala aspoň tri stĺpce. Väčšina polo-štrukturovaných dokumentov má text rozdelený na "názov položky"a "dáta položky", ktoré sú zoradené pod sebou. Preto v prípade, ak by bolo povolené, že tabuľka má mať aspoň dva stĺpce, boli by tieto dáta chybne identifikované ako tabuľka.

Pri detekcii tabuľky anotátor postupne prechádza riadok po riadku a zisťuje počet *SpatialLine* v riadku. Ak je tento počet aspoň 3, zapamätá si ho spolu s bielymi miestami, ktoré oddeľujú *SpatialLine*. Ak v ďalšom riadu opäť nájde aspoň 3 *SpatialLine*, pričom biele miesta majú neprázdny prienik s bielymi miestami predchádzajúceho riadku, vytvorí tabuľku s dvoma riadkami a zapamätá si prienik bielych miest.

Pri vyšetrovaní ďalších riadkov musí platiť podmienka, že prienikom bielych miest sa ich počet nezníži. Ak by sa počet znížil znamenalo by to, že nejaká *SpatialLine* toto biele miesto pretína a "ruší"tým oddeľujúce biele miesto. V tom riadku teda tabuľka končí.

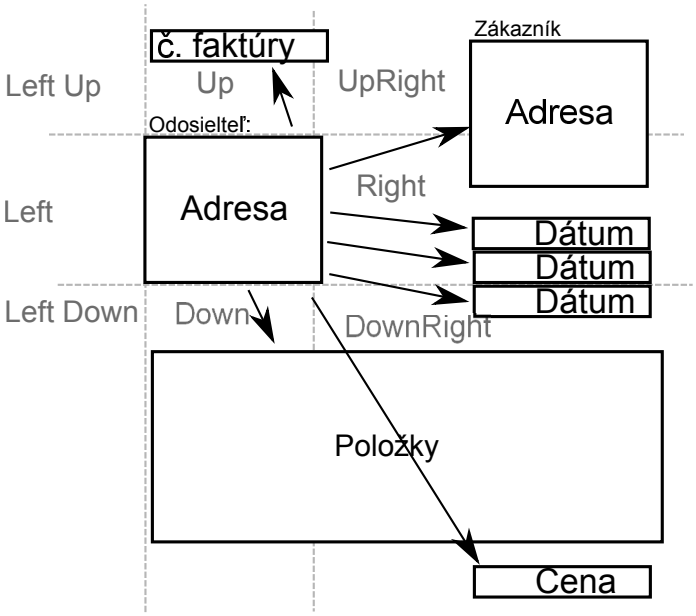
Z naakumulovaných *SpatialLine* vytvorí anotovanú frázu, ktorú pridá do *SpatialDocumentu*.

4.6 Vyhľadávač dokumentov

Celé vyhľadávanie podobných dokumentov sa vykonáva v databáze. Preto je nutné celý dokument spolu so všetkými frázami uložiť do databázy. Pre vyhľadávanie podobných dokumentov sa z nájdených fráz použijú len tie, ktoré majú pravdepodobnosť väčšiu ako 90%. Táto hodnota vychádza z nastavenia *konfiguračného*

súboru, kde fráza s hodnou väčšou ako 0.9 skutočne odpovedá svojmu typu. V prípade, že fráza je súčasťou komplexnej frázy, vyberie sa len rodičovská komplexná fráza. Takéto frázy sa označujú ako *ImportantPhrase*. V bežnom dokumente ich býva do 10-tich. Vzájomná poloha a typ *ImportantPhrase* fráz bude tvoriť vyhľadávací kľúč pre daný dokument.

ImportantPhrase budú rovnaké.



Obrázok 4.38: Vzájomné polohy *ImporetantPhrases*

Hlavou myšlienkou pri vyhľadávaní podobných dokumentov je nájsť dokument, ktorý má *ImportantPhrase* rovnako rozložené, teda že vzájomné polohy *ImportantPhrase* sú rovnaké. Vzájomnou polohou sa rozumie oblasť, do ktorej spadá jedna *ImportantPhrase* voči druhej. Na obrázku 4.38 je zobrazené, do akých oblastí spadajú *ImportantPhrase* voči fráze *adrese odosielateľa*. Ako je vidieť, tak:

- *číslo faktúry* spadá do oblasti "Up"a "UpRight",
- *adresa zákazníka* spadá do oblasti "UpRight"a "Right",
- *dátumy* spadajú do oblasti "Right"jeden do oblasti "RightDown",
- *položky* spadá do oblasti "Down"a "DownRight",
- *cena* spadá do oblasti "DownRight",

Každá oblasť má priradené svoje číslo, ktoré je mocninou čísla dva (tj 1,2,4,8...). To umožňuje ukladať vzájomnú polohu ako bitovú mapu. Vďaka tomu sa dá priamo v databáze pomocou bitových operátorov porovnať či dve podobné frázy z dvoch rôznych dokumentov majú na rovnakých miestach rovnaké typy fráz.

ImportantPhrase je treba ale rozšíriť ešte o jeden atribút, a to je poradové číslo pre daný typ frázy. Ako je zobrazené na obrázku 4.38, dátumy sú v tesnej blízkosti a ich vzájomné polohy voči ostatným sú takmer rovnaké. Ak by sa teda našiel podobný dokument s rovnakým rozložením, bolo by problematické namapovať frázy jedného dokumentu na frázy druhého. Jednalo by sa tu totiž o namapovanie

jedného grafu na druhý (kde uzlami sú frázy a hranami ich vzájomná poloha). Preto pre uľahčenie je treba priradiť indexy frázam.

Priraďovanie indexov sa robí následovne:

1. *ImportantPhrase* sa rozdelia do jednotlivých skupín podľa typu fráz
2. v každej skupine sa *ImportantPhrase* zoradia podľa polohy v dokumente (tj. tie, čo sú najbližšie pravému hornému rohu sú ako prvé)
3. každej *ImportantPhrase* sa priradí index podľa jej poradia v skupine. Tento index sa ukladá ako parameter *"occurence"*

Tento spôsob indexovania zachováva indexy v prípade, keby jedna z *ImportantPhrase* by nebola v dokumente nájdená. Kebyže jedna z *ImportantPhrase* by nebola nájdená, rozhodí to celé indexovanie a podobný dokument by tak nemusel byť nájdený.

V databázi nemusia byť uložené všetky vzájomné polohy. Napríklad ak fráza B je v oblasti UP frázy A, potom A bude v oblasti DOWN frázy B. Preto sa postupne prechádzajú frázy zhora nadol a ukladajú sa len dopredné frázy tak, ako je to zobrazené v kóde 4.13.

```
1 soretedPhreases = sort Important Phrases
2 for (i=0;i<soretedPhreases.size -1; i++){
3   for (j=i+1;i<soretedPhreases.size ;j++){
4      ulozVzajomnuPolohu(soretedPhreases[i],soretedPhreases[j])
5   }
6 }
```

Kód 4.13: Ukladanie vzájomných polôh

Prvá fráza funkcie *ulozVzajomnuPolohu()* sa označuje ako **ukladaná phrase**, a druhá sa označuje ako **referovaná fráza**.

V databázi sa vzájomná poloha ukladá v tabuľke *ssda_important_phrases* do týchto stĺpcov:

- phraseid - id ukladanej frázy
- documentid - id dokumentu
- phrasetype - typ *ukladanej frázy*
- occurence - index *occurence ukladanej frázy*
- area - oblasť, v ktorej sa nachádza *referovaná fráza* voči *ukladanej frázy.*
- referencedphrase - id *referovanej frázy*
- referencedtype - typ *referovanej frázy*
- referencedoccurence - index *occurence referovanej frázy*

Pre nájdenie rovakého dokumentu potom stačí v dokumente vyhľadať dokument, ktorý obsahuje čo najviac zhodných *ImportantPhrases*. SQL dotaz pre nájdenie podobného dokumentu je zapísaný v kóde 4.14, kde číslo "1234567"je ID aktuálneho dokumentu:

```
1  SELECT ref.documentid, sum(1) as matchingrate
2  FROM ssda_important_phrases base ,ssda_important_phrases ref
3  WHERE
4    ref.phrasetype=base.phrasetype
5    and ref.referencedtype=base.referencedtype
6    and ref.occurence=base.occurence
7    and ref.referencedoccurence=base.referencedoccurence
8    and (base.area&ref.area > 0)
9    and base.documentid= 1234567
10 GROUP BY ref.docid
11 ORDER BY matchingrate
```

Kód 4.14: Vyhľadávanie podobných dokumentov

Ako je uvedené v kóde 4.14, porovnávajú sa frázy podľa typu, poradia frázy v dokumente (*occurence*), typu referenčnej frázy a jej poradia (jej *occurance*) a musí byť v rovnakej oblasti.

Dokument, ktorý je vyšetrovaný, čiže ten ku ktorému sa hľadá podobný dokument, sa označuje ako *base document*. V SQL dotaze je záznam z tabuľky *ssda_important_phrases* s aliasom *base*.

Výsledkom SQL dotazu je zoznam, ktorý obsahuje ID dokumentu a počet zhodných prvkov s *base document*. Ako prvý dokument by mal byť **base document**, pretože najpodobnejší si je on sám. Jeho hodnota *matchingrate* udáva maximálne skóre, ktoré je možné získať. Ak by však prvým dokumentom nebol *base document*, je jasné že nájdený dokument má maximálnu zhodu s *base document*. [29] V opačnom prípade sa vezeme druhý v poradí a jeho *matchingrate*. Tento dokument sa označuje ako **matching document**. Ak *matchingrate matching document*u je aspoň 80% z *matchingrate base document*u, potom sa tieto dokumenty automaticky označia za podobné. V opačnom prípade vyhľadávač dokumentov najprv napáruje nájdené *ImportantPhrases*, zobrazí ich užívateľovi s upozornením, na koľko sú dokumenty podobné a opýta sa či napárované *ImportantPhrases* sú v poriadku. Ako prvý dokument by mal byť **base document**, pretože najpodobnejší si je on sám. Jeho hodnota *matchingrate* udáva maximálne skóre, ktoré je možné získať. Ak by však prvým dokumentom nebol *base document*, je jasné že nájdený dokument má maximálnu zhodu s *base document*. [30] V opačnom prípade sa vezeme druhý v poradí a jeho *matchingrate*. Tento dokument sa označuje ako **matching document**. Ak *matchingrate matching document*u je aspoň 80% z *matchingrate base document*u, potom sa tieto dokumenty automaticky označia za podobné. V opačnom prípade vyhľadávač dokumentov najprv napáruje nájdené *ImportantPhrases*, zobrazí ich užívateľovi s upozornením, na koľko sú dokumenty podobné a opýta sa či napárované *ImportantPhrases* sú v poriadku.

Je potrebné si uvedomiť, že do tabuľky sa ukladajú vzájomné polohy fráz "každý s každým", čiže ich kartézsky súčin. 80% zhoda (číselne 0,8) tohto kartézskeho súčinu teda ukazuje, že v $\sqrt{0,8} = 0.891$, teda v 89.1% frázach sú si dokumenty podobné.

[29] Pretože PostgreSQL zaručuje len zoradenie podľa hodnoty *"matchingrate"*, môže sa stať, že prvý záznam nebude *base document*.

[30] Pretože PostgreSQL zaručuje len zoradenie podľa hodnoty *"matchingrate"*, môže sa stať, že prvý záznam nebude *base document*.

V opačnom prípade, keď *matchingrate* nájdeného dokumentu menej ako 80%, vyhľadávač dokumentov najprv napáruje nájdené *ImportantPhrases*, zobrazí ich užívateľovi s upozornením, na koľko sú dokumenty podobné a opýta sa či napárované *ImportantPhrases* sú v poriadku.

5. Užívateľská dokumentácia

5.1 Inštalácia

Aplikácia je multiplatformná, avšak návod a balíčky sú určené pre Windows pracujúci na 64 bitovom procesore. Pre beh aplikácie je nutné mať nainštalovanú platformu Java verzie 8 a vyššie.

Z adresára *install* na priloženom CD je potrebné nainštalovať:

- platformu Java, ktorú je možné inštalovať zo súboru *jdk-8u25-windows-x64.exe*, alebo zo stránky *http://java.com/en/download*
- databázový systém PostgreSQL je možné nainštalovať zo súboru *postgresql-9.4.2-1-windows-x64.exe*, alebo zo stránky *http://www.postgresql.org/download/*
- Tesseract OCR, ktorý je možné nainštalovať zo súboru *tesseract-ocr-setup-3.02.02.exe*, alebo podľa návodu na stránke *https://github.com/tesseract-ocr/tesseract/wiki*. Je dôležité, aby bol spustiteľný z príkazového riadku. Preto cestu k *tesseract.exe* je potrebné pridať do systémovej premennej *Path*.

Pre jednoduchšiu manipuláciu s databázou je vhodné mať nainštalovaného klienta *PgAdminIII*. Ten je možné nainštalovať buď zo súboru *pgadmin3-1.20.0.zip*, alebo zo stránky *http://www.pgadmin.org/download/*.

Adresár *ssda* na priloženom CD treba prekopírovať na harddisk, alebo médium, na ktoré je možné zapisovať. Aplikácia v niektorých prípadoch zapisuje dáta do adresára *out*.

Po inštalácií PostgreSQL je treba vytvoriť databázu zo zálohy a nastaviť prístupové údaje pre aplikáciu. Záloha databáze je uložená v súbore *install/ssda.backup*. Najjednoduchší spôsob obnovy databáze je pomocou klienta *PgAdminIII*:

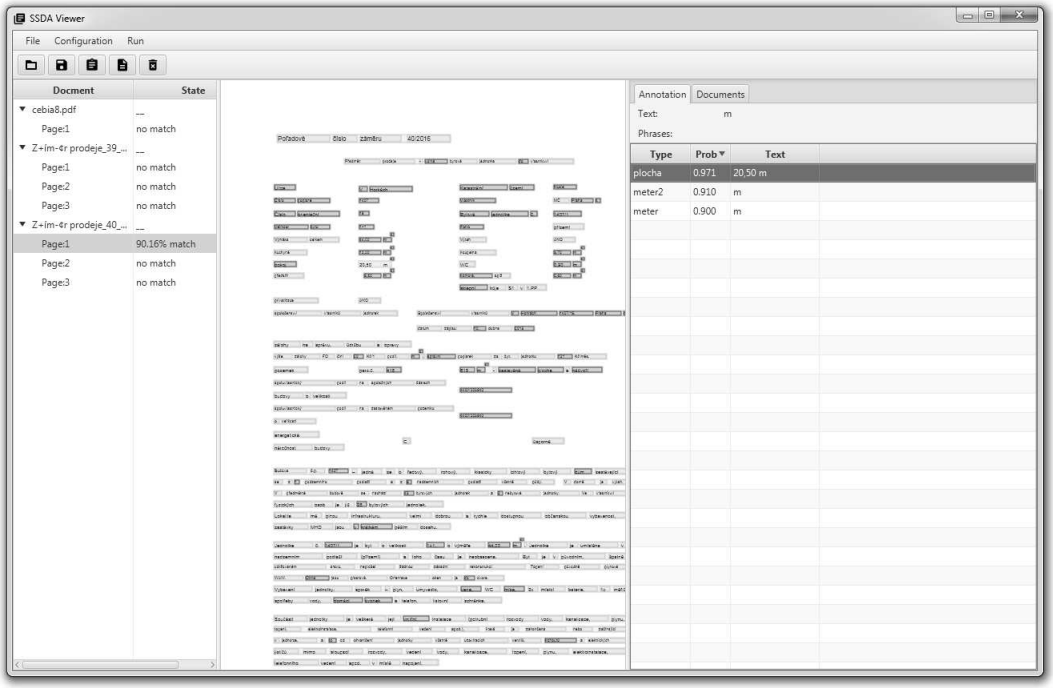
- Pravé tlačítko myši na "Databases", vybrať položku "New database..."a zadať meno databáze (napríklad "ssda")
- Pravým tlačítkom myši kliknúť na novo vytvorenú dazabázu, vybrať "Restore...", následne vybrať do políčka "File"cestu k *install/ssda.backup* a klknúť na talčítko "Restore"

Prístupové údaje k databázi pre aplikáciu sa nastavujú v súbore *ssda/config/-config.xml*.

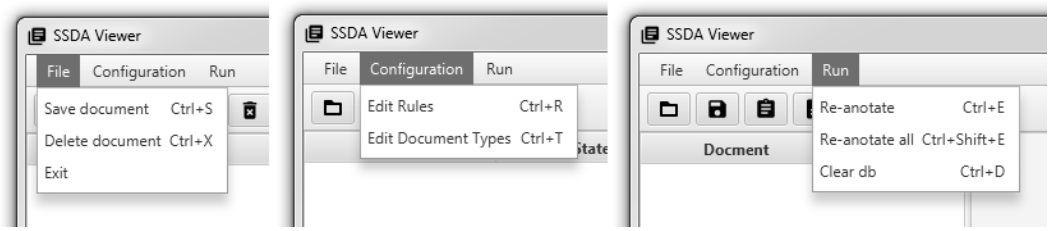
5.2 Aplikácia

Aplikácia sa spúšťa príkazom "ssda/run.bat". Po spustení aplikácie sa zobrazí hlavné okno, zobrazené na obrázku 5.1, ktoré je vizuálne rozdelené na 3 časti:

- Zoznam dokumentov (vľavo),
- Grafický náhľad vybraného dokumentu (v strede)



Obrázok 5.1: Hlavné okno aplikácie po rozoznaní dokumentu



Obrázok 5.2: Hlavné Menu

- Detail anotovaných fraz a detail rozoznaného dokumentu (vpravo).

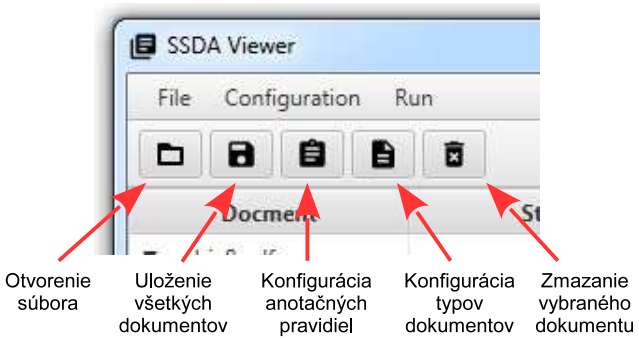
Menu hlavného okna (viď obrázok 5.2) obsahuje :

- *File*
  - *Save document* - uloženie všetkých zmenených dokumentov
  - *Delete document* - zmazanie aktuálne vybraného dokumentu. Okrem skratky *Ctrl+X* je možné použiť aj tlačiko *Delete*.
  - *Exit* - ukončenie aplikácie
- *Configuration*
  - *Edit Rules* - editovanie anotačných pravidiel
  - *Edit Document Types* - editovanie typov dokumentov a ich položiek
- *Run*
  - *Re-anotate* - znovu anotovanie vybraného dokumentu (napríklad po zmene anotačných pravidiel)

– *Re-anotate all* - znovu anotovanie všetkých dokumentov

– *Clear db* - vymazanie všetkých dokumentov a šablón z databázy

Panel nástrojov obsahuje tlačítka pre otvorenie, uloženie a mazanie dokumentov, otvorenie konfiguračného okna pre anotačné pravidlá a typy dokumentov (viď obrázok 5.3).



Obrázok 5.3: Panel nástrojov

5.2.1 Rozoznanie dokumentu

Otvorenie dokumentu/dokumentov pre rozoznanie je možné okrem tlačítka *Open document* aj pomocou *Drag & Drop*. Vkladať je možné dokonca celé adresáre. Po otvorení sa dokument najprv rozparsuje po jednotlivých stránkach na *SpatialDocument*-y. Meno súboru je v najvyššej úrovni stromu, ako ilustruje obrázok 5.4. Spracovanie dokumentu má niekoľko stavov:

- *waiting* - SpatialDocument čaká na anotáciu
- *annotated* - SpatialDocument je anotovaný, čaká na nájdenie šablóny
- *no match* - žiadna dostatočne podobná šablóna sa nenašla
- *X% match* - našla sa podobná šablóna, zhodná na X percent.

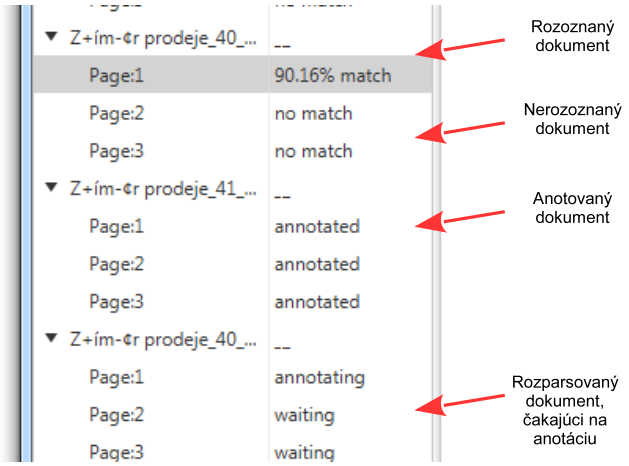
5.2.2 Grafický náhľad dokumentu

Stredný panel obsahuje komponentu pre grafický náhľad vybraného dokumentu. Pomocou myši je možné náhľad posúvať alebo zväčšovať/zmenšovať pomocou kolieska myši. Kliknutím na nejaké slovo sa toto slovo zvýrazní a v pravom paneli zobrazí detail vybraného slova, ako ukazuje obrázok 5.5.

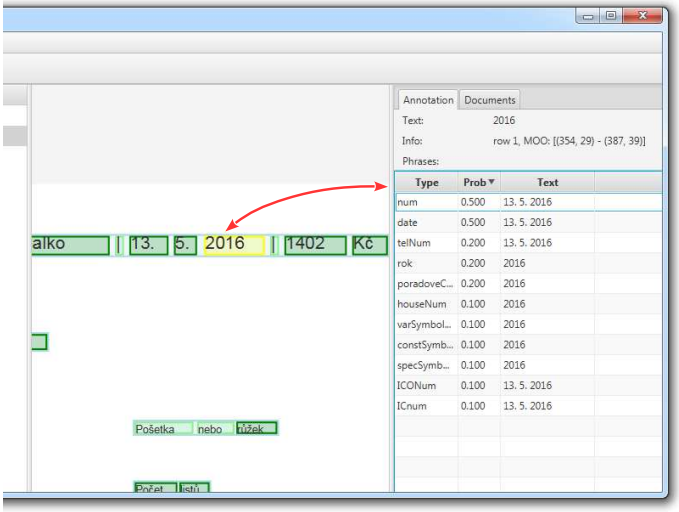
5.2.3 Pravý panel

Panel obsahuje dve záložky. *Annotation* a *Documents*. Záložka *Annotation* zobrazuje bližšie informácie o vybranej fráze. Jej komponenty zobrazujú:

- Text slova, ktorý je možné skopírovať



Obrázok 5.4: Stavy rozparsovania dokumentu



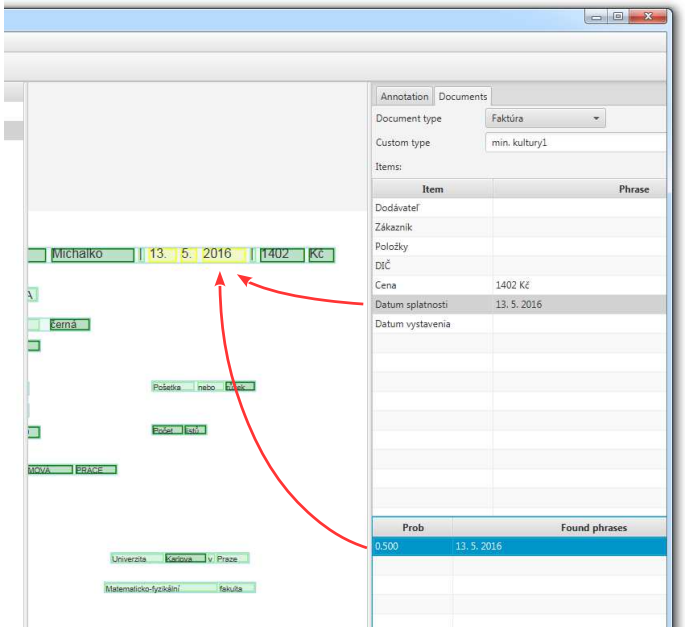
Obrázok 5.5: Detail označeného slova

- Číslo riadku, na ktorom sa slovo nachádza a hodnoty jeho MOO
- Tabuľku anotovaných fráz, do ktorých vybrané slovo patrí. Tá obsahuje:
  - Typ anotácie
  - Text anotovanej frázy
  - Pravdepodobnosť

  Kliknutím na anotovanú frázu v tabuľke sa v grafickom náhľade zvýraznia všetky slová patriace do anotovanej frázy.

Záložka *Documents*, zobrazená na obrázku 5.6, obsahuje komponenty pre zobrazenie/editovanie:

- Typu dokumentu.
- Vlastného označenia typu dokumentu (napríklad poskytovateľa služieb).
- Tabuľku s dokumentovými položkami a nájdenými/priradenými frázami.



Obrázok 5.6: Detail dokumentu

- Tabuľku, ktorá zobrazuje len tie frázy, ktoré je možné nastaviť do vybranej dokumentovej položky. Sú to teda frázy, ktoré majú rovnaký typ ako dokumentová položka.

Ako ilustruje obrázok 5.6, kliknutím na riadok tabuľky, ktorý obsahuje frázu, sa v strednom paneli táto fráza zvýrazní.

5.2.4 Konfigurácia anotačných pravidiel

Dialógové okno pre editáciu anotačných pravidiel (viď obrázok 5.7) je možné zobraziť buď pomocou tlačítka *Configuration rules* v paneli nástrojov (5.3) alebo menu položkou *Configuration / Edit Rules*.

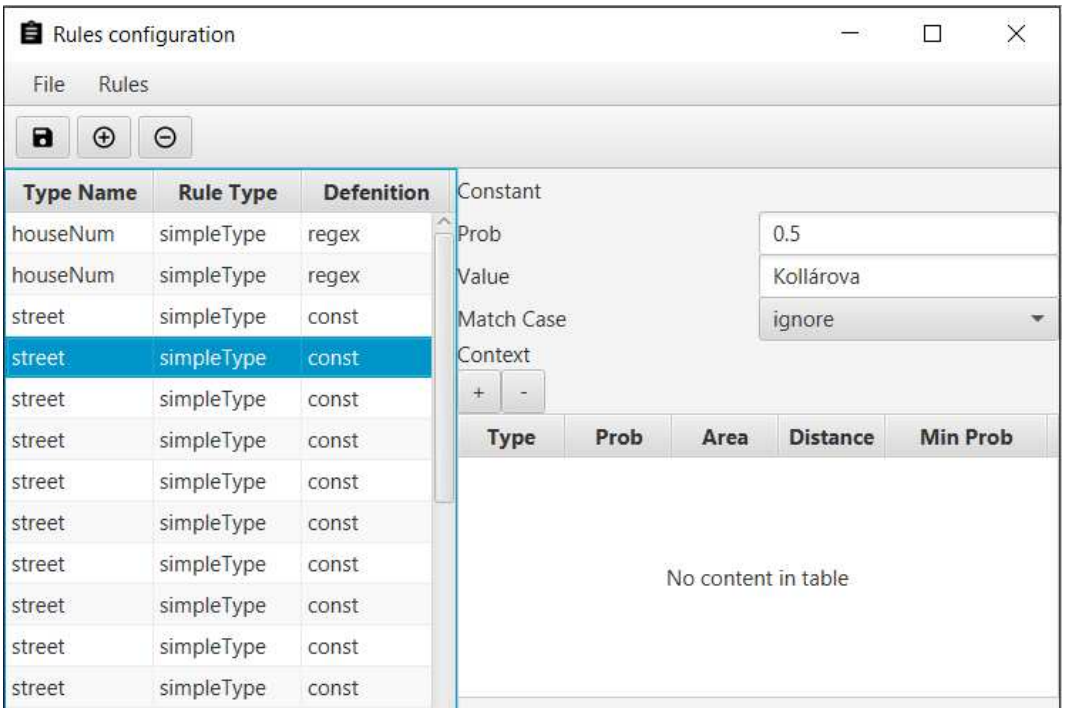
Dialóg edituje anotačné pravidlá, ktoré sú uložené v súbore *config/annotate.xml*.

5.2.4.1 Pridávanie pravidiel

Tlačítkom *Add Rule* v paneli nástrojov, alebo menu položkou *Rules / New rule* je možné pridať nové anotačné pravidlo (obrázok 5.8).

Pri pridávaní nového anotačného pravidla sa zobrazí pomocný dialóg v ktorom je potrebné zadať typ pravidla a jeho definície. Kombobox *Defenition* dáva na výber jednu zo čtyroch možností:

- const (simple) - konštanta,
- db (simple) - zoznam z databáze,
- regex (simple) - regulárne výrazy,
- phrase (complex) - komplexná anotácia zoskupujúca nájdené anotované frázy.



Obrázok 5.7: Dialóg pre editáciu anotačných pravidiel

Tieto typy sú popísané v sekcii Definícia anotácie. Vytvorenie je potrebné potvrdiť tlačítkom *Add* (obrázok 5.9).

Nové pravidlo se umiestnené do zoznamu pravidiel v tabuľke, ktorá sa nachádza v ľavej časti okna. V pravej časti je možné definovať parametre nového pravidla (obrázok 5.10).

Vzhľad pravej časti dialógového okna sa mení s typom anotačného pravidla.

Pre pravidlá typu **konštanta (const)** obsahuje pravý panel nasledujúce položky (viď obrázok 5.11):

1. *Prob* - povinné pole pre definovanie pravdepodobnosti, že nájdené slovo odpovedá typu anotácie. Povolený rozsah čísel je od 0.0 po 1.0.

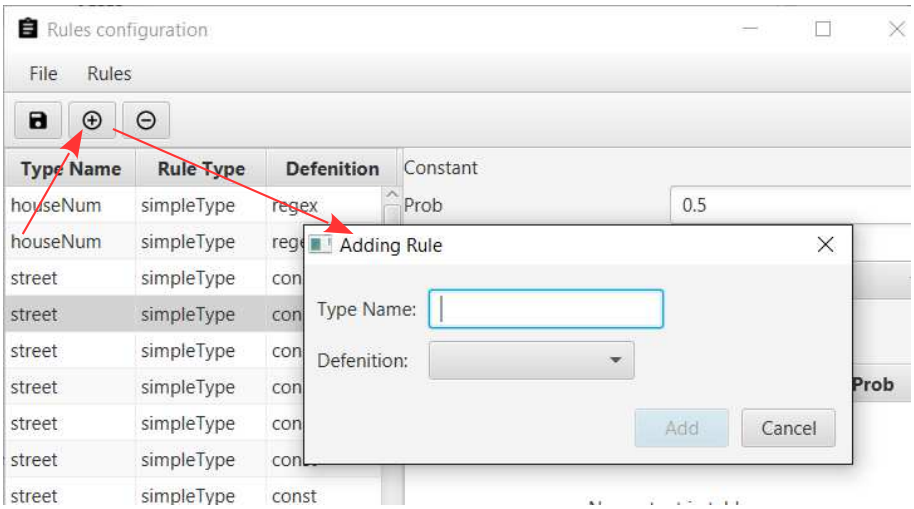
2. *Value* - povinné pole pre uvedenie hodnoty konštanty.

3. *Match Case* - nepovinný atribút. Môže obsahovať jednu z hodnôt *true, false, firstLetter* alebo *ignore* v prípade, ak užívateľ nechce definovať atribút.

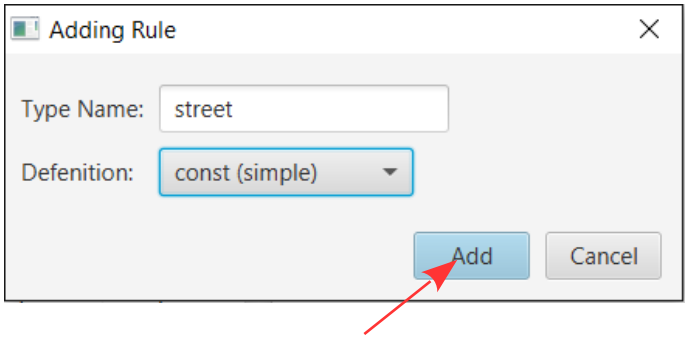
4. *Context* - tabuľka pre pridávanie kontextu, v ktorom sa vyšetrovaná fráza nachádza. Zvyšuje pravdepodobnostné ohodnotenie frázy. Pridávanie kontextu nie je povinné. Pre pridávanie a odoberanie kontextu slúžia tlačítka "+" a "-" umiestené nad tabuľkou.

   Tabuľka obsahuje stĺpce:

   - *Type* - typ kontextovej frázy, ktorá sa bude hľadať v okolí.
   - *Prob* - kontextová pravdepodobnosť, ktorá vylepšuje pravdepodobnosť.



Obrázok 5.8: Pridanie anotačných pravidiel



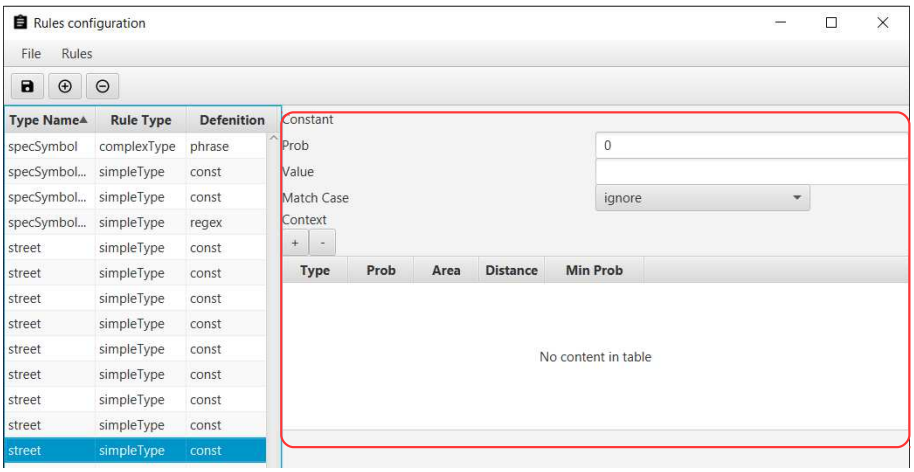
Obrázok 5.9: Pridanie nového pravidla

- *Area* - oblasť, v ktorej sa kontextová fráza nachádza. Môže mať hodnoty: *u, ul, ur, r, l, d, dl, dr, row, col, all* (podrobne sú popísané v sekcii Oblasť). Je možné zadať viac hodnôt, ktoré ale musia byť oddelené čiarkou. Východzia hodnota je *all*.
- *Distance* - vzdialenosť, v akej sa kontextová fráza nachádza. Môže mať hodnoty: *next, near, close, ignore* (podrobne sú popísané v sekcii Vzdialenosť). Východzia hodnota je *ignore* a udáva, že na vzdialenosti nezáleží.
- *Min Prob* (nepovinný atribút) - ak je pravdepodobnosť kontextovej frázy menšia, ako *min prob*, bude táto kontextová fráza ignorovaná.

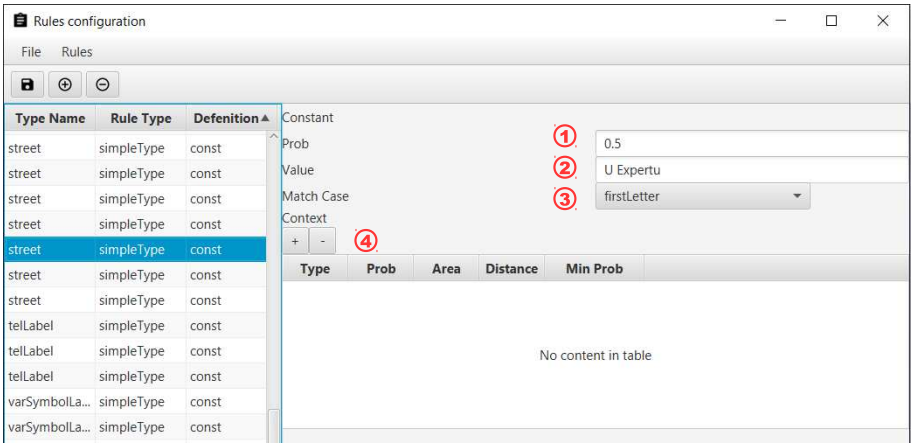
Viac informácií o kontexte je v sekcii Kontext.

Pre pravidlá typu **zoznam z databázy (db)** obsahuje pravý panel nasledujúce položky (viď obrázok 5.12):

1. *Value Column* - povinné pole, stĺpec tabuľky, ktorý obsahuje textové hodnoty.
2. *Probability Column* - povinné pole, stĺpec tabuľky, ktorý obsahuje pravdepodobnosť, že nájdené slovo odpovedá typu anotácie.
3. *Table* - povinné pole, definuje tabuľku, z ktorej sa majú vyberať hodnoty.



Obrázok 5.10: Detail nového pravidla



Obrázok 5.11: Const detail

4. *Where* - nepovinné pole, umožňuje filtrovať hodnoty v tabuľke.

5. *Match Case* - Môže mať hodnotu *true, false, firstLetter*, alebo *ignore* v prípade ak užívateľ nechce špecifikovať atribút. Indikuje, či v nájdenom slove musia byť zhodné aj malé a veľké písmená, alebo či záleží na veľkosti len pri prvom znaku.

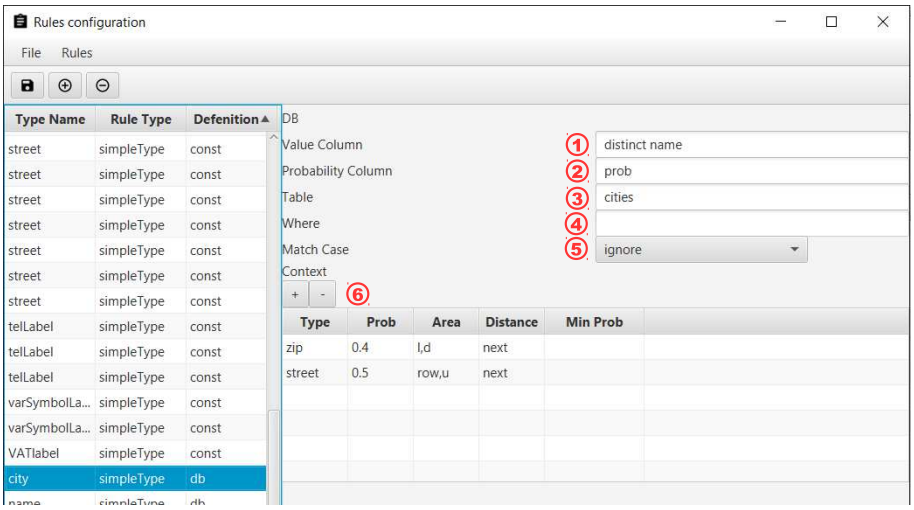
6. *Context* - tabuľka pre pridanie kontextu (rovnaká tabuľka, ktorá je popísaná v paneli pre editáciu konštanty).

Pre pravidlá typu **regulárne výrazy (regex)** obsahuje pravý panel nasledujúce položky (viď obrázok 5.13):

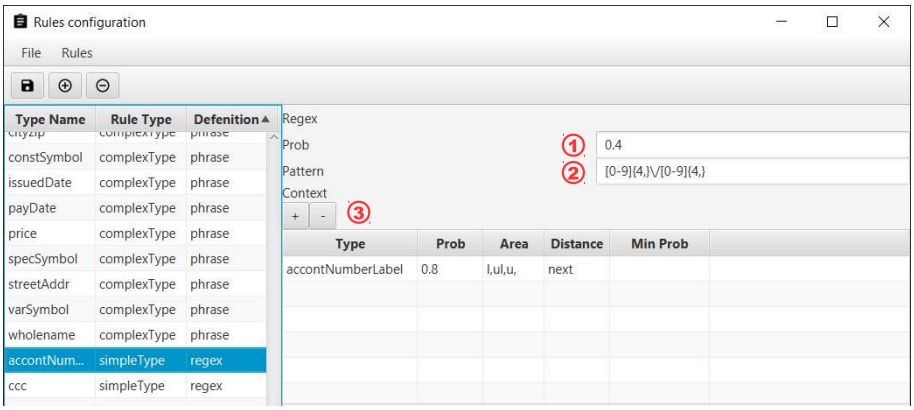
1. *Prob* - povinné pole pre uvedenie pravdepodobnosti, že nájdené slovo odpovedá typu anotácie. Povolený rozsah čísel od 0.0 po 1.0.

2. *Pattern* - povinné pole pre definíciu regulárneho výrazu.

3. *Context* - tabuľka pre pridanie kontextu (rovnaká tabuľka, ktorá je popísaná v paneli pre editáciu konštanty).



Obrázok 5.12: DB detail



Obrázok 5.13: Regex detail

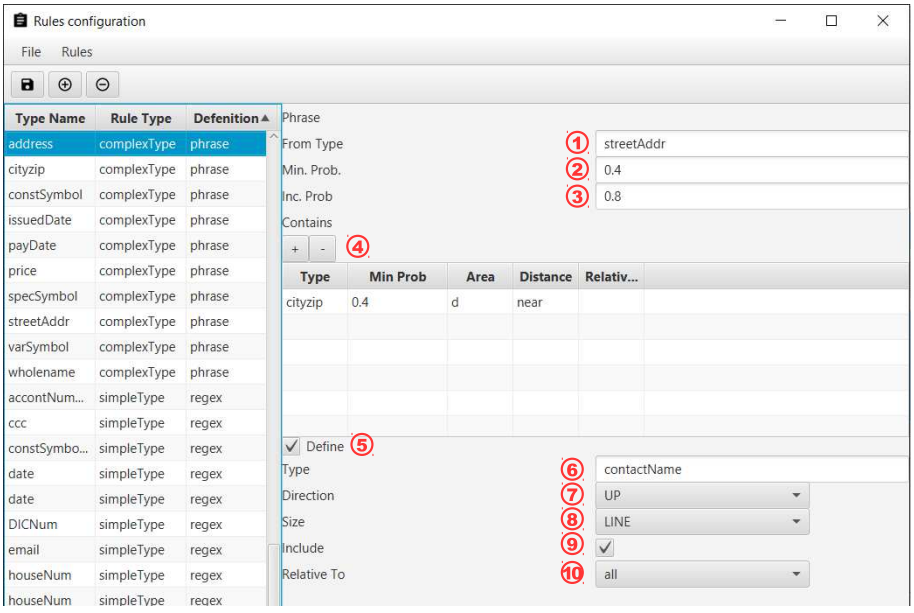
Pre pravidlá typu **komplexná anotácia (phrase)** obsahuje pravý panel nasledujúce položky (viď obrázok 5.14):

1. *From Type* - povinné pole pre definíciu typu východzej anotovanej frázy.
2. *Min. Prob.* - povinné pole pre definíciu minimálne pravdepodobnosti východzej anotovanej frázy.
3. *Inc. Prob.* - povinné pole pre definíciu pravdepodobnosti, ktorá vylepšuje pravdepodobnosť
4. *Contains* - tabuľka pre definíciu ďalších anotovaných fráz, ktoré majú byť do komplexnej anotácie zahrnuté. Pre pridávanie a mazanie anotovaných fráz slúžia tlačítka "+" a "-" umiestené nad tabuľkou. Tabuľka *Contains* má tieto stĺpce:
   - *Type* - typ frázy, ktorá sa bude hľadať v okolí.
   - *Min Prob* - nájdená fráza musí mať väčšiu alebo rovnakú pravdepodobnosť, ako minprob.

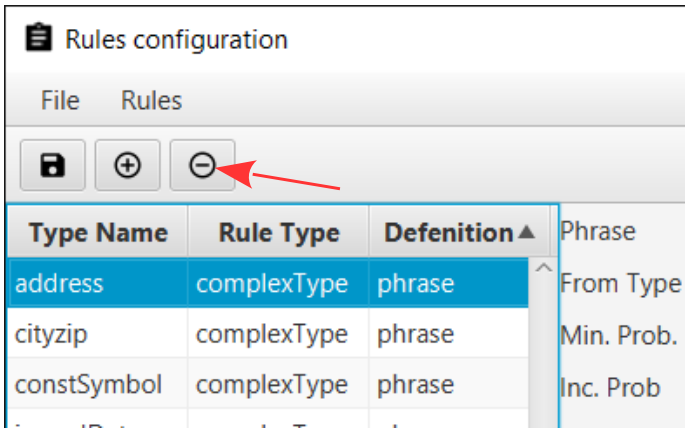
- *Area* - oblasť, v ktorej sa fráza nachádza. Môže mať hodnoty: *u, ul, ur, r, l, d, dl, dr, row, col, all* (podrobne sú popísané v sekcii Oblasť) ktoré možno kombinovať. Východzia hodnota je *all*.
- *Distance* - vzdialenosť, v akej sa fráza má nachádzať. Môže mať hodnoty: *next, near, close, ignore* (podrobne sú popísané v sekcii Vzdialenosť). Východzia hodnota je *ignore* a udáva, že na vzdialenosti nezáleží.
- *Relative To* (nepovinný atribút) - ktorá fráza z už zahrnutých má slúžiť ako východzí bod pre prehľadávanie. Je tiež možné zadať hodnotu *all* - teda doposiaľ vytvorenú frázu, ako celok.

5. *Define* - checkbox sa zaškrtne v prípade, ak je komplexná fráza nájdená a je možné definovať novú frázu v jeho okolí. Ak je toto pole zaškrtnuté objaví sa komponenty 6-10, ktoré sú povinné:
6. *Type* - typ, ktorý je priradený definovanej fráze.
7. *Direction* - smer, v ktorom sa fráza nachádza.
8. *Size* - veľkosť frázy. Môže nadobúdať hodnoty: *word, phrase, line* (teda slovo, slová po prvý oddeľovač, alebo celý riadok).
9. *Include* - určuje, či novo definovaná fráza sa má zahrnúť do súčasnej komplexnej anotácie.
10. *Relative To* - od ktorej frázy sa v smere *Direction* novo definovaná fráza nachádza.

Viac informácií o komplexnej anotácií sa nachádza v sekcii Komplexná anotácia.



Obrázok 5.14: Phrase detail



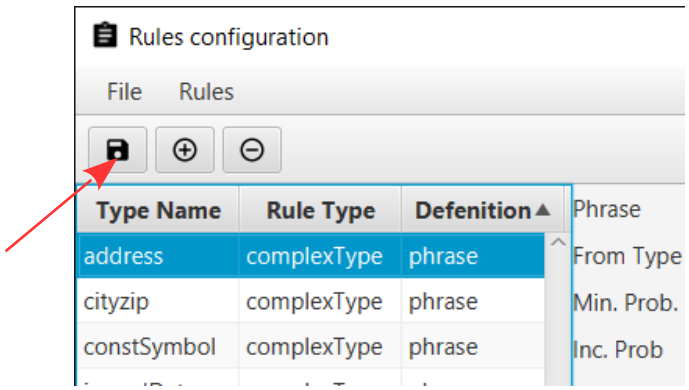
Obrázok 5.15: Mazanie anotačných pravidel

5.2.4.2 Mazanie anotačných pravidiel

Vybrané pravidlo je možné zmazať buď pomocou menu položky *Rules / Delete Rule*, alebo klávesovou skratkou *Ctr + D*.

5.2.4.3 Uloženie pravidiel do konfiguračného súboru

Zmeny v konfigurácií je potrebné uložiť, inak nebudú mať žiadny efekt na anotovanie dokumentov. Uloženie zmien je možné pomocou menu položky *File / Save*, alebo klávesovej skratky *Ctrl + S*. Zmeny sa tak uložia do anotačného súboru.

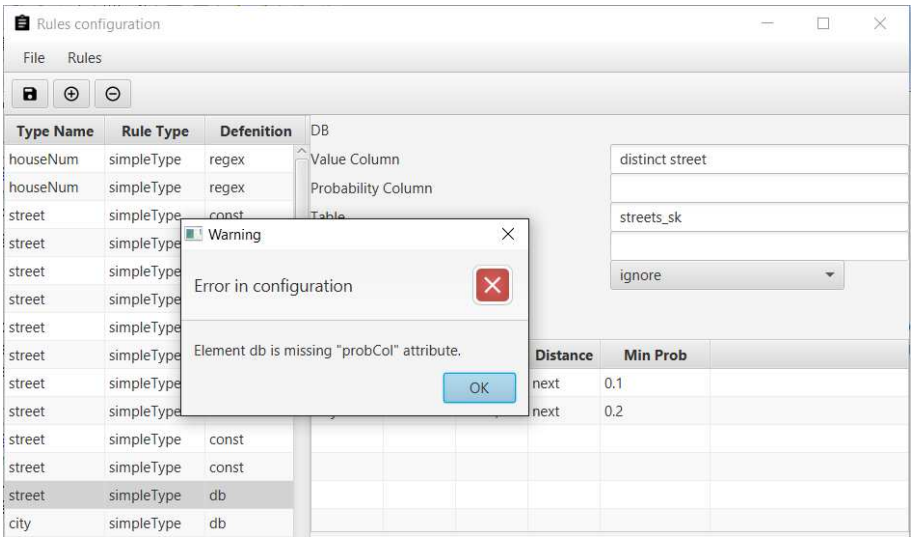


Obrázok 5.16: Uloženie pravidiel do konfiguračného súboru

Pri ukladaní anotácií sa overuje, či všetky atribúty boli zadané korektne. V prípade nájdenej chyby sa zobrazí dialógové okno popisujúce nájdený problém (viď obrázok 5.17). V tomto prípade sa zmeny neukladajú.

V prípade, že v konfigurácií žiadne chyby neboli, vyšle sa signál do hlavného okna, pre znovu načítanie konfigurácie. To je vykonávané na pozadí a môže chvíľku trvať. Preto treba počkať cca 5 sekúnd pred znovu-anotovaním dokumentov.

Dialógové okno umožňuje otestovať správnosť konfigurácie bez nutnosti ukladania, a to buď menu položkou *File / Validate*, alebo klávesovou skratkou *Ctrl + R*

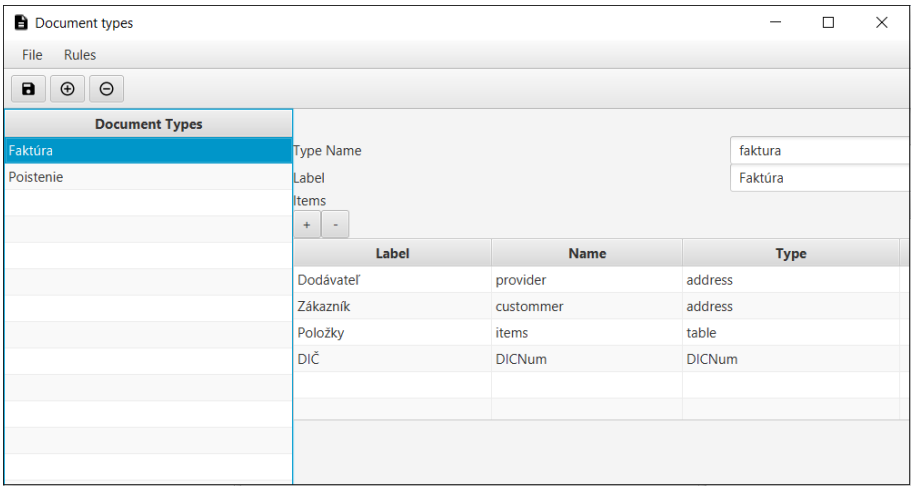


Obrázok 5.17: Chyba konfigurácie pri ukladaní

5.2.5 Konfigurácia typov dokumentu

Dialógové okno pre editáciu typov dokumentov (viď obrázok 5.18) je možné zobraziť buď pomocou tlačítka *Document types* v paneli nástrojov (5.3) alebo menu položkou *Configuration / Edit Document Types* v hlavnom okne.

Dialóg zobrazuje obsah súboru *config/documenttypes.xml* a obsahuje jednoduché rozhranie pre jeho editáciu.

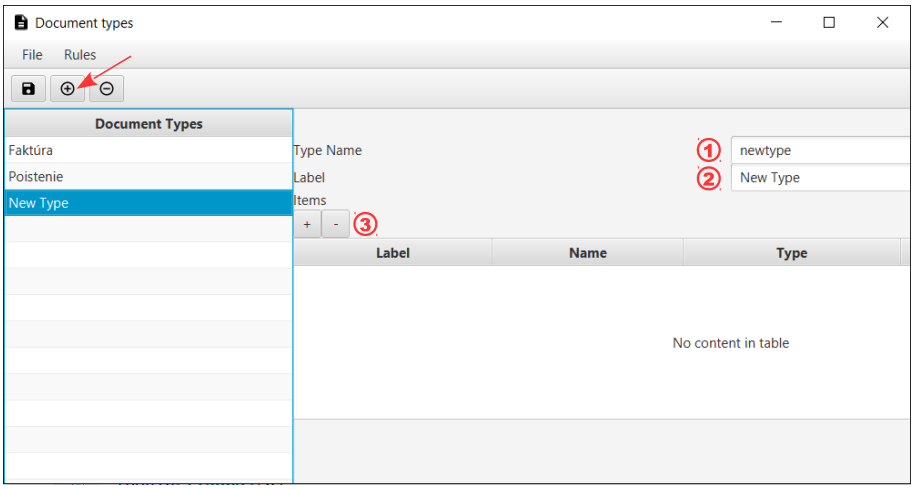


Obrázok 5.18: Dialóg pre editáciu typu dokumentu

5.2.5.1 Pridanie nového typu dokumentu

Tlačítkom *Add Document Type* v paneli nástrojov, alebo menu položkou *Document Types/New document type* je možné pridať nový typ dokumentu (obrázok 5.19). Nový záznam je pridaný na koniec zoznamu v pravej časti dialógu. Ľavá časť dialógu obsahuje editáciu jeho položiek:

1. *Type Name* - povinné pole pre definíciu technického mena typu dokumentu.



Obrázok 5.19: Pridanie typu dokumentu

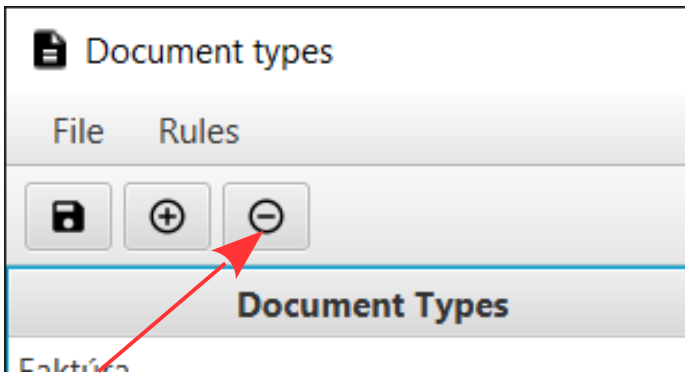
2. *Label* - povinné pole pre definíciu mena dokumentu, ktrá sa zobrazuje užívateľovi.

3. *Items* - tabuľka položiek patriace dokumentu. Pre pridanie a mazanie položiek slúžia tlačítka "+" a "-" umiestnené nad tabuľkou. Tabuľka obsahuje následujúce parametre:

   - *Label* - meno položky v dokumente (ktorá sa zobrazuje v tabuľke DoucmentItems v hlavnom okne),
   - *Name* - technické meno (určené pre program),
   - *Type* - typ frázy (napríklad adresa).

5.2.5.2 Mazanie anotačných pravidiel

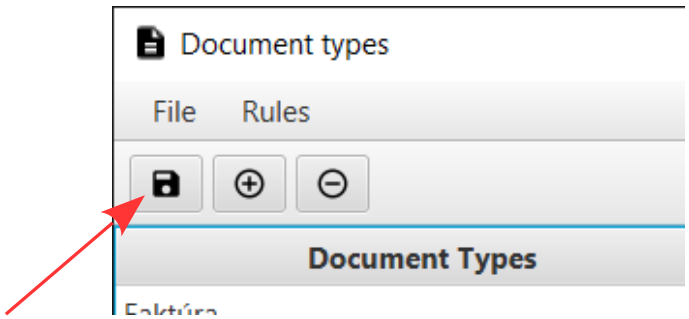
Vybraný typ dokumentu je možné zmazať buď pomocou menu položky *Document Types / Delete document type*, klávesovou skratkou *Ctrl + D*, alebo tlačítkom **Delete Document Type** na paneli nástrojov (viď obrázok 5.20).



Obrázok 5.20: Mazanie typu dokumentu

5.2.5.3 Uloženie typu dokumentu do konfiguračného súboru

Uloženie zmien je možné pomocou menu položky *File / Save*, klávesovej skratky *Ctrl + S*, alebo tlačítka **Save** v paneli nástrojov (obrázok 5.21). Zmeny sa tak uložia do konfiguračného súboru.



Obrázok 5.21: Uloženie typu dokumentu do konfiguračného súboru

6. Programátorská dokumentácia

Programátorská dokumentácia je určená pre tých, ktorý by chceli SSDA aplikáciu upraviť, alebo rozšíriť. Obsahuje popis

- použitých technológií,
- popis balíčkov a tried s možnosťou ich rozšírenia,
- popis tabuliek v databáze.

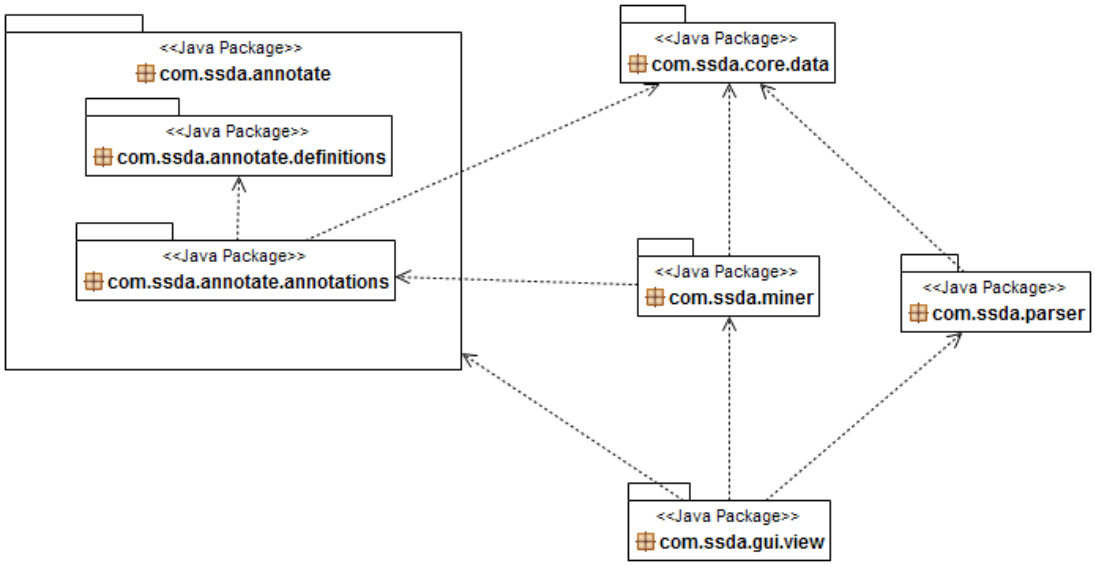
Pre dobrú orientáciu sa ale predpokladá, že programátor je oboznámený so spôsobom implementácie, popísaným v kapitole 4. Kompletná programátorská dokumentácia popisujúce jednotlivé triedy je vygenerovaná pomocou nástroja JavaDoc a je uložená na CD priloženom k tejto práci.

6.1 Použité technológie

- Aplikácia bola vyvinutá v jazyku *Java* verzie 8, aj kvôli jej nezávislosti na operačnom systéme.
- Pri vývoji bolo použité vývojové prostredie Eclipse a nástroj Maven pre kompiláciu zdrojových súborov.
- Ako databázový systém bol zvolený *PostgreSQL* verzie 9.4, ktorý patrí medzi najrozšírenejšie DBMS. Pre prístup do databáze je použitá knižnica *JDBC*, ktorá je k dispozícii v Maven repozitári, a to konkrétne *groupId:org.postgresql, artifactId: postgresql, version: 9.4-1201-jdbc41*.
- Ako OCR aplikácia bol použitý *TesseractOCR* verzie 3.02, ktorý sa z aplikácie spúšťa z príkazového riadku.
- Pre vývoj grafického rozhrania bola použitá JavaFX, kvôli lepšej architektúre (MVC).

6.2 Prehľad balíčkov a tried

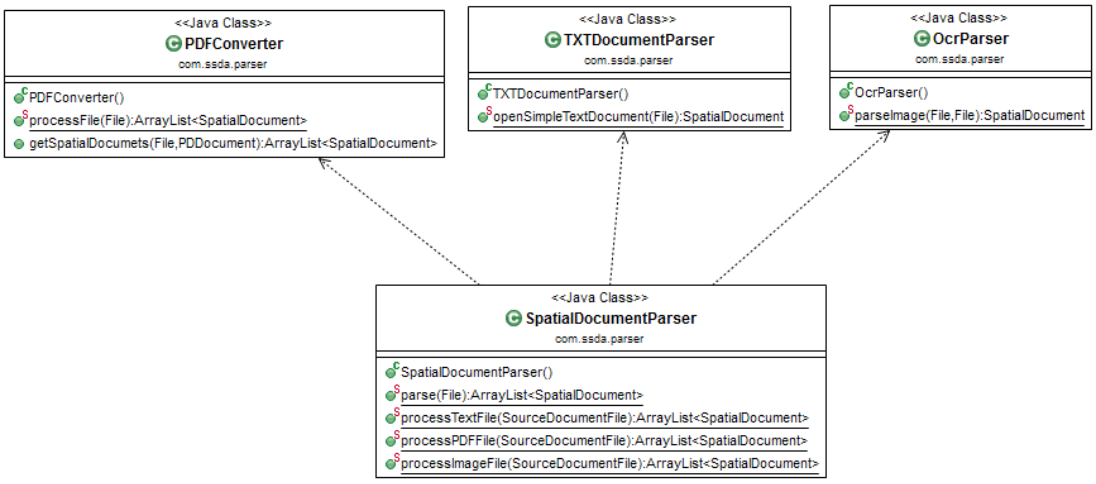
- Balíček *com.ssda.core.data* obsahuje základné dátové triedy: *SpatialDocument, SpatialLine, SpatialWord, AnnotatedPhrase, ImportantPhrase*, ktoré sú popísané v implementácii (viď kapitola Implementácia riešenia). Balíček ďalej obsahuje :
  - triedu *SourceDocumentFile*, slúžiacu ako datakeeper informácií o zdrojovom súbore (meno a typ),
  - triedu *BaseDBObject* ako rodičovskú triedu *Spatial objekov*, pre jednoduché ukladanie dát do databázy.
- Balíček *com.ssda.parser* obsahuje triedy majúce na starosti parsovanie súborov, ktorých výstupom je *SpatialDocument*. Balíček je bližšie popísaný v sekcii Balíček Parser.



Obrázok 6.1: Prehľad závislostí hlavných balíčkov

- Balíček *com.ssda.annotate* obsahuje všetky triedy, ktoré anotujú *SpatialDocument*. Pre viac informácií viď sekciu Balíček Anotátora.
- Balíček *com.ssda.miner* má na starosti nájdenie šablóny k *SpatialDocumentu* a vyhodnotenie podobnosti. Popis jeho tried je v sekcii Balíček Data Miner.
- Balíček *com.ssda.gui* obsahuje triedu *Main* pre spustenie aplikácie a grafické rozhranie.

6.2.1 Balíček Parser



Obrázok 6.2: Hlavné triedy parsovacieho balíčka

Hlavnou triedou balíčka *com.ssda.parser* je trieda *SpatialDocumentParser*. Tá funguje ako rozcestník pre parsovanie súborov podľa ich typu. Ak je súbor typu:

- text - parsovanie sa vykoná v triede TXTDocumentParser;

- pdf - parsovanie sa vykoná v triede PDFConverter;
- png,pjeg,bmp - parsovanie sa vykoná v triede OCRParser, viac o parsovaní obrázkov je popísané v sekcii OCR a balíček OCR.

Ako je vidieť na obrázku 6.2, *SpatialDocumentParser* predáva konkrétnemu parseru súbor, ktorý vracia jeden alebo niekoľko *SpatialDocument*ov (podľa počtu stránok v dokumente).

Pridanie nového parseru, alebo nahradenie existujúceho je možné práve v triede *SpatialDocumentParser*. Tá používa pomocnú triedu *SourceDocumentFile*, ktorá rozoznáva typ dokumentu. Preto pri pridávaní parseru pre nový typ súborov je potrebné upraviť aj triedu *SourceDocumentFile*.

6.2.2 Balíček Anotátora

Obrázok 6.3: Hlavné triedy anotátora

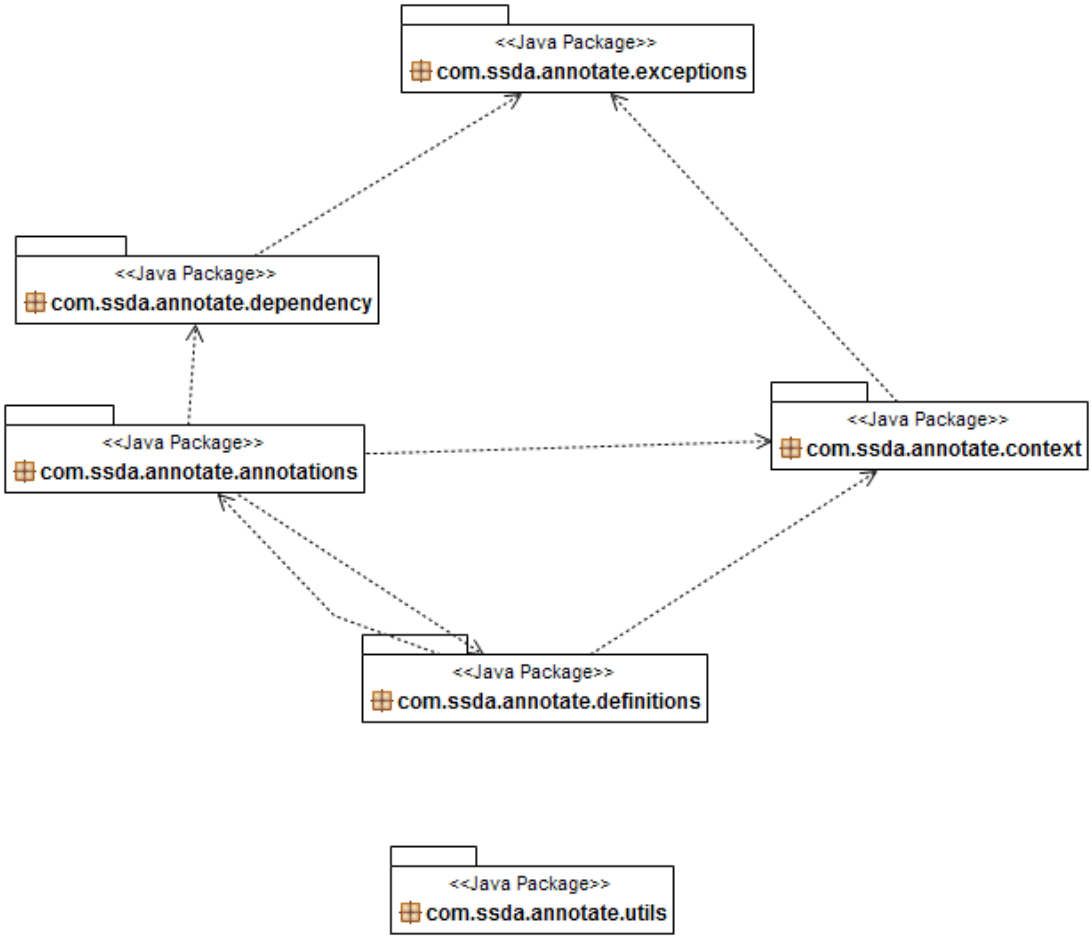
Hlavnými triedami anotátora sú:

- *AnnotationDefinition* - trieda slúžiaca ako datakeeper, ktorá obsahuje definície anotačných pravidiel.
- *AnnotationBuilder* - statická trieda, ktorá:
  1. načíta dáta zo súboru alebo XML dokumentu do *AnnotationDefinition*;
  2. postupne prechádza definície anotačných pravidiel v *AnnotationDefinition*, ktoré inštanciuje a pridáva do výsledného *Annotator*;

3. vráti vytvorený *Annotator*.

AnnotationBuilder tiež testuje správnosť definícií anotačných pravidiel a ich topologickú závislosť.

- *Annotator* - obsahuje funkcie, ktoré anotujú dokument: `annotateByConstants`, `annotatByRegexps`, `anotateWithContext` a `annotateByComplexTypes`. Funkcia `shrinkPhrases` odstraňuje prípadné duplicity fráz.
- *AnnotatorUtil* - Postupne volá jednotlivé funkcie anotátoru nad *SpatialDocument*om.



Obrázok 6.4: Schéma balíčkov anotátoru

Balíček anotátoru sa rozdeľuje na niekoľko častí:

- Balíček *exceptions* - obsahuje definície výnimiek.
- Balíček *dependency* - obsahuje triedy pre topologické usporiadanie a detekciu cyklov. Definície anotačných pravidiel sa pred anotovaním typologicky zoradia a postupne vykonávajú. Cyklická závislosť by viedla k nesprávnemu poradiu, v akom sa anotácie majú vykonávať.
- Balíček *definitions* - konfigurácia a definícia anotácie (napríklad. odkaz na číselník).

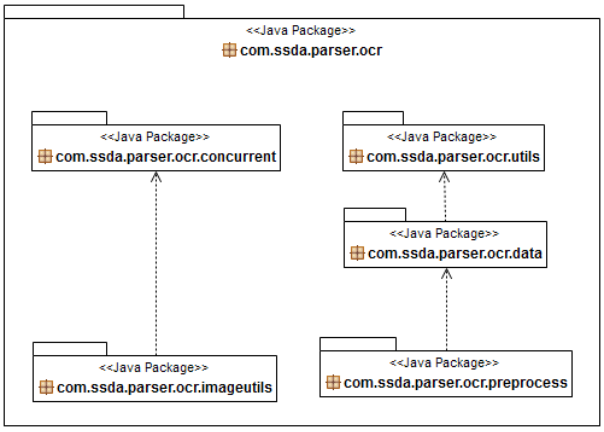
- Balíček *annotations* - triedy, ktoré reprezentujú inštanciované anotačné pravidlá (napríklad, prvky číselníku).
- Balíček *context* - obsahuje triedy pre prácu kontextom fráz.
- Balíček *utils* - obsahuje triedu *AnnotatorUtil* a triedu pre nájdenie tabuľky v dokumente.

V schéme na obrázku 6.4 je vidieť, že balíčky *annotations*, *definitions* sú úzko zviazané. Je to dané tým, že inštancia pravidla si drží odkaz na svoju definíciu, zatiaľ čo definícia vytvára inštanciu pravidla.

Pre pridanie nového spôsobu anotovania *SpatialDocumentu* stačí upraviť funkciu `AnnotatorUtil.annotate(SpatialDocument, Annotator)`. V prípade, že nový spôsob anotovania vyžaduje konfiguráciu, je potrebné:

- V balíčku *com.ssda.annotate.definitions* pridať novú triedu zdedenú od `BaseAnnotationDefinition`, ktorá reprezentuje definíciu a konfiguráciu pravidla.
- V balíčku *com.ssda.annotate.annotations* vytvoriť triedu, ktorá predstavuje inštanciu pravidla `BaseAnnotationDefinition`. Niekedy to ale nie je potrebné. Nový zdroj číselníkov bude vytvárať anotácie triedy *ConstantAnnotation*.
- Upraviť funkciu `AnnotationBuilder.instantiate(AnnotationDefinition)`, ktorá z konfigurácie pravidla vytvára inštanciu.
- Ak je to treba, pridať metódu do triedy *Annotator* pre anotovanie dokumentu a tú volať z metódy `AnnotatorUtil.annotate(SpatialDocument, Annotator)`.

6.2.3 OCR a balíček OCR



Obrázok 6.5: Schéma balíčkov OCR

Balíček OCR parseru sa rozdeľuje na niekoľko častí:

- concurrent - obsahuje triedu *ThreadPool*, ktorá spúšťa paralelné procesy.

- imageutils - obsahuje triedu *CannyEdgeDetector*, ktorá v obrázku detekuje hrany. Pri detekovaní používa triedu *ThreadPool*, pre paralelné spracovanie obrázku.

- data - obsahuje triedy reprezentujúce dáta, ako bod (Point), vlastná reprezetícia obrázku (PixelizedImage), alebo jednu stránku dokumentu (ImagePage).

- preprocess - obsahuje nástroje pre predspracovanie obrázka:
  - *ImgeSplitDetector* - nástroj pre detekciu deliacich čiar.
  - *ImageCutter* - nástroj pre rozdelenie obrázka podľa deliacich čiar.
  - *ImageDeskew* - detektor a vyrovnávač obrázku.
  - *OCRResultParser* - zpracovanie výsledku z OCR aplikácie.
  - *PixelizeImageProcessor* - prevádza bitmapu na *PixelizedImage*.

- utils - obsahuje nástroje pre rutinne funkcie ako manažovanie súborov a adresárov, konvertovanie farieb, alebo formátovanie výpisu do logu.

Samotný parser implementuje trieda *com.ssda.parser.OcrParser*. Tá postupne predspracuje obrázok, rozseká ho na menšie časti nad ktorými je volaná OCR aplikácia. Z výsledkov vytvorí slová *SpatialWord* a následne dokument *SpatiaDocument*.

Samotné volanie TessaractOCR sa vykonáva vo funkcii `ArrayList<SpatialWord> runOcr(ArrayList<ImageBlockCut>)`, kde výsledok spracováva funkcia `ArrayList<SpatialWord> getTextFromOcr(ImageBlockCut block, File htmlFile)`. Preto v prípade zmeny volania Tesseractu, alebo zmenu OCR aplikácie je potrebné upraviť tieto funkcie.

6.2.4 Balíček Data Miner

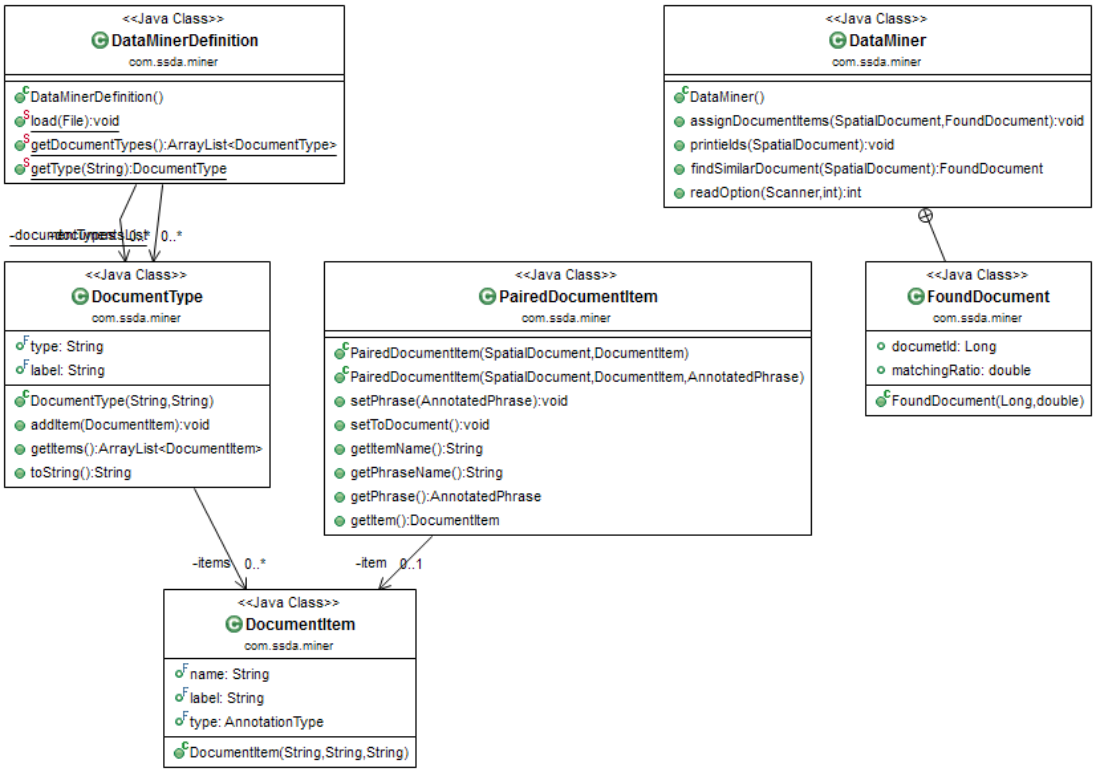
Balíček Data Miner má na starosti nájdenie šablóny k *SpatialDocumentu*, vyhodnotenie podobnosti a získanie dôležitých dát zo šablóny (odtiaľ názov *data miner*). Spôsob, akým sa sa zisťuje podobnosť dokumentu a šablóny a ako sú identifikované dôležité dáta, je popísaný v sekcii 4.5.3. Preto v tejto časti búdu popísané len triedy balíčka a aká je ich rola. Schéma tried je zobrazená na obrázku 6.6.

- *DataMinerDefinition* - nástroj pre načítavanie typov dokumentov a ich položiek.

- *DocumentType* - predstavuje definíciu typu dokumentu, to jest názov typu a jeho položky.

- *DocumentItem* - predstavuje položku typu dokumentu, napríklad telefónne číslo objednávateľa.

- *PairedDocumentItem* - pomocná trieda, ktorá mapuje *SpatialDocument*, *DocumentItem* a frázu. Pomocou nej sú zo šablóny identifikované dokumentové položky v *SpatialDocumente*.



Obrázok 6.6: Schéma balíčkov data mineru

- *DataMiner* - hlavná trieda, ktorá k *SpatialDocumentu* dohľadá šablónu, z ktorej určí typ dokumentu a následne identifikuje dokumentové položky.

6.3 Tabuľky

Súčasťou aplikácie je aj databáza, ktorá slúži pre ukladanie a vyhľadávanie dokumentov a šablón. V tejto sekcii budú popísané len hlavné tabuľky, ktoré tvoria jadro vyhľadávacieho procesu. Ostatné tabuľky vyskytujúce sa v databáze sú číselníky pre anotačné pravidlá.

Tabuľka **`ssda_spaword`** obsahuje dáta *SpatialWord*u:

- `id` - id slova
- `txt` - text slova
- `ltx, lty, rdx, rdy` - MOO slova
- `lineid` - referencia na rodičovskú SpatialLine

Tabuľka **`ssda_spaline`** obsahuje dáta *SpatialLine*:

- `id` - id riadku
- `txt` text - text v riadku
- `ltx`, lty,rdx, rdy - MOO slova
- `docid` - referencia na SpatialDocument
- `rownum` - čislo riadku

Tabuľka **ssda_annotatedphrase** obsahuje dáta anotovaných fráz:

- `id` - id fráze
- `probability` - pravdepodonosť
- `annottype` - typ frázy
- `txt` - textový obsah frázy
- `docid` - referencia na SpatialDocument

Tabuľka **ssda_annotated_words** predstavuje prepojenie slova a anotovaných fráz:

- `annotation` - referencia na anotovanú frázu
- `word` - referencia na SpatialWord

Tabuľka **ssda_important_phrases** obsahuje *vyhľadávací záznamy*, teda *ImportantPhrases* a ich vzájomné polohy:

- `phraseid` - referencia na frázu 1
- `documentid` - referencia na SpatialDocument
- `phrasetype` - typ frázy 1
- `area` - vektor / oblasť
- `referencedphrase` - referencia na frázu 2
- `referencedtype` - typ frázy 2
- `occurence` - číslo výskytu frázy 1
- `referencedoccurence` - číslo výskytu frázy 2

Tabuľka **ssda_document_items** obsahuje *DocimentItems*:

- `id` - id záznamu
- `itemtype` - typ položky
- `phrase` - referencia na frázu v dokumente
- `docid` - referencia na SpatialDocument

Tabuľka **ssda_spadocument** dáta *SpatialDocument*u:

- `id` - id dokumentu
- `filename` - meno súboru, z ktorého dokument pochádza
- `documenttype` - typ dokumentu
- `documentsubtype` - vlastné označenie typu dokumentu
- `vipcount` - počet záznamov v *ssda_important_phrases* tohoto dokumentu - pre rýchlejší výpočet výslednej podobnosti a filtrovanie príliš odlišných dokumentov.

7. Praktické experimenty

Hlavnou úlohou praktických experimentov bolo otestovať funkčnosť modelu nad reálnymi dátami, ktorého implementácia je popísaná v kapitole 4 (SSDA aplikácia). Výsledky experimentov a spôsob ich merania je popísaný v sekcii Merania spoľahlivosti.

Kapitola tiež obsahuje porovnanie obecnejších vlastností SSDA s existujúcimi riešeniami, popísanými v kapitole Existujúce riešenia.

7.1 Merania spoľahlivosti

Ako referenčná aplikácia bola zvolená LANA. Dokáže rozoznávať viac typov dokumentu, je možné zistiť mieru podobnosti dokumentov a pre nachádzanie šablón používa odlišný prístup.

7.1.1 Spôsob merania a príprava

Pre testovanie bolo použitých 127 PDF dokumentov. Keďže LANA predpokladá, že v PDF dokumentoch sa nachádzajú len obrázky (oskenované dokumenty), boli tieto dokumenty konvertované do obrázkov, ktoré následne boli použité ako vstupné dáta. Dokumenty, ktoré boli použité pri testovaní sa nachádzajú na priloženom CD v adresári *testovacieData*.

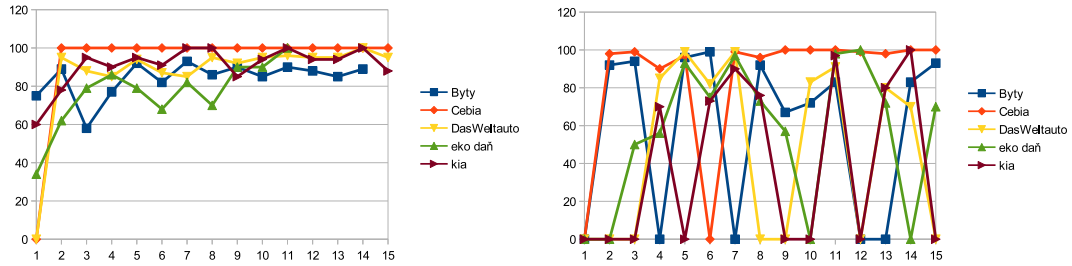
Pri meraní sa kontrolovala správnosť nájdených položiek a miera podobnosti vyšetrovaného dokumentu so šablónou. V prípade, ak bol dokument nesprávne rozoznaný alebo jeho položky boli nesprávne rozoznané, boli tieto nesprávne rozoznané dáta ručne opravené a uložené. Tým sa aplikácia učila správne rozoznávať daný dokument. Učenie a merania boli robené po jednotlivých dokumentoch. Kebyže by boli dokumenty spracovávané dávkovo, skresľovalo by to výsledky.

Celkovo bolo použitých 9 typov dokumentov, ktoré sú označené skratkami [1]:

- Byty - výpis obecného bytu,
- Cebia - osvedčenie Cebia reportu,
- DasWeltauto - popis auta autobazaru,
- Kia - popis auta autobazaru,
- Eko daň - výpis colnej správy o platnosti ekologickej dane,
- IV - vyhláška o insolvencii,
- IR - detail insolventného riadenia,
- Záměr - investičný zámer,
- CNS - výpis evidencie z registru ministerstva kultury.

Typy dokumentov boli rozdelené na dve kategórie:

[1] V prílohe sa testovacie dáta nachádzajú v rovnako pomenovaných adresároch.



(a) SSDA (b) LANA

Obrázok 7.1: Miera podobnosti Lepšie štrukturovaných dokumentov

1. Lepšie štrukturované, kam patrili dokumenty typu: Byty, Cebia, DasWeltauto a Eko daň.

2. Horšie štrukturované dokumenty, alebo dokumenty, ktorých formát dát bol nejednotný. Sem patrili dokumenty typu: IV, IR, Záměr a CNS.

V LANA aj SSDA boli definované typy dokumentov tak, aby obsahovali rovnaké položky. V SSDA bolo naviac potrebné rozšíriť definície anotačných pravidiel tak, aby dokázali anotovať prvky v dokumentoch.

Pred meraniami boli zmazané všetky naučené dáta, aby bolo vidieť, ako rýchlo sa dokážu naučiť nový typ dokumentu.

7.1.2 Namerané výsledky

Grafy na obrázkoch 7.1(a) a 7.1(b) zobrazujú, ako sa aplikácie dokážu učiť v čase. Číslo na horizontálnej ose predstavuje poradie dokumentu. Je teda logické, že prvá hodnota je často nulová, keďže aplikácia sa daný typ dokumentu ešte nenaučila rozoznávať. Hodnoty v grafe predstavujú podobnosť skúmaného dokumentu s nájdenou šablónou. Veľkosti zhody sú uvádzané percentuálne.

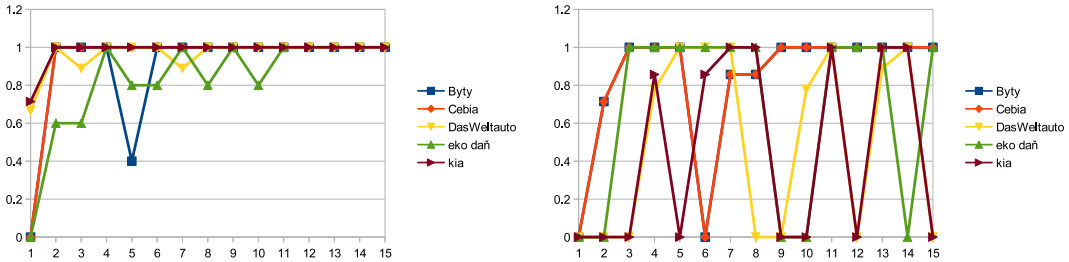
Z grafov 7.1(a) a 7.1(b) vidieť, že SSDA sa dokáže oveľa rýchlejšie učiť a má spoľahlivejšie výsledky ako LANA. LANA často nadobúda 0-ové hodnoty. Je to spôsobené tým, že dokumenty s podobnosťou menšou ako 50% sú ignorované, a teda výsledná zhoda je 0. Miera podobnosti u aplikácie LANA je počítaná z plochy, ktorou sa dokumenty prekrývajú, preto má tiež horšie výsledky než SSDA.

Grafy na obrázkoch 7.2(a) a 7.2(b) zobrazujú mieru úspešnosti pri identifikácií položiek v dokumente. Pretože každý typ dokumentu obsahuje iný počet položiek, výsledná úspešnosť je meraná ako ich podiel, teda:

$$uspesnost = \frac{spravne\,rozoznane\,polozky}{pocet\,poloziek\,v\,danom\,type\,dokumentu}$$

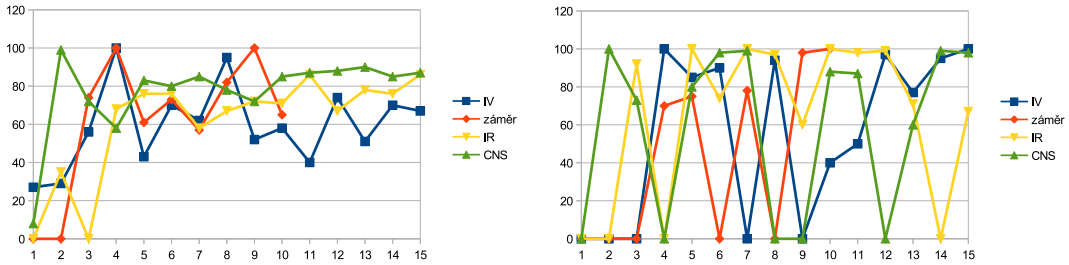
.

Namerané výsledky ukazujú, že dokumenty typu *Kia* a *Cebia* začala správne rozoznávať už od druhého dokumentu, pričom najdlhšie sa "učila" *Eko daň* - až 10 dokumentov. LANA mala najväčší problém s dokumentom typu *Kia*. Ten totiž obsahuje premenlivý počet obrázkov, čo spôsobilo väčšie rozdiely pri porovnávaní. Najlepšie rozoznávala dokumenty typ *Cebia* a *Byty*.



(a) SSDA (b) LANA

Obrázok 7.2: Podiel správne identifikovaných položiek pre Lepšie štrukturované dokumenty



(a) SSDA (b) LANA

Obrázok 7.3: Miera podobnosti pre Horšie štrukturované dokumenty

Grafy na obrázkoch 7.3(a) a 7.3(b), podobne ako grafy 7.1(a) a 7.1(b), zobrazujú proces učenia sa aplikácií, tento krát ale na horšie štrukturovaných dokumentoch. Hodnoty v grafoch predstavujú mieru podobnosti skúmaného dokumentu s nájdenou šablónou.

U priebehu učenia sa SSDA je vidieť, že s počtom spracovaných dokumentov stúpa miera rozpoznania, no LANA má nestabilné výsledky. Najhoršie výsledky v oboch aplikáciách má dokument typu *IV*. Ten totiž nie je skoro vôbec štrukturovaný. Skôr sa jedná o bežný dokument. Jeho náhľad je zobrazený na obrázku 7.4. Jeho dôležité položky majú nestále miesto, alebo sa dokonca lámu na konci riadkov. Preto je pre anotátor správne anotovať text. Rovnako tak LANA nedokáže v dokumente určiť presné miesto, odkiaľ získavať dáta.

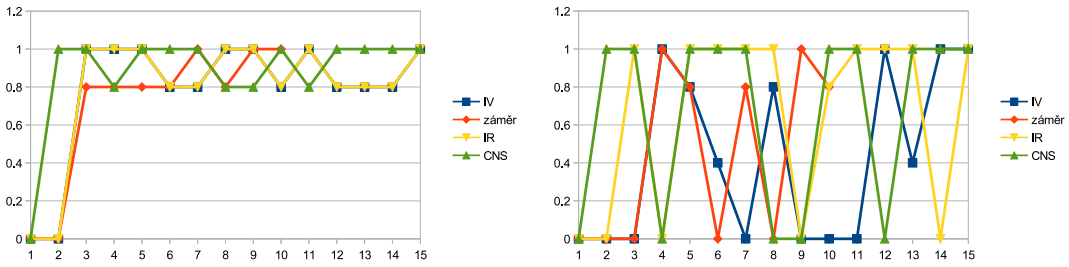
Vyhláška

takto:

Obrázok 7.4: Ukážka layoutu dokumentu typu IV

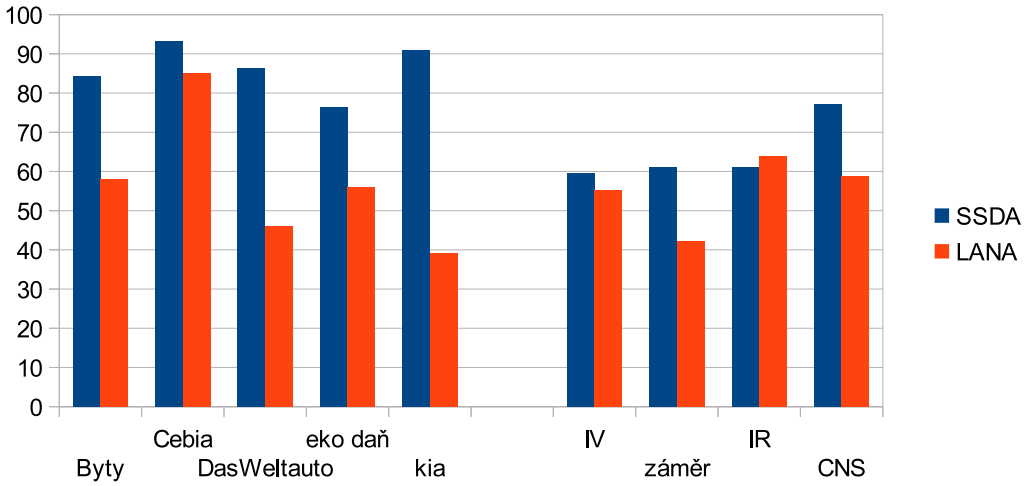
Grafy na obrázkoch 7.5(a) a 7.5(b) zobrazujú úspešnosť pri identifikácií položiek v horšie štrukturovaných dokumetoch. Rovnako ako v grafoch 7.2(a) a 7.2(b), je hodnotou podiel správne rozoznaných položiek k celkovému počtu položiek v dokumente. Z grafov je vidieť, že ani jedna aplikácia nemá 100% úspešnosť, ale zatiaľ čo SSDA správne nachádza 80% položiek, LANA nemá stabilné výsledky.

Celková miera podobností dokumentov a ich šablón je zobrazená na obrázku



(a) SSDA (b) LANA

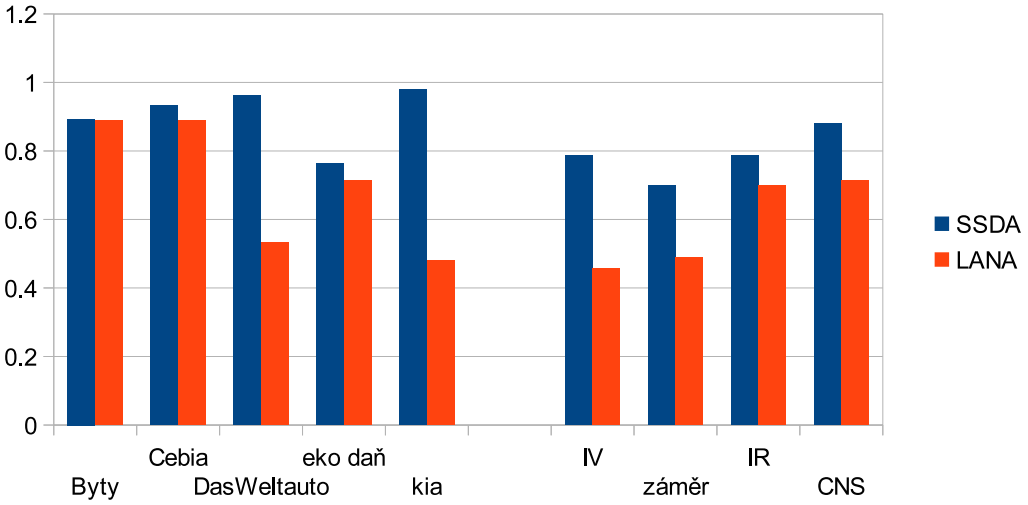
Obrázok 7.5: Podiel správne identifikovaných položiek pre Horšie štrukturované dokumenty



Obrázok 7.6: Priemerná veľkosť zhody dokumentu a šablóny podľa typu dokumentu

7.6. Údaje v ľavej časti dokumentu patria lepšie štrukturovaným dokumentom a v pravej časti horšie štrukturovaným dokumentom. V štatistike nie sú zahrnuté prvé (teda nerozoznané) dokumenty, aby neskresľovali celkový priemer.

Z grafu 7.6 je vidieť, že najväčší rozdiel hodnôt medzi LANA a SSDA je v dokumentoch typu *Kia* a *DasWeltauto*. Je to spôsobené tým, že len tieto dva typy dokumentov obsahujú obrázky, ktorých počet je premenlivý, no štruktúra textu je rovnaká. Naopak podobné výsledky mali na dokumente typu *IR*. Ten má stabilný a dobre štrukturovaný layout, avšak SSDA parser kvôli malej medzere, ktorá odlišuje názov položky a jej dáta, tieto slová spájal do jedného riadku. Anotátor tak nedokázal správne rozoznať niektorá frázy. LANA v tomto type dokumentu má dokonca lepšie výsledky než SSDA.



Obrázok 7.7: Priemerná úspešnosť identifikácie položiek v dokumente

Grafy na obrázku 7.7 zobrazujú priemernú úspešnosť správne rozoznaných položiek podľa jednotlivých typov dokumentov. Do štatistík nie sú zahrnuté prvé dokumenty (kvôli skresleniu výsledkov). Podobne ako v grafe 7.6, sú výsledky lepšie štrukturovaných dokumentov v ľavej a horšie štrukturovaných dokumentov v prvej časti grafu.

Zaujímavým pozorovaním je, že miera podobnosti dokumentov typu *Byty* vychádza lepšie pre SSDA, pričom úspešnosť správne rozoznaných položiek je u oboch aplikácií takmer rovnaká.

Z porovnaním výsledkov v grafoch reffig:statpercent a 7.7 je možné vyvodiť, že veľkosť zhody dokumentu a šablóny odpovedá počtu správne rozoznaných položiek. To potvrdzuje predpoklad, že ak je k dokumentu nájdená šablóna, je pomocou nej možné identifikovať položky v dokumente.

7.2 Porovnanie vlastností existujúcich riešení

Následujúca tabuľka porovnáva niektoré vlastnosti existujúcich riešení.

	vlastnosť	SSDA	LANA	Datamolino	ReadSoft Invoices
Predspracovanie	Používa descale obrázku	áno	áno	?	áno
	Používa odstraňovanie šumu	nie	nie	?	áno
OCR	Aké OCR používa	Tesseract	Tesseract	Vlastné	Používa udajne 3, pre kontrolu správnosti
	Dokáže spracovať aj ručne písané znaky	nie	nie	?	áno
	Podpora rôznych jazykov	áno	áno	áno	áno
	Podpora rôznych fontov	áno	áno	?	áno
	Je možné naučiť aplikáciu nový font	nie priamo	áno	?	nie
	Aký typ machine-learning algoritmu používa	vzájomná poloha dôležitých fráz	prekrývanie blokov textu	?	anotácia textu, následne vyhľadá podľa odosielateľa šablónu dokumentu
	Dokáže software anotovať časti textu?	áno	áno	?	áno

	Dokáže software identifikovať obrázky na dokumente	nie	nie	?	čiarkové kódy
	Pracuje software s obrázkami v dokumente?	nie	nie	?	áno
	Vie extrahovať dôležité položky z dokumetu?	áno	áno	áno	áno
	Aký typy dokumentov rozoznáva	polo-štrukturované dokumenty	polo-štrukturované dokumenty	faktúry, proformy, účtenky	polo-štrukturované dokumenty
	Rozozná ten istý dokument v rôznych rozlíšeniach	áno	áno	áno	áno
	dokáže spracovať viacstránkové dokumenty	nie	nie	áno	áno
	dokáže identifikovať tabuľku v dokumente	áno	nie	?	áno
	rýchlosť spracovania dokumentu 2 sek pre PDF, 40 sek pre obrázky	15 sek	30 sek	3 min - 3 dni	6-12 sek
	je nutná spolupráca koncového užívateľa	áno, pri zlom rozoznaní	áno, pri zlom rozoznaní	nie, všetko robí operátor	áno, ak pravdepodobnosť rozoznaných dát je menšia ako nastavená limita,
Výsledok	zobrazuje umiestnenie zroja dát na obrázku	nie	áno	nie	áno
	umožňuje export dát	nie	áno	áno	áno
	ako sa kontroluje kvalita výsledku	užívateľ	užívateľ	kontroluje operátor	logicky (napr. že sedia súčty), protiúčtovo, alebo či sú správne údaje o dodávateľovi
	Kde sa vykonáva extrakcia dát	lokálne	na vzdialenom serveri	na vzdialenom serveri	na vzdialenom serveri

Tabulka 7.1: Porovnanie vlastností existujúcich riešení

7.3 Pozorovania

7.3.1 Vyhľadávanie slov v texte

Teoreticky by najlepšou dátovou štruktúrou mal byť regulárny výraz. To ale bolo pri testoch vyvrátené. Ako testovacie dáta boli vybrané 4 textové súbory, ktoré mali spolu 885 slov. Vyhľadávali sa slová zo zoznamov: mien, priezvisk a miest. Spolu mali 342280 slov. Pre každý zoznam sa vytvoril jeden regulárny výraz (text, ktorý odpovedal regulárnemu výrazu bolo potom možné označiť ako meno, priezvisko, alebo mesto). Celkový priemerný čas, vyhľadávania slov v súboroch bol 45.5 sekúnd. Pri vyhľadávaní konštánt v texte pomocou *HashMap*y, bolo nutné text rozdeliť na slová a tieto slová následne vyhľadať v *HashMap*e. Celkový priemerný čas vyhľadávania slov bol 0.21 sekúnd.

7.4 Výhody riešenia

Rýchlosť učenia

Aplikácia sa z mála dokumentov naučila správne rozoznávať typ dokument a identifikovať dôležité položky. Dokonca dokázala správne identifikovať typ dokumentu, ktorého šablónu sa ešte "nenaučila". Za to môžu samozrejme správne anotované fráze a fakt, že model umožňuje porovnávať dokumenty, ktorého položky majú rôznu veľkosť.

Neštrukturované dokumenty

Aplikácia síce s väčšou chybovosťou, ale dokázala rozoznať dokumenty, ktoré boli menej štrukturované.

Rôznorodosť dokumentov

Experimenty preukázali, že aplikácia je schopná naučiť sa rozoznávať rôzne typy polo-štrukturovaných dokumentov.

Rozoznanie ešte nenaučených dokumentov

Aplikácia z časti dokázala správne identifikovať typ dokumentu a jeho položky, ktorý spracovávala prvý krát. To vďaka tomu, že už spracovala iný dokument (z inej šablóny), no rovnakého typu.

7.5 Nevýhody riešenia

OCR

Samozrejme najväčším problémom je OCR aplikácia, ktorá nedokáže previesť obrázok do textu tak dobre, ako človek. Problém prevodu obrázku do textu by sa dal zmenšiť kvalitnejším skenovaním dokumentov.

Sklon dokumentu

Ďalším faktorom je sklon dokumentu. Ak dokument formátu A4 je naklonený o viac ako 0.6 stupňa, očíslovanie *SpatialLine* a teda usporiadanie slov v dokumente, môže byť nesprávne, čo môže mať za následok, že v dokumente nebude správne rozoznaná tabuľka, ktorá zaberá celú šírku dokumentu.

Neúplné číselníky

Ak v číselníku chýba názov ulice, *Annotator* ju neidentifikuje a tým pádom celá adresa nemusí byť rozoznaná. Tá môže pre konkrétny typ dokumentu hrať kľúčovú rolu a dokument nemusí byť identifikovaný s tak veľkou presnosťou.

8. Záver

Cieľom práce bolo navrhnúť a implementovať algoritmus, ktorý dokáže analyzovať dokument, nájsť podobný a podľa neho priradiť typ dokumentu a priradiť význam jednotlivým blokom textu.

Ako vstupné dokumenty boli použité vygenerované PDF súbory a obrázky. Pri obrázkoch bolo dôležité, aby rozoznaný text bol v čo najlepšej kvalite. Vývoj modulu, ktorý rozrezával obrázky aby TesseractOCR vracal lepšie rozoznaný text, bol časovo oveľa náročnejší než sa predpokladalo.

Pri testovaní a vývoji aplikácie sa prišlo na možné nedostatky a rezervy, ktorých riešenia by mohli byť označené ako rozšírenie tejto práce:

- Rozoznávanie je veľmi závislé na kvalitne rozoznanom texte, ktorý aby bol správne rozoznaný, musí byť uložený v pravidlách. Ako možné rozšírenie tejto práce by bolo navrhnúť spôsob, ako v dokumente vyhľadávať slová alebo regulárne výrazy s chybou.

- V dokumente sa nepracuje s obrázkami a grafickými komponentami dokumentu. Prevedením týchto grafických komponentov na anotované frázy by spresnilo vyhľadávanie.

- Tabuľky sa zatiaľ rozoznávajú len z usporiadania slov v dokumente. Spresnenie identifikácie tabuliek by viedlo k lepším výsledkom.

- Rozoznávanie viac-stránkových dokumentov. Problémom je správne identifikovať záhlavie a zápätie dokumentu vo fáze anotácie dokumentu. Tento krok by ale eventuálne mohol byť prevedený až pri identifikácií šablóny.

- Nahradiť anotátor oveľa rozvinutejším anotátorom.

Zoznam použitej literatúry

[1] Fayez Tarsha-Kurdi, Tania Landes, Pierre Grussenmeyer. *Hough-transform and extended RANSAC algorithms for automatic detection of 3D building roof planes from Lidar data.*, Espoo, Finland, 2007, pp.407-412., halshs-00264843.

[2] Neznámi autor Anydoby, *Automatic image deskew in Java.* 27-10-2010, http://anydoby.com/jblog/en/java/1990

[3] Faisal Shafait, Daniel Keysers, and Thomas M. Breuel. *Performance Comparison of Six Algorithms for Page Segmentation.* 2006,D-67663, http://www.keysers.net/daniel/files/Shafait–Layout-Analysis-Comparison–DAS2006.pdf

[4] Canny J. *A Computational Approach To Edge Detection.* IEEE Trans. Pattern Analysis and Machine Intelligence, 1986, pp. 679–698

[5] Tom Gibara *Canny Edge Detector Implementation.* 2011,http://www.tomgibara.com/computer-vision/canny-edge-detector

[6] Shelley Powers *Practical RDF.* 2009, ISBN:978-0-596-00263-3

[7] Peter Buneman *Semistructured data*, Proc. ACM Symposium on Principles of Database Systems, pages 117-121, Tucson, AZ., 1997. Abstract of invited tutorial 2002, ISBN:1-4020-0489-3

[8] Josef KOLÁŘ *Teoretická informatika. 2. vyd.*, Praha : Česká informatická společnost, 2004. 205 s. ISBN 80-900853-8-5

[9] Antonin Guttman *R-Trees: A Dynamic Index Structure for Spatial Searching*, Proc. 1984 ACM SIGMOD International Conference on Management of Data, pp. 47-57., ISBN 0-89791-128-8

Seznam obrázků

Zoznam tabuliek

Prílohy

K práci je priložený DVD-ROM, ktorý v jednotlivých adresároch obsahuje:

- *install* - inštalačné balíčky a dáta potrebné pre beh aplikácie
- *progrDoc* - programátorskú dokumentáciu vygenerovanú zo zdrojových súborov
- *ssda* - adresár obsahujúci SSDA aplikáciu
- *statistika* - adresár obsahuje dokument s výsledkami meraní
- *testovacieData* - dáta, ktoré boli použité pri testovaní
- *zdrojoveKody* - zdrojové kódy aplikácie
- *diplPraca.pdf* - text tejto diplomovej práce