VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

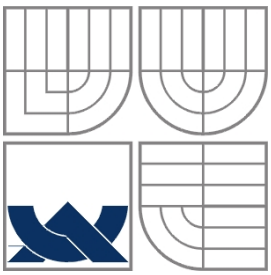
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

EXTRAKCE SÉMANTICKÝCH VZTAHŮ Z TEXTU

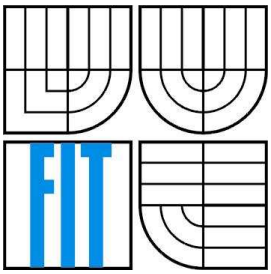
DIPLOMOVÁ PRÁCE
MASTERS'S THESIS

AUTOR PRÁCE Bc. Milan Pospíšil
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

EXTRAKCE SÉMANTICKÝCH VZTAHŮ Z TEXTU

EXTRACTION OF SEMANTIC RELATIONS FROM TEXT

DIPLOMOVÁ PRÁCE
MASTERS'S THESIS

AUTOR PRÁCE — Bc. Milan Pospíšil
AUTHOR

VEDOUCÍ PRÁCE — Smrž Pavel, doc. RNDr., Ph.D.
SUPERVISOR

BRNO 2010

Abstrakt

Dnes existuje spousta polostrukturovaných dokumentů, které by bylo vhodné převést do strukturované podoby. Cílem práce je navrhnout systém, který umožní tuto práci co nejvíce zautomatizovat. To může být obtížný problém, protože většina těchto dokumentů není generovaná automaticky počítačem a systém proto musí tolerovat nepřesnosti. Protože je třeba i určité sémantické pochopení problému, bude systém testován na doméně sady dokumentů zápisů ze schůzek.

Abstract

Today exists many semi-structured documents, whitch we want convert to structured form. Goal of this work is create a system, that make this task more automatized. That could be difficult problem, because most of these documents are not generated by computer, so system have to tolerate differences. We also need some semantic understanding, thats why we choose only domain of meeting minutes documents.

Klíčová slova

extrakce informací, wrapper, Microsoft Word, klasifikace textu

Keywords

information extraction, wrapper, Microsoft Word, text classification

EXTRAKCE SÉMANTICKÝCH VZTAHŮ Z TEXTU

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Pavla Smrže, doc. RNDr., Ph.D.
Další informace mi poskytli Ing. Marek Shmidt
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Milan Pospíšil
26.5.2010

Poděkování

Děkuji zejména Doc. Smržovi za jeho cenné rady a čas, který si našel na konzultace.

1 Obsah

1 Obsah ... 1
2 Úvod ... 3
3 Extrakce informací z dokumentů ... 4
3.1 Aplikace IE ... 4
3.2 Typy dokumentů ... 5
Strojově generované dokumenty ... 5
Částečně strukturované specifické dokumenty ... 5
Otevřený text ... 5
3.3 Přehled metod pro extrakci ... 6
Ručně definované nebo učící metody ... 6
Pravidlové nebo statistické metody ... 6
3.4 Pravidlové systémy ... 8
Rysy entit: ... 8
Pravidla pro rozpoznání entity ... 8
Překrývání entit a hierarchie ... 9
3.5 Statistické systémy ... 11
Neuronové sítě ... 11
Klasifikace ... 11
Perceptron ... 12
3.6 Lixto ... 15
4 Popis systému ... 16
4.1 Schéma systému pro extrakci informací z dokumentů ... 16
4.1.1 Převod z formátu Microsoft Word ... 17
4.1.2 Popis elementů ... 17
4.1.3 Převod do dat ... 18
4.2 Tvorba interního modelu dokumentu ... 19
4.2.1 Rozdělení na slova ... 19
4.2.2 Indexace slov ... 19
4.2.3 Rozdělení textu na stavební bloky ... 20
4.2.4 Tabulky ... 20
4.2.5 Specifické elementy dokumentu ... 21
4.3 Základní nástroje pro popis elementů ... 22
Definice vzoru elementů ... 23
Condition Pattern ... 24
Condition PatternReference ... 24

Rysy (Feature Pattern) 24
4.3.1 Vázací podmínky (Feature binding) 24
4.3.2 Alternativní názvy 25
4.4 Popis rysů a vázacích podmínek 26
4.4.1 Text Binding 26
4.4.2 Simple Number Binding 28
4.4.3 Element binding 28
4.4.4 Element boundary binding 29
4.4.5 Element Occurence 29
4.4.6 Další rysy 30
4.5 Algoritmus extrakce 33
4.6 Named Entity Recognition 34
4.6.1 Přidávání názvů 34
4.6.2 Analýza odkazů 35
4.7 Rozpoznávání času a data 37
4.7.1 Rozpoznávání času a data 37
4.7.2 Rozsah času 38
4.8 Akce 39
4.8.1 Add to pattern TextBinding 39
4.8.2 Delete on bind 39
4.9 Převod do dat 41
4.9.1 Příklad 42
4.10 Řešení problémů a chybné klasifikace 43
4.11 Uživatelské rozhraní 45
4.11.1 Základní uživatelské rozhraní 45
4.11.2 Uživatelské rozhraní pro editaci rysů (vlastností) 50
5 Experimenty 52
5.1 Řešení obecných problémů 52
5.1.1 Agendy 52
5.1.2 Hlavičková data 55
5.1.3 Tabulková data 56
5.2 Nebraska Wheat Agenda 57
5.3 Nebraska Wheat Meeting 60
5.4 Agenda (Kiwi) 61
5.5 Meeting (Kiwi) 61
5.6 Montreal Meeting Agenda 62
6 Závěr 63

2 Úvod

V současné době existuje spousta dat v databázích. Tato data jsou strukturovaná, dají se třídit, vyhledávat, transformovat. Lze z nich také pomocí pokročilých nástrojů dolovat další informace, které nejsou na první pohled zřejmé. Na druhou stranu existuje ještě větší množství dat, které strukturované nejsou. Zejména se jedná o webové stránky a dokumenty.

Tyto dokumenty obsahují užitečná data, které by bylo dobré mít ve strukturované formě. Ale jejich manuální procházení a plnění do databáze je nesmírně zdlouhavé, někdy až téměř nemožné. Pro webové stránky existuje řada nástrojů (wrapperů), které dokáží pomocí definice od uživatele vytáhnout potřebná data. Webové stránky nejsou jedinými zdroji dat. Tím dalším jsou textové dokumenty, například vytvořené v nástroji Microsoft Word. Vzhledem k tomu, že tento nástroj je velmi rozšířený, je třeba hledat způsoby jak popsat strukturu dokumentu, aby z něj mohl program automaticky (či poloautomaticky) vytáhnout data. To nemusí být vůbec snadné, protože tyto dokumenty jsou velmi často psány lidmi a nejsou tak dobře strukturované jako webové stránky, které obvykle generuje počítač. Tato práce si klade za cíl vytvořit nástroj, který umožní pomocí uživatelského rozhraní popsat dokument (či sadu dokumentů) tak, aby program byl schopen z nich vytáhnout informace s jistou tolerancí. Tento problém je pochopitelně obtížný, proto se práce zatím práce zaměřuje pouze na dokumenty zápisů ze setkání (meeting minutes). Je jasné, že pokud bude práce dostatečně obecná a úspěšná, půjde to rozšířit a použít na celou řadu různých dokumentů. Část práce se zabývá základní analýzou MS Word dokumentů, další část popisem jazyka pro popis struktury a poslední část popisem konkrétních dokumentů. Na rozdíl od klasického webového wrapperu se práce musí potýkat s tolerancí nepřesnosti a základními NLP problémy (Named Entity Recognition, rozpoznání potenciálních užitečních informací – datum,čas, nadpisy… analýza vět, odstavců…). Očekává se, že jazyk pro definici dat nebude sloužit pouze k popisu jednoho konkrétního dokumentu, ale pro celou sadu podobných dokumentů. Jazyk musí na jednu stranu umět popsat elementy dost obecně – aby se ty jednoduše definovatelné daly využít ve spoustě podobných dokumentů. Ale zároveň jiné elementy popsat konkrétně – zejména takové, které budou mít složitou specifickou strukturu. V semestrální práci bude představen návrh řešení problému, který vychází z analýzy a prototypování. Je však téměř jisté, že po implementaci a testování nebude zcela vyhovovat a bude poupraven. Návrh by proto měl být škálovatelný.

Je pochopitelné, že některé informace půjdou popsat snadno, jiné nepůjdou téměř vůbec. Jedná se o dokumenty, které se nedrží žádné struktury. Extrahovat informace čistě z volného textu by znamenalo mít k dispozici velké množství označkovaných dat, což nemáme.

Dalším problémem, který je třeba řešit, je náročnost na uživatele. Je sice krásné vytvořit složitý nástroj, který umožní rozpoznávat s velikou přesností, ale uživatelé ho nebudou umět použít. Protože nebude dostatek označkovaných dokumentů (tj. trénovacích dat), bude se klasifikační model muset vytvářet s asistencí uživatele. Komunikace s uživatelem bude důležitá i ve fázi rozpoznávání. Systém nemusí mít příliš velkou přesnost a proto musí hlásit nejasnosti uživateli, který může zkontrolovat výsledky, upravit klasifikační model, nebo nepřesnosti upravit jednorázově ručně. Kontrola dat je důležitá, protože často si nemůžeme dovolit nechat systém automaticky zpracovat dokumenty a pak se spoléhat na správnost dat. To sice znamená pro uživatele práci navíc, ale i tak to bude rychlejší, než informace přepisovat ručně.

V následující kapitole proberu obecně obecná témata, jako klasifikaci textu, Named Entity Recognition, wrappery a analýzu existujících podobných technologií – např. Lixto. Další kapitoly se budou věnovat rozborem konkrétní problematiky a návrhem řešení.

3 Extrakce informací z dokumentů

IE je obor zabývající se automatickou extrakcí informací z nestrukturovaných, nebo polostrukturovaných dokumentů. Jak již bylo zmíněno v úvodu, extrahovat data je možné i z webových stránek. Na to jsou určeny wrappery, které se přesně řídí strukturou dokumentů a pravidly. Pokud chceme ale extrahovat data z méně strukturovaných dokumentů, musíme mít k dispozici pokročilejší nástoroje, které umožní alespoň přibližné pochopení sémantiky textu. IE také čerpá zkušenosti z oblasti Přirozeného zpracování jazyka.

Zpočátku se IE zaměřovala na identifikace jmenných entit – obecných, jako jsou jména lidí, měst, organizací, datumy, časy apod a jejich základní relace. Pokud bylo potřeba nějaké lepší porozumnění textu, dosahovalo se celkem slušných výsledků, ale jen v úzkých oblastech. Některé aplikace nyní uvedeme. Rozdělení systémů v této kapitole je převzaté z [1].

3.1 Aplikace IE

Sledování zpráv
Tato část je zaměřena na automatické vyhledávání specifických událostí z novinových článků. Může se jednat o různé nehody, epidemie nemocí, či teroristických událostí. Základem těchto extrakcí je nalézt pojmenované entity a pomocí NLP postopů (například identifikace podstatných jmen, přídavných jmen a sloves) zjistit jaké jsou relace mezi nima.

Obchodní aplikace
Zde se jedná zjišťování informací o produktech, ceně nebo zákaznících z emailů, či jiných dokumentů.

Čištění dat
Oblast IE se dá dobře využít i v databázích. Například máme k dispozici sloupec adresa a naším úkolem je přetransformovat ho na strukturované informace o zemi, městu, ulici, číslu popisném apod.

Vědecké aplikace
Zde se jedná především o oblast bioinformatiky.

Sledování názorů, či nelegálního obsahu
Sledování názorů bere v úvahu, zda je názor spíše negativní, či pozitivní. Vyhledávání nelegálního obsahu může vyhledávat určité věci jako rasistické texty apod – zde však již může jít o nástroj na potlačování svobody slova.

3.2 Typy dokumentů

Strojově generované dokumenty

Zde se může jednat zejména o webové stránky, ale i o dokumenty v např. MS Word. Tyto nástroje se nazývají Wrappery. U těchto dokumentů obvykle nebývá potřeba porozumět částem textu. Důležitá je struktura.

Částečně strukturované specifické dokumenty

Toto je případ, kterým se zabývá diplomová práce. Jedná se o dokumenty, které mají danou určitou strukturu, ale nedá se na ní za každou cenu spoléhat, jako u strojově generovaných dokumentů. Musíme se vypořádat s rozmanitostí, chybějícími daty, různým pořadím elementů apod. Tyto typy dokumentů je třeba popsat částečně strukturálně – identifikovat věty, odstavce, získat informace o tabulkách, seznamech, nadpisech… A zároveň je částečně popsat sémanticky – identifikovat pojmenované entity, datumy atd.

Otevřený text

Otevřený text není strukturován téměř vůbec - s výjimkou rozdělení do vět a odstavců. Kromě identifikace jmenných entit je třeba používat pokročilé nástroje jako zjišťovat informace o slovních druzích apod. Extrahovat informace z otevřeného textu je nejobtížnější.

3.3 Přehled metod pro extrakci

Ručně definované nebo učící metody

Ručně definované systémy vyžadují odborníka, který vytvoří pravidla tak, aby dávala co nejlepší výsledky. To vůbec nemusí být snadné. Je potřeba totiž znát dobře doménovou oblast a zároveň problematiku extrakce. Je třeba vyvinout robustní extrakční pravidla, která budou dobře generalizovat (tj. tolerovat variabilitu). Na druhou stranu, učící systémy se učí automaticky z velkého množsví označkovaných dat. Přestože ručně definovaná pravidla jsou vývojově starší, neznamená to, že by byla automaticky horší. Silně totiž záleží na problému a možnostech.

Zatímco ruční pravidla jsou náročná na vytvoření, nepotřebujeme na ně velké množství označkovaných dat. Učící systémy se totiž potýkají také s množstvím problémů. Zejména se jedná o problém generalizace, kdy se systém naučí nějaká pravidla z trénovacích dat, ale tato pravidla potom nefungují na ostatních dokumentech. Nebo, v tom horším případě, se systém není schopen problém naučit vůbec. Příčin může být mnoho:

Špatně zvolený učící algoritmus
Používáme učící algoritmus (či model klasifikátoru), který se absolutně nehodí na daný problém.

Trénovací data nejsou variabilní
Máme sice velké množství dat, ale mezi nima se nachází velké množství dat, které se různě opakují. Tím pádem učících dat nemáme mnoho. Může se pak stát, že se potom systém naučí tato pravidla velmi silně a odmítá jakoukoliv jinou možnost. U trénovacích dat je důležité aby obsahovaly různé možnosti. Většina trénovacích modelů je schopná dobře generalizovat, ale jen pokud má k dispozici dobré trénovací vzory, které pokrývají různé okrajové možnosti – tímto se zabývá například SVM klasifikátor.

Model pracuje dobře na kvalitních trénovacích datech, ale špatně na těch ostatních
Tomuto problému se říká přeučení. Vzniká například v okamžiku, kdy řešíme jednoduchý problém složitým modelem. Model je tak přesný, že dokáže přesně popsat trénovací data a nepřipustí žádnou jinou alternativu. To vzniká například u neuronových sítí. Jednoduché modely tuto vlastnost obvykle nemívají, protože nejsou schopny trénovací data popsat tak přesně. Z toho plyne, že velká přesnost u trénovacíh dat nemusí znamenat dobrý výsledek. Toto se řeší například prokládáním trénovacích a testovacích dat, ale tímto se v semestrální práci zabývat nebudeme.

Takže volba mezi ručními pravidly a učícím algoritmem není vždy jasná. Obecně se dá říci, že máme-li dostatek trénovacích dat a problém je těžké jednoduše popsat, je lepší použít učící algoritmus.

Pravidlové nebo statistické metody

Pravidlové metody pracují s přesnými pravidly. Zatímco statistické metody pracují s čísly (například váhy, práh apod.). Pravidlové metody jsou jednodužší na pochopení a vytvoření. Umožňují lepší kontrolu dat a správnost výsledků. Na druhou stranu netolerují žádné malé změny a nepřenosti.

Pravidlové metody jsou dobré v oblastech, kde si nemůžeme dovolit chybné výsledky a problém lze dobře popsat. Zatímco statistické metody jsou vhodné tam, kde se dá nepřesnost tolerovat a problém je příliš složitý na to, aby šel popsat přesnými pravidly. Další výhoda je vysvětlení dosaženého výsledku. Statistické metody nám většinou nic neřeknou o tom, proč systém dospěl k určitému rozhodnutí, z pravidel se to dá ale často vyčíst.

Pravidlové metody se například využívají u extrakce z počítačem generovaných dokumentů. Vzhledem k tomu, že počítač používá na generování sám pravidla, není obtížné tyto dokumenty takto

popsat. Zatímco statistický přístup je vhodný na popis dat, které pochází z přirozeného prostředí – například od člověka.

Existují kombinace mezi ručně definovanými nebo učícími metodami a mezi pravidlovými a statistickými metodami.

Ruční pravidlové
Jedná se o problémy, které lze dobře popsat ručně a přesně. S těmito pravidly se setkáváme vlastně docela často. Může se jednat například o klasifikaci čísel do 2 tříd – sudá a lichá. V Oblasti IE se může jednat o identifikaci něčeho, co se dá popsat třeba regulárním výrazem – názvy, které se drží přesných standardů apod. Ruční pravidlové jsou častější než učící pravidlové.

Učící pravidlové
Učící pravidlové metody se používají u problémů, které mají nějaká vnitřní pravidla, ale jsou poměrně složitá na vytvoření a zároveň máme k dispozici velké množstvé trénovacích dat. Učící algoritmy pro pravidlové systémy jsou například Bottom-up, nebo Top-down metody. Dají se využít i jiné metody, jako je genetické programování.

Učící statistické
Statistické metody bývají nejčastěji učící. Aplikují se na rozsáhlá trénovací data a jsou tolerantní vůči chybám a nepřesnostem. Mezi klasické metody patří Bayesovké klasifikace, Markovy modely, Neuronové sítě, SVM apod.

Ruční statistické
Podobají se ručním pravidlovým. Opět je vytváří uživatel, ale tentokrát se nejedná o pevná pravidla, ale o model tolerující nepřesnost.

Model, který využívám v diplomové práci je hybridní. Je to vlastně pravidlový přístup, kde pravidla nejsou pevná, ale mají statistický charakter. Jsou ruční, ale zároveň částečně učící.

3.4 Pravidlové systémy

Pravidlové systémy se obvykle skládají z těchto pravidel: *Nalezený vzor -> Akce*. Nalezený vzor (či entita) se skládá z takzvaných vlastností, nebo rysů (ang. Features). Rysy mohou obsahovat informace o výskytu určitých rysů, nebo dříve nalezených vzorů. Je tedy běžné, že vzory se hledají v **kaskádách**. Postupuje se obvykle od obecnějších a snadno identifikovatelných vzorů, až k těm specifickým.

Například se může jednat o nalezení ulice, města a čísla popisného a následně vytvořit další entitu s názvem Adresa, která nalezené vzory obaluje.

Rysy entit:

- Řetězce obsažené ve vzoru.
- Pravopisné rysy, jako například velká písmena. Toto je dobré vodítko pro názvy.
- Slovní druhy – podstatná jména jsou například dobrým kandidátem pro jmenné entity, slovesa jsou zase důležitá pro odvozování relací.
- Externí znalosti ze slovníků.
- Jiné, dříve nalezené vzory – zde je třeba se zmínit o pojmu Bag Of Words, který se v IE používá často. Slova, která se nachází okolo zkoumaného prvku nám dávají velice silnou informaci o jeho sémantice. Z důvodu zjednodušení se často nesnažíme najít mezi slovy gramatické závislosti a proto nás zajímá jen jejich výskyt. Bylo zjišťěno, že toto zjednodušení bývá na spousty problémů dostatečné.
- Vizuální prvky. Například velikost písma vůči okolnímu textu, informace o tabulce, seznamu, odstavcích… - toto se dá využít ale pouze v částečně strukturovaných dokumentech.

Pravidla pro rozpoznání entity

Jsou zde důležité tři parametry:

- Pravidla pro začátek entity
- Pravidla pro obsah entity
- Pravidla pro ukončení entity

Pravidla mohou být různá. Může se jednat o prosté textové vzory, o dříve nalezené entity či obecné informace typu – entita začíná v dalším odstavci, za již nalezenou entitou, nebo v řádku pod, buňky v tabulce napravo atd. Vnitřní pravidla entit se obvykle skládají z dříve nalezených entit nebo textu. Konec entit je podobný jako začátek, avšak často je ho velmi těžké definovat. Může být totiž omezen výskytem jiné entity, koncem odstavce apod.

Jiná pravidla mohou být tvořená regulárním výrazem, bezkontextovou gramatikou, či dokonce běžným programem. Programovací jazyk má největší vyjadřovací schopnost a často se používá pro nalezení doménově specifických entit, které se těžko definují pomocí běžných pravidel. Spousta nástrojů je proto otevřená a umožňuje přiložit vlastní kódy pro rozpoznávání. Volání tohoto kódu může být implementováno jako volání události pro pravidla – tzn. Pravidlo má tvar Nalezený vzor -> Akce a kód je možné napojit na obě strany pravidel. Vzhledem k tomu, že jsou těmto funkcím předány i informace o stavu rozpoznávání, jedná se o velmi silný nástroj.

Překrývání entit a hierarchie

Je běžné, že entity se mohou překrývat. Jsou ale dva případy překrývání:

Entita je vytvořena na základě nalezených entit

To je případ dříve uvedené adresy. Nebo se může jednat o jméno:

<PERSON><title>Dr.</title> <first name>Sheldon</first name> <last name>Cooper</last name>, <title>Ph.D.</title><PERSON>

Je jasné, že entity Person a ostatní nejsou v konfliktu. Pochopitelně by bylo možné při přiřazení entity odebrat, ale mohli bychom ztratit informace, které využívají nějaké jiné entity. Např:

<Meeting Date><Date>21.2.2009</Date><Meeting Date>

Mohli bychom sice odebrat entitu Date, ale jiná entita by se na tuto entitu mohla odkazovat a to by nebylo dobré. Pochopitelně by se mohla odkazovat na přesnější zařazení Meeting Date, než na obecnou date. Ale to by způsobovalo silně rekurzivní pravidla, kterým se chceme vyhnout, protože Meeting date by se zase mohlo odkazovat na původní pravidlo. Proto je lepší když se entity odkazují na entity, které byly vytvořeny v předchozím kroku. Pokud se definuje určitá kaskáda entit, potom bychom tyto problémy neměli. Obecnější pravidla by se aplikovala dříve a konkrétnější později. Je tedy potřeba definovat určitou hierarchii a říci, které entity mohou obsahovat jiné entity bez omezení a v jakém pořadí je aplikovat. Ty ostatní se nesmí překrývat.

Entita se překrývá a neměla by:

Druhý případ je složitější:

<PERSON>Dr. Sheldon<PERSON>Cooper</PERSON>Ph.D.</PERSON>

Systém nám vytvoří dvě entity místo jedné. Nebo jiný případ:

*<**RECORD**> <Date>21.2</Date> <PERSON> John Connor </PERSON> <TASK> <**RECORD**> Prepare next meeting </TASK> </**RECORD**>*
*<Date>21.2</Date><PERSON>Sarah Connor<PERSON></**RECORD**>*

Zde byly údaje Datum, Osoba, úkol a u druhého záznamu úkol chyběl. Systém byl původně naučen na šablonu, úkol, datum, osoba. Protože mu úkol u druhého záznamu chyběl, vydedukoval, že úkol za jménem patří Johnu Connorovi. Druhý záznam zase předpokládal, že na prvním místě se nachází úkol. Tento problém by pochopitelně byl těžko řešitelný i pro člověka, který by neměl velké znalosti problému a neměl by k dispozici vizuální informace – jakože je každý záznam na samostatném řádku atd. Mohou ale nastat horší problémy. Často se stane, že se navážou různé entity na překrývající se oblasti. Potom je problém vyřešit, která vlastně platí.

Toto jsou věci, které se nedají jednoduše obecně popsat. Velmi záleží na doméně, ve které se pohybujeme. Existuje ale několik doporučených postupů:

Entity jsou neseřazené a disjunktní

Populární postup, kdy se entity vytvářejí nezávisle na sobě a teprve tehdy, až dojde ke konfliktu (překrytí), spustí se následující heuristiky:

- Preferuj entity, které jsou delší
- Sluč tyto entity do jedné – to lze udělat pouze v případě, že akční část tohoto pravidla je stejná. Pokud není, je třeba využít jiných složitějších specifických postupů.

Tato metoda je jednoduchá, protože umožňuje definovat pravidla nezávisle na sobě. Ale u složitějších problémů selhává.

Pravidla mají definované seřazení

Tento problém byl načnut už v části o hierarchii entit. V tomto případě je to rozšířeno dál – na priority. Každá entita má definovanou prioritu a pokud se vyskytne konflikt, vyhraje ho entita s vyšší prioritou. Prioritu je třeba volit podle určitých kritérií. Entity, které mají složitější pravidla a vysokou přesnost vyhledávání by měly mít vyšší prioritu, než entity, které mají pravidla jednoduchá a mohou se vyskytovat často a s chybama. Priorita může také zohledňovat délku entity, nebo u hybridních systémů i ohodnocení pravidel – statistické metody totiž nepracují na systému Ano/Ne, ale používají ohodnocení – o tom ale v další části.

Učící algoritmy

Pravidlové systémy bývají nejčastěji ruční, přesto existují postupy, jak je automaticky naučit. V diplomové práci učení pravidel nejspíš nebude využito vůbec a proto se tím nebudeme zabývat.

3.5 Statistické systémy

Tyto systémy jsou určeny zejména pro učící algoritmy. Místo pevných pravidel jsou modely založeny na statistice, neuronových sítích apod. To je mnohem vhodnější na učení. Podobně jako v pravidlových systémech, i zde se pracuje s jistými entitami. Text může být kupříkladu rozdělen na tzv. tokeny, což můžou být třeba slova. Každý token má jisté vlastnosti – slovní druh nebo typ (zda se jedná o jméno, místo…). Učící systém potom pomocí trénovacích dat zjišťuje vzájemné závislosti mezi tokeny a vytváří tak model. V zásadě se tento postup příliš neliší od pravidlových systémů. Je možné pohlížet na model jako na sadu pravidel, ale tato pravidla jsou tolerantnější a snáze se na ně dají aplikovat učící algoritmy.

Model, který se bude používat v diplomové práci, bude model hybridní. Bude se jednat o pravidlové systémy s měkkými pravidly reprezentovanými neuronovou sítí, která se bude konstruovat automaticky pomocí uživatele a bude možné jí pak místy upravovat na trénovacích datech. Vzhledem k tomu, že dat bude málo, není možné vytvořit model, který by se natrénoval kompletně z trénovacích dat bez zásahu uživatele.

Neuronové sítě

Neuronové sítě jsou velmi robustní nástoje, které mají spousty využití v různých oblastech. Inspirují se totiž skutečnými neurony v mozku, ale běžné modely mají k nim mají daleko. V oblasti IE se příliš nepoužívají a to z důvodu malého množství trénovacích dat. Velmi snadno se totiž dokáží přetrénovat – tj. vytvářet modely, které sice přesně vystihují trénovací data, ale neumí pracovat na testovacích datech v provozu, které se liší. Další nevýhoda je složitě zakódované řešení – ze změti spojů mezi neurony se toho příliš mnoho vyčíst nedá a některé systémy vyžadují vysvětlení. Ale největším problémem je vysoká dimenzionalita a řídkost rysů – text obsahuje hromady možných rysů a každý token přitom má velmi malou část z nich. Pokud bychom vykonstruovali neuronovou síť a začli jí učit, byla by ohromná a přitom by byla využita jen malá část na malých trénovacích datech. Odborník zběhlý v neuronových sítích snadno vydedukuje, že takový model by silně trpěl přeučením a neefektivitou. Proč byl tedy v diplomové práci zvolen model inspirující se neuronovou sítí? Vše bude vysvětleno po úvodu do jejich problematiky.

Klasifikace

Než budou popsány neuronové sítě, je třeba si vysvětlit pojem klasifikace, který s tím souvisí. Klasifikace je hledání transformační funkce mezi vstupy a výstupy. Vstupy jsou nějaká data a výstupy znamenají třídu, do které patří. Vstupem může být tedy například obrázek nějakého čísla a výstupem třída, do které patří – tj. 0, 1, 2, 3…9.

Klasifikace je při extrakci informací běžnou činností. Přiřazení elementu určité části textu je v zásadě klasifikační problém. Máme text a na jeho základě přiřazujeme sémantiku každému slovu. Sémantika se dá pojmout jako klasifikační třída, které může klasifikátor nabývat. Statistické i pravidlové systémy jsou tedy klasifikátory. Oba modely mají nějaké vstupy a na jejich základě počítají výstup.

Klasifikace může mít mnoho podob. Například se může rozlišovat klasifikace do **jedné z N tříd** a **do více tříd**. V prvním případě klasifikátor přiřadí každému vstupu právě jednu třídu a v druhém případě je vstupu přiřazeno více tříd – navíc s informacemi o míře příslušnosti k dané třídě. Pravidlové systémy obykle podporují jen první variantu, kdežto u statistických není problém podporovat obě.

Dalším pojmem je **binární klasifikace**. Ta umožňuje rozlišovat pouze dvě třídy – obvykle se používá na rozlišení – ano/ne – vzor patří do třídy / nepatří. Pokud chceme klasifikovat do více tříd, není problém mít paralelně několik binárních klasifikátorů, které se vzájemně ani nemusí ovlivňovat.

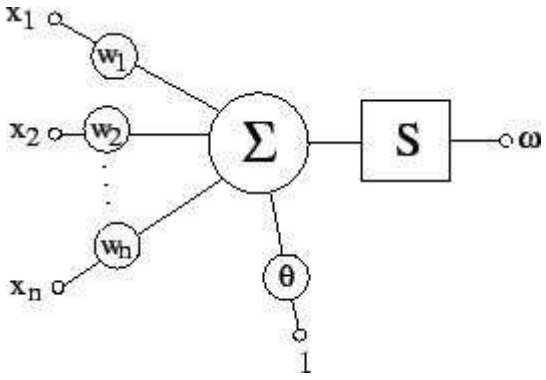
Lineární klasifikátor je takový klasifikátor, který umožňuje klasifikovat do dvou tříd a navíc lineárně separabilní data – tzn. data, která lze rozdělit na dvě části dělící rovinou – tj. ve 2D přímkou, ve 3D rovinou… Například fukce AND je lineárně separabilní, pokud bychom si jí promítli na osu x a y, měli bychom bod 1 při x = 1 a y = 1, v ostatních částech 0, takže by šlo snadno zkonstruovat

přímku, která rozdělí prostor na části, kde funkce AND dává 1 a kde 0. Pokud bychom si vzali funkci XOR, u ní to možné není, protože dává výstup 1 na 00 i na 11. Díky tomuto poznatku se mimojiné zastavil v 70. letech výzkum neuronových sítí, protože perceptrony (budou popsány dále) jsou lineární klasifikátory. Pak bylo ale konstatováno, že nelineární klasifikátor lze získat vhodnou kombinací klasifikátorů lineárních, což je zřejmé.

Komprese. Přestože komprese patří do zcela jiné oblasti, je třeba si uvědomit, čím vlastně klasifikátor je. Nejlepší klasifikátor by byl takový, který by měl obrovskou tabulku dat a při výběru třídy vstupu by ho porovnal s databází, vybral nejpodobnější prvek a přiřadil jeho třídu na výstup (tento postup se také někdy používá, např. u K-Nearist alogirtmu). Klasifikátor provádí kompresi tím, že nachází mezi prvky vzájemné závislosti, kóduje je do modelu a ten pak využívá. Model je samozřejmně menší než by byla původní databáze trénovacích dat. Tato komprese však bývá nepřesná, proto se dá používat jen u určitých dat – jako jsou třeba multimedia.

Perceptron

Perceptron vymyslel v roce 1957 Frank Rosenblatt. Inspiruje se biologickým neuronem, který má hromady vstupů, tělo a jeden výstup. Vstupy symbolizují vstupní informace, které se v těle spojí a rozhodne se, zda se má výstup zaktivovat. Vstupní informace může signalizovat nějakou vstupní veličinu, nebo může být výstupem jiného neuronu. Pokud jsou vstupní informace dostatečně silné (přesáhnou vnitřní práh neuronu), neuron se zaktivuje a vyšle signál.



Obrázek 3.1

Každý vstup má svojí důležitost. Ta hraje při rozhodování důležitou roli. Uveďme jeden příklad z praxe (při značném zjednodušení).

Mějme neuron, který určuje zda se nám bude líbit osoba opačného pohlaví.
Vstup – oči, důležitost: 5
Vstup – Tvář, důležitost: 3
Vstup – Společné zájmy, důležitost: 4
Práh: 6

Potkáme-li tedy osobu, která bude splňovat oči na 100%, tvář na 50% a společné zájmy na 80%, potom vnitřní hodnota neuronu bude 5 * 1 + 3 * 0.5 + 4 * 0.8 = 9.5. Práh je 6, neuron tedy vyšle signál a vyhodnotí, že se nám daná osoba líbí.

Perceptron je tedy definován počtem vstupů, jejich příslušnými vahami (důležitostmi) a prahem. Můžeme to matematicky popsat vzorcem: (následující vzorce jsou převzaté z wikipedie):

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{else} \end{cases}$$

Perceptron je tedy binární klasifikátor, který klasifikuje vstupní data do dvou skupin – 1 či 0.

Učení perceptronu

Je jednoduché. Perceptronu jsou postupně předkládána trénovací data ve formě – vstup a požadovaný výstup. Perceptron je na začátku naplněn náhodnými vahami (případně může být nadefinován uživatelem, jako v případě diplomové práce).

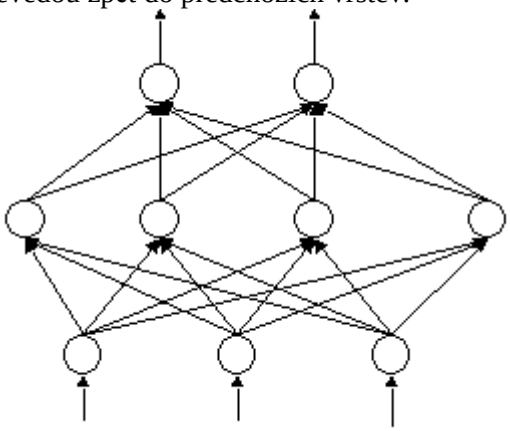
Pokud dostane na vstup nějaký trénovací vzor, vyhodnotí ho a porovná svůj výstup s požadovaným výstupem. Je-li tento výstup správně, nedělá nic. Ale pokud je tento výstup špatně, potom poupraví váhy:

$$w_{t+1}(j) = w_t(j) + \alpha(y - f(x))x(j) \quad (j = 1, \ldots, n)$$

Nalevo je nová hodnota váhy, která se získá jako stará hodnota váhy plus přírustek. Tento přírustek je tím větší, čím je rozdíl y – f(x), což znamená rozdíl ve výstupu mezi skutečnou hodnotou a požadovanou hodnotou. Pokud bude skutečná hodnota větší, než požadovaná, závorka vrátí zápornou hodnotu a váha perceptronu se bude snižovat, pokud je skutečná menší, váha se bude zvyšovat a pokud jsou stejné, výsledek bude 0, tudíž nová váha bude rovna staré váze. Celé se to ještě násobí vstupem (protože čím větší je vstup, tím více by se mělo „pohnout“ s váhou) a koeficientem učení, který bývá kolem 0.1 a snaží se brzdit rychlost učení, kdyby byl 1, tak by se perceptron učil příliš velkými skoky a měl by problém dosáhnout optimální hodnoty. Na druhou stranu malá hodnota se učí pomaleji a proto se někdy zpočátku určuje hodnota učení vyšší a postupně se snižuje. Toto pravidlo pro učení jednoho perceptronu bude v zásadě použito i u neuronových sítí a je tedy důležité.

Vícevrstvé sítě

Perceptron je lineární klasifikátor. Nedělá nic jiného, než že protne prostor přímkou (rovinou) a rozděluje prostor do dvou tříd. To na spousty problémů nestačí a proto byly vyvinuty vícevrstvé sítě. Ty umožňují vzájemné propojení neuronů a tím tak změnit lineární klasifikátor na nelineární. Nejjednodužší je vícevrstvá dopředná síť, která neumožňuje zpětné propojení neuronů, které by způsobily rekurzi. Máme zde vstupní vrstvu neuronů, které vedou do další vrsty a ty opět do další, ale nikdy nevedou zpět do předchozích vrstev.



Obrázek 3.2 – neuronová síť

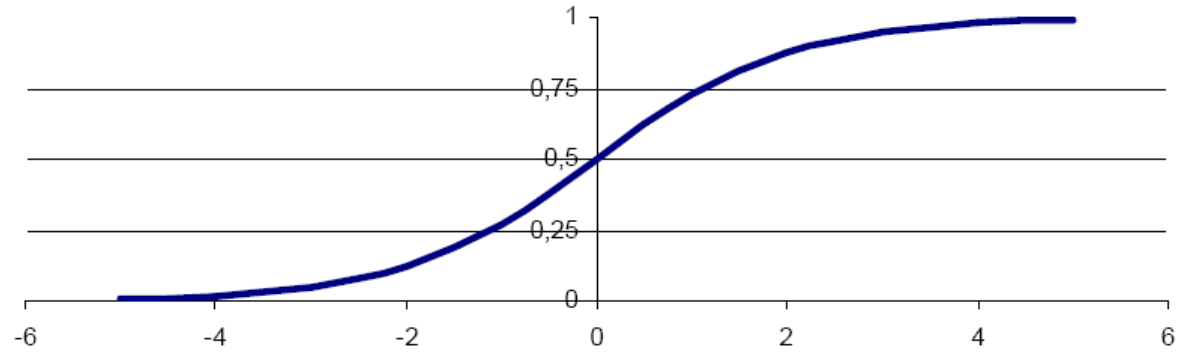
Tato síť se dá dobře naučit algoritmem Backpropagation. Je velmi známý a populární, vzhledem k tomu, že vyžaduje velké množství trénovacích dat, pravděpodobně nebude v diplomové práci využit. Bude proto uvedena jen jeho stručná myšlenka:

Backpropagation

Máme učící algoritmus pro jeden neuron, který potřebuje znát svůj požadovaný výstup. To je však možné jen u neuronů ve výstupní vrstvě. Známe tedy chybu neuronů ve vstupní vrstvě, které si mohou poupravit své váhy. To ale nestačí, lepší by bylo kdyby řekly neuronům v nižší vrstvě, že je něco špatně a ať si je poupraví taky. To je základní myšlenka algoritmu. Algoritmus postupuje po vrstvách, od poslední výstupní do spodnějších. Každý neuron si potom spočítá svojí chybu na výstupu a rozešle je neuronům vespod. Tuto chybu znásobí vahama, které je spojují – tím dává najevo, že čím silnější propojení mezi neurony je, tím zásadněji se spodní neuron podílí na chybě a tím víc by jí měl eliminovat. Neurony v nižší vrstvě tedy dostanou informaci o chybě od každého neuronu ve vyšší vrstvě, výsledky sečtou a tím dostanou celkovou chybu. Sečtení chyb je důležité, protože se stane, že jeden neuron bude požadovat zvýšení výstupu, jiný zase snížení – například by jeden mohl požadovat zvýšení s chybou 5 a druhý snížení s chybou -3, výsledkem může být (zjednodušeně, vzorec je pro to trochu složitější) zvýšení s chybou 2. Laicky řečen, neuron odpoví: „Ano, zvýším propříště svůj výstup, ale ne tak, jak žádáš, musím brát ohled i na ostatní“.

Přechodová funkce

Aby mohl algoritmus Backpropagation fungovat, je třeba mít jinou výstupní funkci neuronu než skokovou (výstup 0 nebo 1). Je třeba vědět, jak moc se přibližujeme k daným třídám. Proto se často používá funkce sigmoida, protože má vhodný tvar a dobře se derivuje – derivace je potřeba u výpočtu chyby. Obrázek je převzatý ze studijní opory Získávání znalostí z databází.



Obrázek 3.3 – funkce sigmoida

Sigmoida má hladký přirozený průběh a proto bude asi použita i v diplomové práci, přestože není nutně třeba, protože se používá zejména kvůli učení Backpropagation. Model neuronové sítě bude základem pro klasifikační model diplomové práce. Protože to bude síť, která se bude konstruovat ručně, nebude problém s vysokou dimenzialitou rysů (budou využity jen ty důležité) ani s neprůhledností zakódování řešení – každý neuron bude mít svojí sémantiku a vysvětlení. Nejprve bude ale stručně představen systém, který vznikl jako inspirace pro tuto práci – Lixto.

3.6 Lixto

Lixto je kvalitní komerční nástroj. Je určen k extrakci informací z webových dokumentů (i když výrobci uvádí, že se nemusí jednat jen o web). Jeho nespornou výhodou je uživatelská vstřícnost. Pomocí jednoduchých označování požadovaných informací a doplňujících údajů si uživatel nadefinuje model, který se přeloží do extrakčního jazyka Elog, který uživatel vůbec nemusí znát.

Elog se skládá z takzvaných vzorů (patterns), které mají určité filtry – podmínky. Tyto podmínky mohou být například:

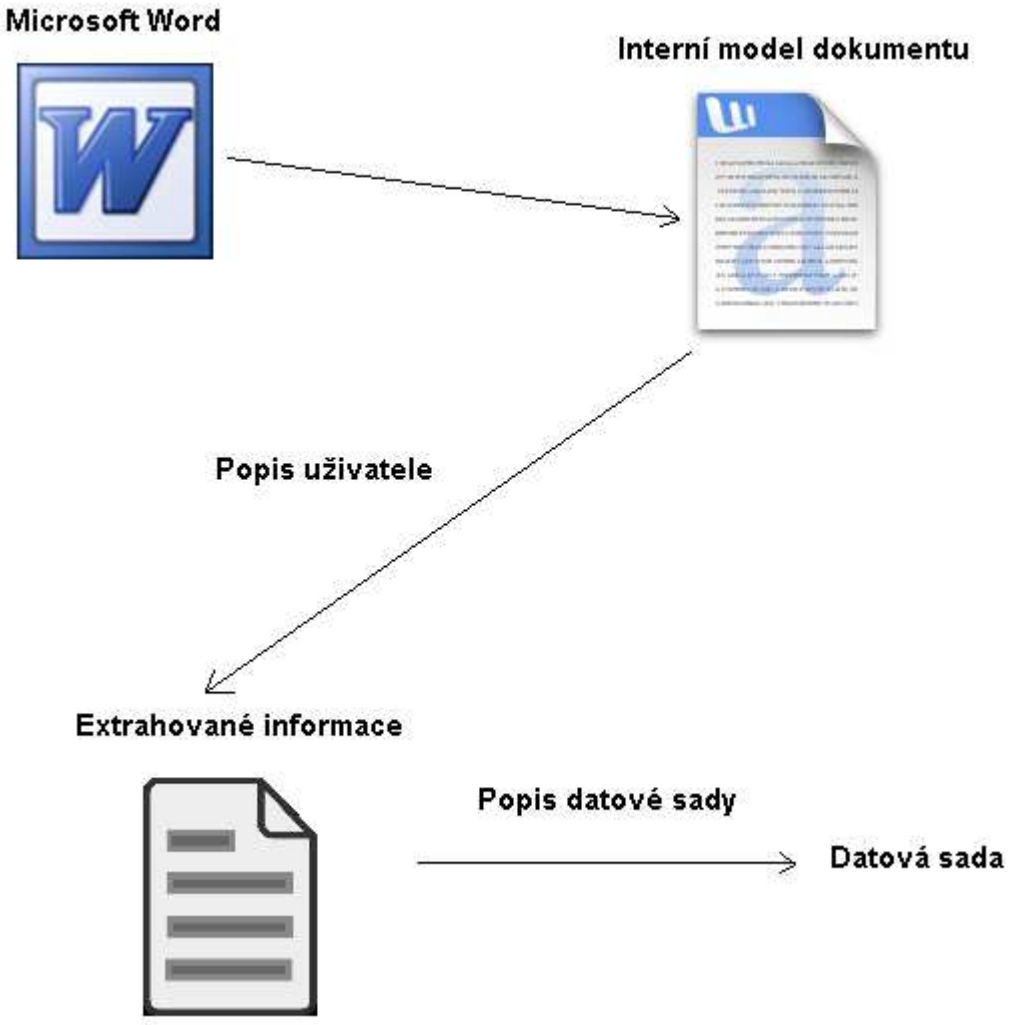
After / Before	Výskyt elementu před/za elementem
Not after / Not Before	Nepřítomnost elementu
Elements inside	Podmínky pro elementy uvnitř elementu
Regulární výraz	Pokud regulární výraz uspěje, nalezený element je označen
Rozsah	Výběr jen určitých částí elementu – první část, poslední, druhá atd.

Tento jazyk je plně skryt před uživatelem. Uživatel pouze označuje požadované části dokumentu a podmínky jsou automaticky vyhledávány z kontextu. Začíná se těma nejjednodužšíma a upravují se a přidávají dokud nejsou označeny všechny požadované elementy a žádné jiné. Vzhledem k tomu, že Lixto je komerční nástroj, který se nepodařilo sehnat, omezíme se v diplomové práci pouze na tyto základní informace a pokusíme se navrhnout systém podobný Lixto. Lišit se bude především v tom, že nebude určen pro web, ale pro polostrukturované dokumenty psané lidmi, takže musí být tolerantní vůči nepřesnostem a za druhé, bude optimalizovaný na doménu pro extrakci z dokumentů záznamů ze schůzek.

4 Popis systému

Návrh systému byl převzat ze semestrální práce a rozšířen. Základní myšlenka však zůstala stejná.

4.1 Schéma systému pro extrakci informací z dokumentů



Obrázek 4.1 – schéma systému

Extrakce se skládá ze 3 fází:

1. Systém automaticky načte data z programu Microsoft Word pomocí COM objektů a uloží si je do interního modelu. Tento krok je nezbytný, protože komunikace přes COM je pomalá a na druhé straně to umožňuje rozšířit systém i o jiné formáty, než jen Microsoft Word.

2. Uživatel ručně popíše pomocí uživatelského rozhraní elementy, které chce vyextrahovat. Alternativně lze elementy popsat i ve formátu XML, který je však příliš upovídaný na to, aby sloužil přímo pro uživatele. Je to spíše jakýsi komunikační formát pro případ integrace externích nástrojů a především slouží k serializaci pravidel pro účely persistence.

3. Uživatel popíše jak chce převést vyextrahované dokumenty do dat – tj. které elementy mají tvořit řádky tabulky, které sloupcové údaje apod. Výsledná datová sada se dá opět převést do XML, nebo RDF či naplnit do databáze – RDF a databázi ale systém implicitně nepodporuje, na to je třeba vytvořit vlastní přídavný modul.

4.1.1 Převod z formátu Microsoft Word

Převod dat z formátu Microsoft Word do interního modelu je úzké hrdlo aplikace, protože omezuje možnosti systému. Vzhledem k limitům komunikačního rozhraní COM se musela extrakce přizpůsobit například faktu, že se nepodařilo najít způsob, jak získat některé vizuální parametry textu. Toto omezení bylo v některých případech tak silné, že se určité problémy nepodařilo pomocí stávajících postupů vyřešit. Dala by se najít i jiná řešení, ale byla by příliš komplikovaná, nepřesná a pro uživatele nesrozumitelná. Tyto problémy budou rozebrány v kapitole… (TODO).
Dalším problémem je rychlost komunikace. Načíst základní informace - tj. text, tabulky apod., netrvá dlouho. Ale načítání informací o fontu je podstatně pomalejší. Pravidla, která tyto informace vyžadují často, velkým způsobem zvyšují časovou náročnost extrakce.
Nedokonalá komunikace COM se podepsala i na uživatelském rozhraní. Například není možnost programově označit nesouvislé části dokumentu. A alternativa, vybarvování nalezených elementů ,není zcela komfortní. Trvá dlouho, než se označení provede a uživatel si takto může omylem změněný dokument uložit a poškodit. Komunikace s programem MS Word byla převzata z [5].

4.1.2 Popis elementů

Zde přichází na řadu uživatel. Jeho úkolem je popsat, jakým způsobem jsou v dokumentech zapsány informace, které chce extrahovat. Úkol to není zcela jednoduchý a velmi záleží na složitosti, strukturovanosti a podobnosti zdrojových dokumentů. Úspěšnost extrakce pak nezáleží jen na systému samotném, ale zejména na vytvořených pravidlech od uživatele.

4.1.3 Převod do dat

Po skončení druhé fáze jsou v dokumentu nalezeny a ohodnoceny požadované informace (elementy). Tyto samotné informace ještě nestačí k tomu, aby byl výsledek rozumně interpretovatelný. Převod do dat tedy vytváří pravidla, která by se dala stručně shrnout tímto příkladem:

Pro každý nalezený element ItemRecord (položka agendy ve tvaru datum, čas…) se podívej na položku ItemTime, která označuje čas, dále vyber seznam osob a ulož je do kolekce v rámci struktury ItemRecord. Poslední informaci, datum položky, nalezneš buď uvnitř záznamu, nebo se podívej na nejbližší platný element datumu schůzky vlevo (ve směru toku textu). Výsledné datové položky ItemRecord ukládej do kolekce AgendaItems.

4.2 Tvorba interního modelu dokumentu

Interní model dokumentu slouží k uložení potřebných informací získaných z původního dokumentu MS Word. Má i další důležitý úkol – zoptimalizovat přístup k textu. Textová data jsou příliš rozsáhlá, než aby se v nich dalo vyhledávat metodou slovo po slovu. Dále model dělí text na logické celky – vzhledem k tomu, že vzdálenost se počítá skrze vzdálenost ve slovech, je třeba oddělit konce řádků, konce odstavců atd., několika prázdnými slovy.

4.2.1 Rozdělení na slova

Text se dělí na slova jednak podle pravidel Wordu a jednak podle přídavných pravidel. Postup má svoje výhody i nevýhody.
Hranice slov má následující dělící pravidla:

- Slova jsou oddělena mezerou, či nějakým speciálním znakem (konec odstavce, řádku atd.)
- Pokud slovo obsahuje znaky, pak tyto znaky slovo dělí a samy tvoří další slovo, takže například:

  ABC-DEF je tvořeno slovy: ABC, -, DEF

  Ale text ABC-*-DEF je tvořen ABC, -*-DEF

Tato pravidla dělení slov jsou přímo importovaná z Wordu. Systém přidává další pravidlo:

- Čísla tvoří samostatná slova.
  Tzn. Text 13th bude tvořen 2 slovy – 13, th

Toto je důležité pravidlo, protože systém vyhledává z důvodu výkonnosti pouze celá slova – například při identifikaci datumu je velice často uveden den ve tvaru 13th, což není číslo. Nicméně nikde není řečeno, že by systém nedokázal vyhledávat i části slov, nebo pomocí regulárního výrazu, ale extrakční pravidla se při běžných úkonech bez tohoto obejdou.

4.2.2 Indexace slov

Při vyhledávání slov by bylo časově náročné projíždět celý dokument a testovat každé nalezené slovo na rovnost s požadovaným. Snazší je vytvořit takovou strukturu, která pro každé slovo v dokumentu připojí seznam pozic, na kterých se nachází. Navíc se jedná o hashovací tabulku, takže poměrně rychle

lze zadat dotaz, který vrátí všechny výskyty požadovaného slova. Tabulka výskytu slov navíc nemusí obsahovat všechna slova. Obsahuje pouze ta, která se vyskytnou v extrakčních pravidlech.
Co se týče slov, je třeba udělat malou vsuvku – systém je implementován v prostředí .NET, které používá metodu zadržování řetězců – tzn . všechny řetězce ukládá do Hashovací tabulky a porovnání dvou řetězců je pak porovnání dvou referencí na to samé slovo v Hash tabulce. Tudíž není třeba řešit efektivitu při porovnávání slov, pokud ke slovům přistupujeme jinak, než skrze index.

4.2.3 Rozdělení textu na stavební bloky

Text je v modelu uložen jako pole slov. Vzdálenost dvou elementů se pro efektivitu výpočtu počítá jednoduše jako rozdíl pozic v poli slov. Tento přístup je sice rychlý – například kdyby se jednalo o rozdíl v počtu znaků, nebo skrze nějakou uživatelem definovanou sémantickou blízkost slov – například určitá slova by ji zvyšovala, jiná snižovala, pak by měl sice systém větší popisovací možnosti, ale byl by pomalý a pro uživatele zbytečně složitý. Ale i takto by se dala v nějakém pokročilém pravidlu počítat vzdálenost. Současná pravidla ale používají výhradně vzdálenost ve slovech.
Mějme ale následující text:
Dnes večer jsem šel ven s Janou <konec odstavce>.
Zítra půjdu zase za Markétou.
Které slovo je k elementu Jana blíž? Rozhodně by to mělo být slovo „s“. Z tohoto důvodu jsou speciální oddělovací slova v textu znásobeny 4x (tj. místo jednoho slova /n/a jsou to slova 4). Toto číslo ale lze nastavit. Slovo zítra potom bude mít od slova Jana vzdálenost 5.
Tento způsob má ale i své nevýhody – například pokud jde o slova tvořená kombinací krátkých slov a znaků. Tato slova potom vytváří velkou vzdálenost v textu, větší než by vizuálně měla být.

4.2.4 Tabulky

Tabulky tvoří podstatnou informaci o struktuře dokumentu. Pokud je hodně informací v tabulkách, lze dokument popsat velice přesně. S tabulkami však souvisí jeden problém – sloučené buňky. Tento problém se řešit dá, ale ne ve všech situacích. Komunikační rozhraní s Wordem totiž nedává zcela jednoznačnou informaci o tom, které buňky jsou přesně sloučené – pouze vrátí výjimku, pokud se odkazujeme na adresu, která je součástí jiné sloučené buňky – někdy ale není jednoznačné, zda je buňka sloučená s jinou buňkou ve vedlejším sloupci, nebo v předchozím řádku. Ve spoustě případů je tabulka převedena správně, v jiných případech ne.

I když se převod tabulky nepovede přesně, stále je možná extrakce. Hranice buněk a řádků se určí vždy přesně. Ale některé pokročilejší dotazy nemusí fungovat. Které to jsou, uvedeme nyní. Například:

Tabulka 4.1 - ukázka

10-15h	Kafe
	Přestávka
	Kino

Element Kafe, Přestávka, či kino se chce dotázat na výskyt elementu čas na stejném řádku v tabulce. Element Kafe čas určitě najde. Ale co elementy přestávka a kino? Vzhledem k tomu, že čas se vyskytuje pouze na prvním řádku, tak by ho tam najít neměly. Ale díky nahrazení obsahu sloučených buněk na referenci na buňku s obsahem 10-15h, tam čas najdou také.

Tento případ fungovat bude, ale existují případy, kdy to jednoznačné není. Tyto případy však bohužel nejsou v době psaní zprávy již k dispozici a protože jiné zatím nebyly objeveny a přesnost rozpoznávání se tím příliš nezhorší, nebudeme se tímto problémem dále zabývat.

Díky rozdělení textu do bloků popsaných v předchozí části, má tabulka i pěkné vzdálenostní vlastnosti. A to takové, že vzdálenost dvou buněk je oddělena několika znaky konce buňky a vzdálenost mezi řádky je oddělena dalšími znaky konce řádku tabulky.

4.2.5 Specifické elementy dokumentu

Jazyk pro popis elementů obsahuje poměrně dost nástrojů, některé jsou navíc, vyloženě pro urychlení práce. Ale často je třeba pokládat dotazy typu:

Element se nachází někde okolo zdrojového elementu na stejném řádku.

(jinak – hledej element pobliž v určité toleranci počtu slov, ale nesmíš při tom překročit znak konce řádku)

Element se nachází někde poblíž konce řádku

(jinak – někde za elementem – třeba v toleranci 5 slov, se nachází element konce řádku)

Element nesmí mít rozsah mimo odstavec

(jinak – element nesmí obsahovat element konce odstavce)

Systém totiž až na výjimky umožňuje operace pouze s elementy. Ale díky tomu, že řádky, konce řádků, tabulka, buňka, konec buňky, konec řádku tabulky…atd., mohou být také elementy, získáváme širší škálu dotazů, aniž bychom museli v navrženém systému cokoliv upravovat.

4.3 Základní nástroje pro popis elementů

Protože dokumenty jsou ve formátu programu Microsoft Word, musí se brát ohled na jeho možnosti. Zatím se například nepodařilo zjistit pozici (x, y) určité části textu v dokumentu. Tato pozice by mohla být důležitou vizuální informací.

Přesto můžeme získat více informací, než by bylo k dispozici v otevřeném textu:

- Word umí rozdělit text na slova a věty. Dělení na slova je celkem kvalitní, dělení na věty se však řídí poměrně jednoduchými pravidly a nefunguje přesně, na věty se tedy příliš spoléhat nedá.
- Text je rozdělen na odstavce. Odstavce jsou přínosnou informací, přesto je zde jedna nevýhoda. Začátečníci často nevědí, že stisk klávesy enter, nezpůsobí ukončení řádku, ale ukončení odstavce. Na ukončení řádku je klávesa shift+enter. Z těchto důvodů je třeba informaci o odstavcích brát s rezervou.
- Spousta strukturovaných informací má tvar tabulek. To je velmi důležitý fakt, některé složitě strukturované informace nebude možné zjistit jinak, než z tabulek (či seznamů). U tabulek může být zásadní informace o nadpisu pro sloupce, či řádky.
- Seznamy nám rovněž mohou sloužit užitečně jako tabulky.
- Informace o velikosti, tvaru písma (tučné, kurzíva) apod, nám bude dávat užitečnou informaci o nadpisech, či o důležitých částech dokumentu.
- Informace o pozici vzhledem k nadřazenému bloku (blokem se myslí celý dokument, nebo část dokumentu, která již byla označená). Například pokud budeme hledat datum schůzky, bude se nacházet někde na začátku dokumentu. Seznam účastníků tam najdeme taktéž. Pozice může být absolutní, či relativní.
- Informace o délce elementu.
- Informace o pozici elementu na řádku – zda je spíš více vlevo, nebo více vpravo.
- Všechny délky a pozice je třeba popsat podle různých kritérií. Někdy jimi bude procentuální část vzhledem k nadřazenému elementu, jindy počet slov, jindy počet znaků, či vět, nebo odstavců.
- Samotný obsah textu – v neposlední řadě.

Dále bude třeba definovat další rysy, podobně jako v nástroji Lixto

- Přítomnost elementu před/za, či nepřítomnost elementu
- Přítomnost elementů uvnitř elementu
- Apod. – podrobněji dále.

Toto všechno se dá využít a popsat. V předchozích kapitolách byla popsána základní teorie neuronových sítí. Jejich použití by mohlo dát model, který splňuje tyto požadavky:

- Lze snadno popsat. Každý vstup neuronu by byl aplikací některých informací popsaných výše. Je na uživateli, které z nich shledá důležitými a jakou jim přiřadí váhu.
- Díky tomu, že neuron si uživatel „postaví“ sám, je vyřešen problém vysoké řídké dimenzionality. Zároveň bude model dobře čitelný.
- Neuronové sítě tolerují nepřesná data.
- Bude možné využít i vícevrstvé sítě, pokud by bylo třeba. Ale sama informace o ostatních elementech tvoří model vícevrstvé sítě, jelikož ostatní elementy jsou samy o sobě neurony. Tzn. výsledný model nebude lineární klasifikátor. Vícevrstvá síť by spíše tvořila možnost alternativ. Jednalo by se o uzly AND a OR – ale na to má systém lepší nástroje.

Poslední a neméně důležitá podmínka je, aby byl systém předvídatelný. Jedná se o ruční klasifikátor, takže uživatel musí mít představu, co se uvnitř děje, aby to mohl ovlivňovat.

Definice vzoru elementů

Je třeba rozlišovat element a vzor. Vzor je definice a element je konkrétní výskyt v dokumentu. Vzor je základní stavební kámen aplikace. Pomocí vzoru se definují podmínky, za jakých se může element (konkrétní výskyt vzoru) navázat na určitou oblast v dokumentu, jaké bude mít ohodnocení a zda bude takový element na základě ohodnocení vůbec platný. Dále lze také definovat jakým způsobem bude element převeden do datových sad.

Přehled základních parametrů:

Název
Musí být jedinečný. Rozlišuje se krátké jméno a celé jméno. Celé jméno je spojení krátkého jména a jmenného prostoru.

Jmenný prostor
Jmenné prostory jsou důležité, protože nám umožňují definovat vzory platné pouze pro určité dokumenty. Navíc umožňují lepší správu a pořádek. Jmenné prostory mají stromovou strukturu a jednotlivé názvy jsou oddělené tečkou, např:

Common.DateTime.Time
Common.NER.Person

Pokud rozpoznáváme určitou sadu dokumentů, je dobré si určit, které vzory budou při rozpoznávání platit. Má to několik výhod:

- Rozpoznávání je rychlejší, nevyhodnocuje se tolik vzorů.
- Rozpoznání je přesnější, než kdyby se testovaly i neplatné vzory s nízkým score.
- Je to snazší, než vyladit ruční klasifikátor tak, aby systém vždy automaticky poznal, že něco někam nepatří na základě kontextu celého dokumentu.

Příklad použití:

Uživatel má svojí sadu dokumentů. Vytvoří si svůj vlastní jmenný prostor **Můj Jmenný Prostor.** Pro něj si nadefinuje vzory platné ve všech jeho dokumentech (jsou-li takové). Pod tímto jmenným prostorem si založí názvy **NATO Meeting Minutes** a **OSN Meeting Minutes.** Vytvoří si pravidlo, které bude platit v Můj Jmenný Prostor.OSN Meeting Minutes a pokud se takový jmenný prostor vyskytne v dokumentu, potom pravidlo platí.

Condition Pattern Reference
Obsahuje odkaz na kořenovou podmínku, která musí být splněna, aby byl vzor platný. Condition pattern bude uveden v další části.

Minimum binding threshold
Minimální vnitřní hodnota výstupní podmínky pro to, aby byl element označen jako platný a mohl v dokumentu nadále existovat.

Binding Order:
Důležitý údaj. Říká v jakém pořadí se má element pokusit hledat svoje umístění. Elementy se totiž vážou v přesném pořadí.

Condition Pattern

Každý condition pattern obsahuje nějaký rys, nebo jiné condition patterny. Jedná se vlastně o neuron, který byl definován ručně. Vhodným seskupením ConditionPatternů lze vytvářet funkce AND, OR, či dokonce Maximum, nebo jinou funkci. Lze také zvolit výstupní funkci – skoková, sigmoida apod. Systém v současné době podporuje pouze sigmoidu, nebo lineární přechodovou funci.

Parametry:

Type	Informace zda se jedná o podmínku, či koncový rys (Feature pattern)
Threshold	Práh neuron
FeaturePattern	Pokud je typ podmínky koncový rys, ponese se zde informace o koncovém rysu.
Conditions	Seznam podřízených podmínek ConditionPatternReference. Každá podmínka obsahuje buď FeaturePattern, nebo Conditions. Conditions jsou podřízené neurony, které jsou spojené váhou Weight, uvedenou v referenci. Vytváří se tak strom podmínek.
Output type	Typ výstupu – Suma – klasicky sečte výstupy podřízených neuronů a vynásobí vahama. Max – vybere maximální hodnotu podřízeného neuronu (nezávisle na váze) a přiřadí jí na výstup neuronu.
FunctionType	Typ výstupu funkce, sigmoida, skoková apod. Nebude aplikováno u max.

Condition PatternReference

Obsahuje důležitou informaci – váhu mezi neurony. Má ale I jednu velice užitečnou výhodu. Reference nemusí odkazovat pouze na samostatnou podmínku, specifickou pro daný vzor, ale i na jiný vzor. Tím pádem se k dané podmínce připojí kořenová podmínka odkazovaného vzoru. To umožňuje znovupoužitelnost, kterou se práce později bude zabývat podrobněji.

Rysy (Feature Pattern)

Třída FeaturePattern udává nějaký koncový rys – například obsah textu či velikost písma, nebo výskyt elementu v okolí.

FeaturePattern bude mít na starosti jednu věc – vracet informaci o kvalitě nalzeného rysu v určitém intervalu, pokud možno 0..1. Konkrétní implementace bude záležitostí odvozených tříd. Které třídy to budou již bylo stručně popsáno v úvodní části.

4.3.1 Vázací podmínky (Feature binding)

Speciální rysy jsou takzvané vázací rysy. Od ostatních se liší tím, že na jejich základě systém rozhoduje, kde bude začátek a konec elementu. Pro účely vázacích vzorů, systém projíždí všechny FeaturePattern nezávisle na váze a umístění v neuronové síti a snaží se aplikovat všechna pravidla na

danou oblast. Na základě toho vznikne různé množství elementů, dále se testují další rysy a teprve potom se rozhodne, které z nich platí a které jsou chybně navázané.

Vázací rysy jsou například:

- Přímé vázání na element – na základě jiného elementu vytvoří nový element, nejjednodužší možnost
- Skupina elementů – váže se na základě nějaké skupiny elementů s danou tolerancí vzdáleností nejbližších od sebe, ukončovací elementy zase říkají, kdy nekompromisně vázaní zastavit a ukončit.
- Koncové body – element je určen skupinou elementů pro začátek a pro konec.

Vázací rysy jsou vlastně odvozené třídy od Feature Binding. Umí vracet ohodnocení a dají se napojit na condition pattern.

4.3.2 Alternativní názvy

Alternativní názvy posilují jazyk o jednu příjemnou vlastnost. Představme si situaci, kdy máme vytvořenou sadu popisků typu Datum, Název organizace, Účastníci…atd… Víme, že za každým názvem najdeme patřičná data. Někdy je můžeme nalézt snadno – za datumem bude následovat element datum, za účastníkama zase seznam osob. Ale někdy data obsahují jen otevřený text. Jak určíme hranice, kde končí?

Není to snadný úkol, ale jedna z možností je říci, že určitě končí elementem Datum, Název organizace a účastníci - a to z důvodu, že za těmito elementy se nacházejí jiná data. Bylo by pracné vypisovat všechny tyto elementy do podmínky. A co když se objeví později definice nových vzorů, bude potřeba dopsat nové odkazy? Jednodužší řešení je přiřadit skupině těchto elementů alternativní název, například Common.Caption a pak do omezující podmínky oblasti dat přiřadit ukončující podmínku Common.Caption.

Alternativní název poslouží v široké škále případů. Další možností je definovat alternativní definice vzorů. Máme dva vzory, každý má zcela odlišnou definici, ale mají se pro okolní elementy tvářit jako totéž. Nadefinujeme každý zvlášť (třeba i v jiném jmenném prostoru) a přiřadíme oběma stejné alternativní jméno, nebo určíme jeden jako primární a ten druhý bude mít alternativní název jméno prvního.

4.4 Popis rysů a vázacích podmínek

Rysy a vázací podmínky sice netvoří jádro systému, jdou snadno přidávat a ubírat, ale vytváří celou filozofii popisu dokumentů. Jak již bylo řečeno, rysy se dělí do 2 skupin:

- Běžné rysy
- Vázací rysy

Obě skupiny mají jedno společné – umí vracet ohodnocení. Vázací podmínky se ale přesto mohou používat pouze na ohodnocení a ne na vázání – mají vlastnost binding, kterou lze nastavit na false. Rysů a vázacích podmínek je velké množství proto budou uvedeny pouze ty nejdůležitější plus příklady použití.

4.4.1 Text Binding

Toto je jeden z nejzákladnějších rysů. Umožňuje vázání na slovo, či skupinu slov. Vrací ohodnocení, které říká jak moc vázání odpovídá definici. Definuje se seznamem vět, které má vzor obsahovat (takže je to pole polí řetězců).

Text Binding pracuje ve dvou modech:

- SingleBinding
- Multi Binding

Single Binding je jednodužší možnost, která neumožňuje žádnou toleranci. Zkrátka nalezená slova musí přesně odpovídat některým z uvedených vět v definici. Takže pokud bude například v definici toto:

Not Attendants

Non Attendants

Non Participants

Element se naváže pouze na tyto dvě dvojice slov, na nic jiného. Slova musí dodržet přesné pořadí a nesmí mezi nima být nic navíc.

Multi Binding je tolerantnější, toleruje pořadí slov a dokonce ani slova nemusí být úplně vedle sebe – dá se nastavit tolerance počtu přeskočených slov. Multibinding pak vrací ohodnocení v závislosti na tom, jak moc dobře se nalezená slova shodují se slovy v definici. Multibinding nachází shluky slov z definice, označí je elementem a spočítá skore.

Příklad:

Not Attendants

Non Attendants

Non Participants

V předchozím případě se element navázal pouze přesně na jeden z těchto tří výskytů skupin slov. Multibinding pracuje v určité toleranci. Zaprvé je možné zavést toleranci vzdálenosti slov, takže by příjmul i slova Non …cokoliv… Participants pro případ tolerance vzdálenosti 2 slova. Zadruhé příjme i slova Not Participants, protože Not a Participants se obojí vyskytuje v definici slov, přestože Not Participants uvedeno v definici není.

Výpočet ohodnocení:

Pro každý záznam v definici (Not Attendnats, Non Participants, Non Attendants) se spočítá blízkost. A to tak, že se vezme počet slov, které sedí s textem v elementu. Potom se spočítá poměr délky elementu a vzoru ve slovech a pokud je číslo větší než 1, překlopí se – tzn. Pokud je element dvakrát delší než vzor, bude se násobit číslem ½, pokud je poloviční jako vzor, bude to opět ½. Nakonec se vezme v úvahu délka znaků slov, které element obsahuje ku délce znaků vzoru a opět se výsledek upraví tak, aby poměr byl menší než 1. Výsledné hodnoty se znásobí a každému vzoru se přiřadí podobnost s elementem. Nebere se v úvahu pořadí slov, což může někdy vadit, ale zatím to v extrakci nevadilo, proto to nakonec nebylo do výpočtu zahrnuto.

Z těchto výpočtů se spočítá průměrná hodnota podobnosti a maximální hodnota podobnosti. Výsledek bude:

0,5 Maximální hodnoty+ 0,5 Průměrné hodnoty

Tento empirický vzorec bere v úvahu důležitost nejlepšího výsledku a podpůrné skore všech ostatních výsledků.

Konflikty:

Při navázání na text hrají konflikty velice důležitou roli. Vyhodnocují se konflikty pro elementy, které jsou vázané na text, které mají povolené vyhodnocení konfliktů a jsou ve stejném pořadí vázání – to je důležité, protože pokud by neměly stejné pořadí vázání, potom by se mohly zpětně zrušit a celá klasifikace by začala dávat nesprávné výsledky.

Konflikty se vyhodnocují pouze pro elementy, které se překrývají celé. Potom jsou aplikována následující pravidla:

- Preferují se element y s vyšším ohodnocením
- Preferují se elementy s větší délkou
- Priorita od uživatele také hraje roli

Tyto tři údaje se vynásobí (poměr délek se bere vždy k nejdelšímu elementu v kolizní skupině).

Postup je potom následující:

Pro každý element, který byl v tomto kroku přidaný, který má povoleno řešení konfliktů a který má vázání na text, proveď:

- Vezmi všechny vnitřní elementy
- Spočítej pro každý score, urči vítěze a všechny ostatní přidej do seznamu pro odstranění

Na konci algoritmu se všechny elementy ze seznamu odstranění smažou.

Tímto nástrojem lze třeba řešit situaci:

Attendants

Non Attendants

Navážou se tři elementy – jednou NonAttendants a dvakrát Attendants. Jenže spodní attendants je v konfliktní skupině Non Attendnats a bude „vyzván k souboji o přežití“. Souboj by mohl přežít (za předpokladu stejných priorit) jen tehdy, kdyby měl dvojnásobné ohodnocení – protože délka spodního elementu NonAttendants je 2x větší. Tato situace by nastat mohla v případě okolních podmínek. Ale protože oba elementy za sebou očekávají seznam osob, s nejvyšší pravděpodobností element Attendants prohraje, protože základní ohodnocení budou téměř stejná.

Slova, která element nemá obsahovat

Zatím byla popisována pouze slova, která element má obsahovat. Co se slovy, které nesmí? Text binding umožňuje zadat pouze požadovaná slova. Ale zde přichází malý trik. Můžeme totiž vytvořit další podmínku TextBinding, která bude mít zápornou váhu a vypnutou vlastnost Binding. Tudíž nebude v seznamu vázání (nebude se podle definice slov vyhledávat), ale spočítá se skore slov v definici ku obsahu. Čím lepší bude, tím horší bude výsledné ohodnocení (záporná váha u podmínky).

4.4.2 Simple Number Binding

Stejně jako Text Binding se váže na text, Simple Number Binding se váže na přirozená čísla – ne reálná, protože desetinná tečka rozděluje číslo na více slov. Na desetinná čísla by bylo potřeba jiné pravidlo.

Vázání na čísla zkrátka vyhledá všechna čísla v dokumentu, vytvoří elementy a každému přiřadí ohodnocení podle tabulky intervalů. Využití je celkem jasné, není třeba uvádět příklady.

4.4.3 Element binding

Toto vázací pravidlo pracuje ve 2 modech – Single a Multi.

Single Binding se váže právě na jeden element z množiny a Multibinding se váže na shluk elementů z množiny při dané toleranci vzdálenosti . To je velice podobné jako v případě Text Binding. Ale jsou tu navíc ještě stop elementy – v některých případech je totiž třeba vázání nekompromisně zastavit při příchodu nějakého elementu – například konce odstavce, konce řádku tabulky atd.

Další dvě pomůcky jsou MinimumBindingScore a MaximumBindingScore. Každý Element má přiřazené ohodnocení a pokud přesáhne MaximumBindingScore, element se ukončí. MinimumBindingScore zase říká, že při ukončení (StopElementem, či velkou vzdáleností mezi elementy) se otestuje score a pokud je menší, element se nevytvoří.
Poslední pomůcka je ElementCounter, který každému elementu přiřadí maximální počet (implicitně neomezeně). Pokud je počet překročen, element se ukončí a naváže.

4.4.4 Element boundary binding

Najde první výskyt elementu z množiny startovních elementů a druhý výskyt z množiny koncových elementů a naváže na ně výsledný element. Uživatel si může rozhodnout, zda počáteční či koncový element bude zahrnut do výsledného elementu.

4.4.5 Element Occurence

Nejedná se o vázací podmínku, pouze o ohodnocovací rys. Hledá výskyty jiných elementů v okolí zdrojového elementu (či uvnitř jeho samého). Pracuje opět ve 2 modech.
V prvním hledá výskyty elementů z definované množiny s daným ohodnocením a rozsahem okolí a sčítá výsledky. Výskyty elementů jsou omezené tzv. stop elementy. Pomocí toho lze nadefinovat třeba dotaz:

„Vrať mi ohodnocení (znásobené def. Váhou) všech elementů A a B v okolí (-50,50) slov. Ale oblast bude ohraničena elementy konec odstavce.“

Jinými slovy, hledáme elementy A a B v tomtéž odstavci.
Ve druhém hledá pouze jeden konkrétní výskyt a bere přitom v úvahu ukončovací elementy – to jsou elementy, které se když se nachází mezi zdrojovým a nalezeným elementem, snižují celkové skore výsledku. Pokud je elementů nalezeno víc, vrátí se pouze výsledek s nejvyšším ohodnocením.
Při vyhodnocování se bere v úvahu i samotné ohodnocení nalezeného elementu. Rozsah okolí je nastaven intervalem a může se řídit levým okrajem zdroje, oba okraji zdroje, nebo pravým okrajem zdroje.
Element Occurence má ještě jedno využití, ač k tomu rys původně nebyl navržen. Stop Elementy totiž neznamenají automaticky, že se hledání na nich ukončí, ale pouze snižují ohodnocení výsledku. Na druhou stranu, je možné definovat u StopElementu opačné skore, tím pádem StopElementy nebudou ohodnocení snižovat, ale zvyšovat. To může pomoci nadefinovat např. dotaz typu:

„Najdi element A v určitém okolí, který je ale v jiném odstavci“.

V překladu by to znamenalo:

Hledej element A a pokud mezitím najdeš element konec odstavce, zvyš ohodnocení.

4.4.6 Další rysy

Mezi další nástroje jazyka patří například:

SplitElementBinding	Vytvoří elementy pomocí jiných dělících elementů v určité oblasti (oblast je opět definovaná elementem).
TableBinding	Umožní vázat elementy na sloupce (ale i řádky, tam to však nemá velký význam, zde stačí navázat na element TableRow). Podmínkou je, že musí být nalezen nějaký popisek (třeba hlavička sloupce s názvem sloupce) a lze zadat za jakých podmínek se vázání ukončí (např. nalezení jiného popisku).
DataScore	Spočítá ohodnocení na základě definice dat (viz později).
ElementsAfter/ElementsBefore	Zjistí, zda se za či před elementem nenechází jiný element (či skupina elementů). Lze nastavit podmínku AND nebo OR. Vrací 0, nebo 1. Podobá se ElementOccurence a teoreticky pomocí toho lze nasimulovat, ale složitě.
BetweenElements	Náhražka kombinace ElementsAfter a ElementsBefore.
ElementsInside	Podobá se DataScore. Bere v úvahu počty elementů obsažené uvnitř. Definují se intervaly určitých elementů (počty) a jejich ohodnocení. Je to vlastně alternativa k data score v případě, že není definovaná sada dat, které má element obsahovat.
ElementsInsideCoverage	Místo počtu elementů počítá pokrytí elementů v oblasti (v jiném elementu) ve znacích. Takže díky tomu můžeme vyjádřit, že položka Attendants by měla být z 50-100% znakově pokryta elementy Person. Znakové pokrytí lépe odpovídá vizuálnímu pokrytí, protože pokrytí ve slovech je velmi zkresleno počtem slov a počtem krátkých oddělovacích znaků.
ElementVisualParameters	Vrátí ohodnocení v závislosti na volných místech okolo elementu (tabelátory, odstavce, konce řádků), na velikosti písma, tučnosti písma a na počtu velkých písmen ve slově. Všechny tyto parametry se dají váhovat dle potřeby. Jedná se o velici důležitý nástroj pro vyhledávání důležitých popisků. Často se totiž stává, že nalezený text odpovídá hledanému popisku, zde potom rozhoduje zejména viditelnost (kromě okolních elementů). Důležité záchytné body v dokumentu jsou téměř vždy dobře viditelné.

Length	Délka elementu ve znacích či slovech. Lze nastavit požadovaný interval, nebo bez intervalu – potom je ohodnocení rovno určité délce / zvolená konstanta.
PositionInRegion	Pomocí tohoto rysu můžeme říci, že chceme aby se náš element nacházel v prvních 0-50 slovech nadřazeného elementu (zvolíme kterého). Můžeme volit, zda se bude jednat o absolutní, či relativní pozici, o tom, zda chceme počítat vzdálenost od levého, či práveho rohu, nebo dokonce můžeme zvolit relativní délku pokrytí jiného elementu ve slovech. Například takto můžeme nadefinovat, že hlavičkové informace (Název, Datum atd…) by se měly nacházet na začátku dokumentu.
StopWordsDensity	Hustota stop slov nám udává, zda daný element obsahuje spíše běžný text, či nějaké informace. Experimentálně bylo zjištěno, že volný text obsahuje kolem 10-20% stop slov, zatímco různé tabulky apod, obsahují kolem 1% stop slov.
TableParametersFeature	Pokud se element nachází v tabulce, tak díky této vlastnosti lze získat ohodnocení, pokud tabulka splňuje určitá kritéria (rozsah počtu sloupců či řádků).
TableFeature	Pokud je element v tabulce, lze získat ohodnocení v závislosti na tom, na kterém řádku či sloupci se nachází – jsou určeny rozsahy řádků či sloupců, pokud se do nich element vejde, je vráceno ohodnocení, pokud ne, může být vráceno záporné ohodnocení (nastaví uživatel). V příští verzi by bylo možné se odkazovat na jiný element v závislosti na poloze zdrojového elementu – například 2 řádky nad tebou v tomtéž sloupci by se měl nacházet element… Ale tato možnost v aplikaci využita nebyla a není v tuto chvíli implementována.
TableElementOccurence	Hledá výskyt elementu na řádku, či ve sloupci – obvykle se bude jednat o hlavičku sloupce či řádku. Totéž by se dalo popsat i pomocí TableFeature, pokud by ta možnost byla doimplementovaná, ale tento rys je pro běžnou práci snazší.
Horizontal Alignment	Tento rys měl udávat pozici elementu v závislosti na levém a pravém okraji tiskutelné plochy. Vzhledem k nedostatku informací poskytovaným komunikačním rozhraním (viz. Úvod) tato možnost nemohla být implementovaná.

ColumnOccurence	Sčítá výskyty daných elementů ve sloupci – je to vhodné v případě kdy chceme posílit svojí pozici ve sloupci pro daný element – pokud v tom samém sloupci byly například stejné elementy s vysokým score, pak jsme zde správně, jinak ne.

4.5 Algoritmus extrakce

1. Seřazení do skupin podle BindingOrder
2. Pro každou skupinu podle pořadí a pro každý vzor ve skupině:
    a. Spusť vázací podmínky
       V této chvíli se naváží elementy daného vzoru.
    b. Zavolá se AfterBinding pro všechny přídavné moduly.
    c. Elementy se ořežou podle oblastí, ve kterých mají platit (volitelné pro každý element).
    d. Jsou aplikovány uživatelské elementy.
       Uživatelské elementy jsou uživatelem označené, platné pouze v rámci jednoho dokumentu. Má to praktické využití v případě, kdy je neproduktivní upravovat definice vzorů a místo toho je lepší ručně nadefinovat, že na konkrétním místě má, nebo nemá element být – viz v dalších kapitolách.
    e. Proběhne první ohodnocení právě navázaných elementů.
    f. Jsou vyřešeny konflikty pro elementy vázající se na text.
    g. Elementy s ohodnocením nižším, než MinimumBindingScore, jsou odstraněny.
    h. Spouští se Akce.
       Akce jsou uživatelem definované operace, definované volitelně pro vzory. Bude o nich řeč dále.
    i. Vytvořené elementy jsou opět ohodnoceny (vliv akcí, či odstraněných elementů apod, mají vliv na změnu ohodnocení).
3. Všechny elementy jsou znovu ohodnoceny.
   Je to z důvodu, že některé elementy se odkazují na jiné, které jsou navázány až v budoucnu – nyní se jejich vliv projeví. Sice to neumožňuje zkontrolovat minimum binding score, ale může to pomoci při rozhodování, zda se daný element bude převádět do dat, či nikoliv.
4. Jsou provedeny akce na konci rozpoznávání (například řešení uživatelem def. konfliktů).
5. Provede se další ohodnocení.
6. Provede se analýza koreferencí a následně převod do dat.

4.6 Named Entity Recognition

Pojmenované entity tvoří důležitou součást extrakce. V projektu nejsou použity žádné externí nástroje, místo toho je zde možnost vytvořit si vlastní sady pojmenovaných entit jako běžné elementy. Na druhou stranu to nijak nevylučuje použití externích služeb, dokonce v kombinaci s vlastoručně definovanými elementy.

V projektu je použit pouze jeden typ pojmenovaných entit a to entity Person, které se skládají z elementů FirstName a SurName. FirstName a SurName jsou obyčejné elementy, které se vážou na TextBinding, mají tedy textovou sadu, na kterou se navážou, definovanou uživatelem. Person je potom element, který se váže na FirstName a SurName bez tolerance vzdálenosti. Tzn., tyto elementy musí být vedle sebe. Tento přístup má nejmenší chybovost. Pokud bychom chtěli pro určité oblasti v dokumentu definovat vlastní pravidla elementu Person, stačí vytvořit vlastní element a přiřadit mu alternativní název Common.NER.Person. To, že je nějaká oblast označena jako Person však neznamená, že se musí o osobu skutečně jednat. Uživatel se může rozhodnout, že v dané oblasti bude každou položku Person brát v potaz jako osobu, jindy se může rozhodnout otesotvat okolí, zda je element platný – na to jsou různé nástroje, například vlastní element, který se naváže na Person a pokud nesplní požadavky na okolí, výskyt Person smaže.

Vzhledem k tomu, že elementy Person jsou plně definované pomocí nástrojů jazyka bez čehokoliv externího, lze podobným způsobem nadefinovat elementy Organisation, Place apod. V Projektu to však uděláno není, protože bez externích nástrojů by bylo potřeba mít obrovskou databázi názvů.

4.6.1 Přidávání názvů

Je dobré mít množst přidávat názvy do seznamu jmen. První možnost je označit jméno a kliknout např. na tlačíto Add First Name, či Add Sur Name. Pokud jméno není v seznamu uvedeno, automaticky se přidá a uloží. Pro ostatní pojmenované entity tato možnost není, avšak přesto lze otevřít textový editor a ta jména tam přidat ručně.

Další možností je říci při extrakci: Obsah nalezených elementů automaticky přidávej do seznamu jmen. To lze udělat pomocí speciální akce AddTextToPatternTextBinding. To bylo využito například v tabulce, která obsahovala tabulku s hlavičkou FirstName a LastName. V takovém případě je přesnost vysoká natolik, že si můžeme dovolit každý nalezený element přidat do seznamu. Seznam je však třeba čas od času zkontrolovat, zda se v něm nenacházejí nesmysly.

4.6.2 Analýza odkazů

Anglicky Coreference Analysis. Toto se využívá zejména u pojmenovaných entit, ale lze to zobecnit na jakýkoliv element. Opět jí lze řešit i externím modulem, ale systém má v sobě implementovanou základní funkčnost.

Odkazy jsou například toto:

Milan Pospíšil je element Person

Milan je také element Person, oba elementy patří do jedné třídy, ale je zřejmé, že Milan bude nejspíš odkazovat na osobu pojmenovanou Milan Pospíšil.

Každý vzor, který chce řešit odkazy si zvolí název pojmenované třídy. To je pro případ, kdy bychom chtěli dva zcela rozdílné vzory přidat do jedné třídy odkazů. Následně systém rozdělí elementy do tříd podle názvu třídy. Potom pro každou třídu probíhá proces odděleně.

Každý element ve třídě se porovná s ostatními elementy a přiřadí svůj odkaz jednomu z nich podle ohodnocení.

Ohodnocení se spočítá následovně:

- Spočítá se počet společných slov zdrojového a cílového elementu.
- Odečte se od něho počet slov, které jsou v zdrojovém a nejsou v cílovém elementu, slova, která obsahuje cílový element a zdrojový ne, nás vůbec nezajímají, protože to se u analýzy odkazů předpokládá, že zdrojový element bude obsahovat zkrácený počet slov než cílový element.

- Ohodnocení se vynásobí délkou cílového elementu. To má logiku, čím delší je cílový element, tím větší je šance, že se odkazujeme na plný název.

- Dále hrají roli priority, které však nakonec nebyly v projektu otestovány ani použity. Smysl priorit byl například v tom, že určité oblasti elementů budou mít větší ohodnocení na to aby se staly kandidáty na odkaz. Například osoby v oblasti Attendants apod.

Příklad:

Mějme dva elementy:

Milan Pospíšil, Milan

Spočítáme ohodnocení pro element **Milan Pospíšil**:

Porovnání s elementem Milan Pospíšil

Počet společných slov: 2, počet rozdílných slov: 0

Délka cíle: 2

Celkové ohodnocení: 4

Porovnání s elementem Milan

Počet společných slov: 1, počet rozdílných slov: 1

Délka cíle: 1

Celkové ohodnocení: 0

Bude tedy vybrán cíl Milan Pospíšil, což je správně.

Ohodnocení pro element **Milan**:

Porovnání s elementem Milan Pospíšil

Počet společných slov: 1, počet rozdílných slov: 0

Délka Cíle: 2

Celkové ohodnocení: 2

Porovnání s elementem Milan

Počet spol. slov: 1, rozdílných: 0

Délka cíle: 1

Celkové ohodnocení: 1

Bude tedy vybrán cíl Milan Pospíšil, což je správně.

Nedostatky

Systém splňuje opravdu jen základní funkčnost. Analýza odkazů není tématem práce. Například neřeší věci typu: Pospíšil M.

Najde osobu Pospíšil a zkratkou se nezabývá. Toto by pochopitelně nebyl takový problém doimplementovat, ale analýza odkazů nebyla v hlavním zájmu práce.

4.7 Rozpoznávání času a data

4.7.1 Rozpoznávání času a data

Opět se jedná o relativně provizorní verzi, která si nekladla za cíl rozpoznávat čas a datum s vysokou přesností. Na to už dávno existují hotové kvalitní nástroje. V projektu byla spíše snaha popsat tyto entity pomocí existujících nástrojů a zhodnotit výsledek.

Takto popsané entity se nerozpoznávají s příliš vysokou přesností, občas se stane, že se nějaké neplatné datum či čas označí tam, kde být nemělo. Na druhou stranu pro většinu dokumentů to stačilo.

Jak rozpoznání probíhá? Nejprve jsou nalezeny základní stavební kameny, tj. čísla, která mají přiřazené ohodnocení podle obsahu (například čísla větší než 60 nebudou dobrým kandidátem na čas). Na druhou stranu je povolit musíme, protože co když bude někde údaj 100 hodin. Jedná se bezpochyby o čas. Dále jsou nalezeny názvy měsíců, různé přípony st, th (pro čísla dnů), názvy typu seconds, hours, minutes… Oddělovače :, ., /, - atd.

Základní kameny se dělí do těchto kategorií (uvedeme jen ty nejdůležitější):

- DateNumber – číslo pro datum
- TimeNUmber – číslo pro čas
- Month – měsíc (ve slově)
- Koncovka u čísel (th, st, rd, nd…bývá jen za číslem dne)
- Oddělovače času
- Oddělovače data
- Přídavné názvy (jako např. minutes, seconds, hours…)
- Časové údaje (a.m, p.m, morning, midnight, afternoon)

Určit nečíselné údaje je poměrně snadné a přesné. Problém je s čísly. Systém nezkoumá pořadí čísel vzhledem k formátu data a času – to by šlo pouze pomocí externího kódu. Pouze se zabývá okolím čísel a na základě toho rozhoduje, zda se jedná spíše o číslo data, či o číslo času. Vyhraje položka s vyšším ohodnocením. Tím pádem systém pouze říká, kde je datum a čas, ale už neříká konkrétní hodnotu.

Na základě toho se na tyto výskyty „natáhnou“ elementy Time a Date, každý má definovanou sadu elementů, které má pokrýt, sadu elementů, které už nesmí pokrýt a toleranci vzdálenosti.

Příklad

19th of january 2009 17:00

19	bude vyhodnoceno jako číslo data, protože má za sebou koncovku th
of	používá se často v datumu, je uvedeno v seznamu slov, které se v datu vyskytují
january	bez pochyby název měsíce
2009	zde bude mít větší ohodnocení číslo datumu, z okolí to zřejmé být nemusí – měsíc před tím nemusí rozhodovat (vezměme v úvahu 19th of january 17:00)

17 Jedná se téměř jistě o čas, jelikož za číslem následuje oddělovač času. Datum zde bude označeno také, ale vyhodnotí se konflikt s časem, který bude mít vyšší hodnotu

00 Opět se jedná o číslo

Definice času a datumu má dobrý recall, ale nižší přesnost – je to dáno tím, že se nevyhodnocuje sémantika jednotlivých prvků vzhledem k formátům. Ve většině případů to nevadí, protože uživatel často na čas a datum váže vlastní elementy u kterých dále zjišťuje okolí a potom teprve vyhodnotí, zda je to to, co hledal.

4.7.2 Rozsah času

Rozsah času se definuje následovně:

Je použit rys ElementBinding, který se váže na elementy čas, konec odstavce, element rozsahu (-, to atd), či konec buňky tabulky. To umožní spojit například toto

17:00 <tab> 19:00, 17:00 to 19:00, 17:00-19:00 nebo dva časy v sousedních buňkách tabulky. Ukončovací elementy jsou například konec řádku, či konec řádku tabulky.

Vzor Rozsahu času není zcela přesná definice a má své nedostaky. Existuje zejména jako zjednodušení, aby si uživatel nemusel pokaždé rozsah času definovat sám. Pokud existují případy, kdy je rozsah času často určen špatně, lze na něj navázat testovací element, který rozhodne zda bude rozsah času odstraněn či ne a nebo si definovat vlastní doménově specifický rozsah, kterému lze poté jednoduše přiřadit Alternative Name na Common.DateTime.TimeRange.

Problém:

Může nastat situace, kdy jsou časy od sebe vzdáleny více jak jednu buňku, či jeden odstavec a přesto se element naváže. Co s tím? Jedna z možností je přidat rys ElementsInside a v případě více jak 1 výskytu buňky snížit hodnocení. Další možnost je omezit počet elementů konec odstavce v ElementBinding.

Rozsah data

Nebyl potřeba, není tedy definován, ale jednalo by se o podobný případ jako rozsah času.

4.8 Akce

Akce jsou předdefinované dodatečné operace nad elementy. V současné době má systém implementované dvě akce – AddToPatternTextBinding a DeleteOnBind.

4.8.1 Add to pattern TextBinding

Tato akce již byla zmíněna v sekci pojmenovaných entit. Umožňuje přidat obsah elementu do databáze jmen. Nemusí se nutně jednat jen o osoby, může jít o jakýkoliv vzor obsahující TextBinding, který má nastaveno Binding na true – to je z důvodu, kdy chceme použít další rys TextBinding na slova, která nemá vzor obsahovat. Předpokládáme, že vázací podmínka TextBinding bude u vzoru jen jedna, protože nemá smysl jich mít více.

4.8.2 Delete on bind

Toto je velice důležitá akce, protože rozšiřuje extrakční nástroj o mnoho možností. Zde je jejich přehled:

Ověření platnosti elementu

Často se stane případ, kdy chceme ověřit platnost existujícího navázaného elementu vlastními dodatečnými pravidly a rozhodnout se, zda je element platný a pokud ne, smazat ho. Lze to snadno provést tak, že se na původní element navážeme a pokud splní podmínku MinimumBindingScore, provede se akce, která původní element smaže.

Nahrazení původního elementu

Někdy se stane, že máme definovaný element, který je využívaný ve více dokumentech, ale my potřebujeme, aby pro něj v určité sadě platila jiná pravidla (například pro převod do dat). Jednoduché řešení je navázat nový element na původní a původní smazat.

Řešení konfliktů

Konflikty se již řešily u TextBinding, ale zde máme možnost řešit konflikty pro jakékoliv elementy a podle definice uživatele. Kromě přímého smazání lze totiž nastavit podmínky, za jakých se element smaže. Tyto podmínky jsou například porovnání ohodnocení, délky elementu ve slovech (nebo i znacích) a priority. Další možností je nastavit rozsah oblasti, ve které se bude mazat (mazání tedy není omezeno pouze na oblast elementu). A poslední vlastnost je ta, že si musíme zvolit elementy, které chceme podrobit testu na konflikt. Toto je důležité, protože jedině uživatel ví, které elementy mají být v konfliktu a které ne.

Tento přístup byl například použit při rozpoznávání data a času – zejména čísel DateNumber a TimeNumber – bylo zde třeba vyřešit, zda číslo patří spíše k času, nebo k datu a element s nižším ohodnocením smazat.

Další praktické využití bylo v případě rozpoznání časového záznamu u agendy, kde se objevoval údaj o délce záznamu (v čase), tudíž byl někdy považován za čas agendy a vytvářel tak chybný nový záznam:

15:00-15:30 Break 0:30

Čas 0:30 mohl být považován za začátek nového záznamu, což je chyba, protože patří k předchozímu záznamu. Řešení bylo okamžitě po navázání ItemTime navázat další element ItemTimeLength a pokud uspěl v podmínkách, které si definoval uživatel speciálně pro svojí sadu dokumentů, původní element ItemTime se smazal.

4.9 Převod do dat

V okamžiku, kdy skončí fáze nalezení elementů, je třeba tyto elementy převést do datových sad. Na základě datových dat lze potom jednoduše výsledek uložit do XML, nebo napsat vlastní převod do databáze. Převod do dat je také jedním z řešení alternativních definic – může se jednat o dva rozdílné elementy různě definované, které se však zapíší do dat pod shodným klíčem – to bylo například využito v jedné sadě testovacích dokumentů. Převod do dat sice funguje, ale není zcela vyladěn, tudíž existují některé sekundární parametry, které v současné době nejsou použité, nebo nefungují zcela správně – jedná se však spíše o parametry, které mají zpříjemnit uživateli orientaci.
Každý vzor může mít definici převodu vlastní, nebo se může odkázat na definici převodu jiného vzoru – to například v případě, kdy se jedná o 2 různé vzory se stejným převodem do dat.
Definice převodu dat vypadá takto:

Name	Jméno klíče, pod kterým bude položka uložena. Různé typy elementů můžou mít stejné jméno klíče a tudíž budou přidány do stejné sady dat.
PatternName	Konkrétní jméno vzoru, který převod do dat provedl.
RootData	Zda se jedná o zdrojová data, tj. zda element této definice začne s převodem, či bude volán při převodu otcovského elementu.
Count	Interval. Počet výskytů, které se v dokumentu očekávají. Pokud je počet výskytů elementů větší než je počet count, potom se seřadí podle ohodnocení a vyberou se ty nejlepší.
Data Attribute	Vniřní položky elementu. (kolekce)
MinimumScore	Minimální score elementu pro převod do dat.

Definice Atributů dat:

Name	Název atributu.
Elements	Elementy, které má atribut obsahovat. U každého elementu lze nastavit SearchInsideFirst na true a potom se bude nezávisle na DataSource prvně prohledávat výskyt elementu uvnitř.
Count	Počet elementů, interval. Pokud je elementů nalezeno více, potom se seřadí podle ohodnocení a vyberou se ty nejlepší.
DataSource	Zdroj dat. Data mohou být: **Uvnitř Elementu** **Nejbližší výskyt** – zleva, zprava, obojí – hledá se však pouze 1 výskyt

Search in columns V případě zapnuté volby se hledá ve sloupcích tabulky, volba data source určuje, zda se bude hledat výskyt směrem nahoru (zleva), dolu (zprava), nebo nejbližší.

4.9.1 Příklad

Mějme element **ItemRecord,** který obsahuje jeden časový záznam agendy s časem **ItemTime,** datem **HeadLineDate** a seznamem osob Person.

Definice bude vypadat takto:

ItemRecord

Count – 0, Max

Name – můžeme ponechat prázdné, bude to nahrazeno celým jménem vzoru

Root data – true

DataAttribute:

ItemTime	(1,1)	- jeden výskyt, Inside - uvnitř
Person	(0..Max)	- neomezené množství lidí, Inside – uvnitř
HeadLineDate	(1,1)	- jeden výskyt, Nearest Left, SearchInsideFirst Pokud by bylo datum uvnitř záznamu, vezme se uvnitř, jinak se hledá nejblížší výskyt vlevo.

Výsledná struktura záznamu bude potom vypadat například takto:

Test2.ItemData.ItemRecord

DataItem

Test2.ItemData.ItemTime

3 : 30 p

Common.NER.Person

heile

Test2.HeadlineDate

monday july 5 , 1999

Položek DataItem je (0…N). V tomto výpisu není uveden celý textový obsah elementu ItemRecord, v databázi či XML však uvedený být může.

4.10 Řešení problémů a chybné klasifikace

Některé problémy se nedají vyřešit prostou úpravou definice vzorů, jsou třeba ojedinělé, nebo příliš složité. Zde jsou popisy problémů a možnosti řešení:

V některých situacích je velice obtížné popsat konec datové oblasti.

Je možné dokument manuálně doplnit o značku konce oblasti a na jejím základě vytvořit element.

Na jednom konkrétním místě se neustále objevuje element, který by tam být neměl, v ostatních dokumentech vše funguje správně.

Pro konkrétní dokument je možné vytvořit uživatelský element a určit kdy a kde se vytvoří (nebo smaže).

Extrakce funguje přibližně na 90%, ale je vyžadována přesnost téměř 100%.

Není jiné řešení, než ručně projít výsledky a opravit chyby.

V některých dokumentech se klasifikují elementy, které se tam vůbec nemají co pohledávat.

Buď se zpřesní popis elementů (což nemusí být snadné), nebo se pro vybranou sadu prostě tyto vzory elementů vyřadí z jmenného prostoru a nebudou vůbec brány v potaz.

Někdy se tabulková data nenacházejí v tabulce a jsou odděleny tabelátory, potíž je v tom, že nadpisy se nachází v „prvním řádku“ tabulky, tj horizontálně.

Např:

Full name	*status*	*att. %*	*phone*	*company*	*e_mail*
Mr. Houman Alborzi	voter	100	+1 301 405 2775	University of Maryland	houman@cs.umd.edu
Mr. Dimitri Alexandrou	voter	100	+1 919 462 6544	Atmel Corporation	da@dctcorp.com

Systém tuto situaci neumí řešit, protože nejsou k dispozici vizuální parametry textu, které by mohli říci – ano, daný textový blok se nachází přesně pod tímto napisem. Počítat počty tabelátorů taky není možné, protože jedna věc je ve wordu používat pravítka a druhá věc rozdílné počty tabelátorů pro odsazení nestejně dlouhých bloků –standardně má být pomocí pravítek každý blok oddělen jedním tabelátorem, většina uživatelů to nedělá.

Jediné řešení, které by bylo možné, je při znalosti pořadí a obsahu elementů. Pokud se jedná názvy osob, telefonní čísla apod, je možné nadefinovat elementy, které se naváží na konkrétní obsah a neoznačené textové bloky s těžko rozpoznatelným obsahem klasifikovat podle okolí.

Možná by nebylo na škodu nadefinovat v budoucnu nějaký rys, který bude brát v potaz pořadí popisků (FullName, status, phone atd…). Ale daleko čistší by byly vizuální informace, protože tato řešení jsou příliš komplikovaná.

Mám seznam dat, které jsou vnořené v jiné datové oblasti, např:

Author: Pat Kinney
Mike McInnis-Assisting
Pat Kinney, Intermec Technologies
550 Second St SE
TELEPHONE: 319.369.3593
FAX: 319.369.3299
EMAIL: kinneypw@norand.com

Telephone, Fax a Email jsou popisky, ale patří pod data Author. Jenže data Author končí elementem popisek, takže se zastaví na Telephone.

Řešením je popisky telephone, fax a author nenazývat alternativním Common.Caption a navázat data Author. Teprve po navázání dat je možné navázat pomocný element s alternativním názvem Common.Caption na tyto 3 elementy. Díky tomu můžeme v další fázi navázat datovou oblat telephone, fax a Email, rozdělenou právě novými popisky.

4.11 Uživatelské rozhraní

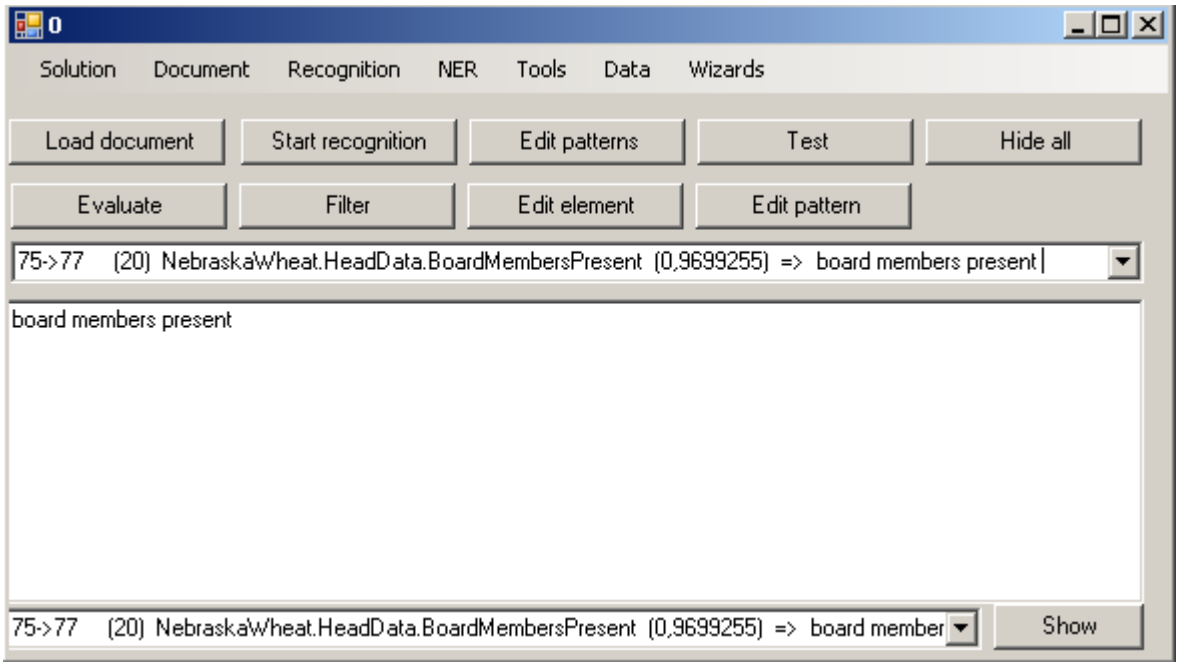
Uživatelské rozhraní je v současné době v rozpracovaném stavu. Umožňuje sice většinu operací, ale bylo příliš časově složité vytvořit vše co by bylo v aplikaci potřeba. Vylepšení budou zhodnocena alespoň teoreticky.

Uživatelské rozhraní se dělí do několika skupin:

4.11.1 Základní uživatelské rozhraní

Obsahuje základní nezbytné prvky k udržování aplikace, vzorů a jejich rysů.

Hlavní okno



Obrázek 4.2 – hlavní okno

Toto okno udržuje přístup ke všem ostatním oknům a je vždy „On Top“. Horní nabídka (drop down list) obsahuje všechny viditelné elementy v dokumentu, seřazené podle pozice. Obsahuje však jen ty, které si uživatel zvolí. Spodní nabídka obsahuje ty elementy, které označil uživatel v dokumentu. Tlačítko show daný element zobrazí uvnitř prostředního pole.

Prostřední pole obsahuje celý text uvnitř elementu nebo průchod ohodnocovacím stromem. Ten se hodí v případě, kdy chceme vědět, jakým způsobem došlo k výpočtu ohodnocení elementu. Ohodnocovací strom však z nedostatku času není zcela vyhovující, určitě by to šlo napsat lépe.

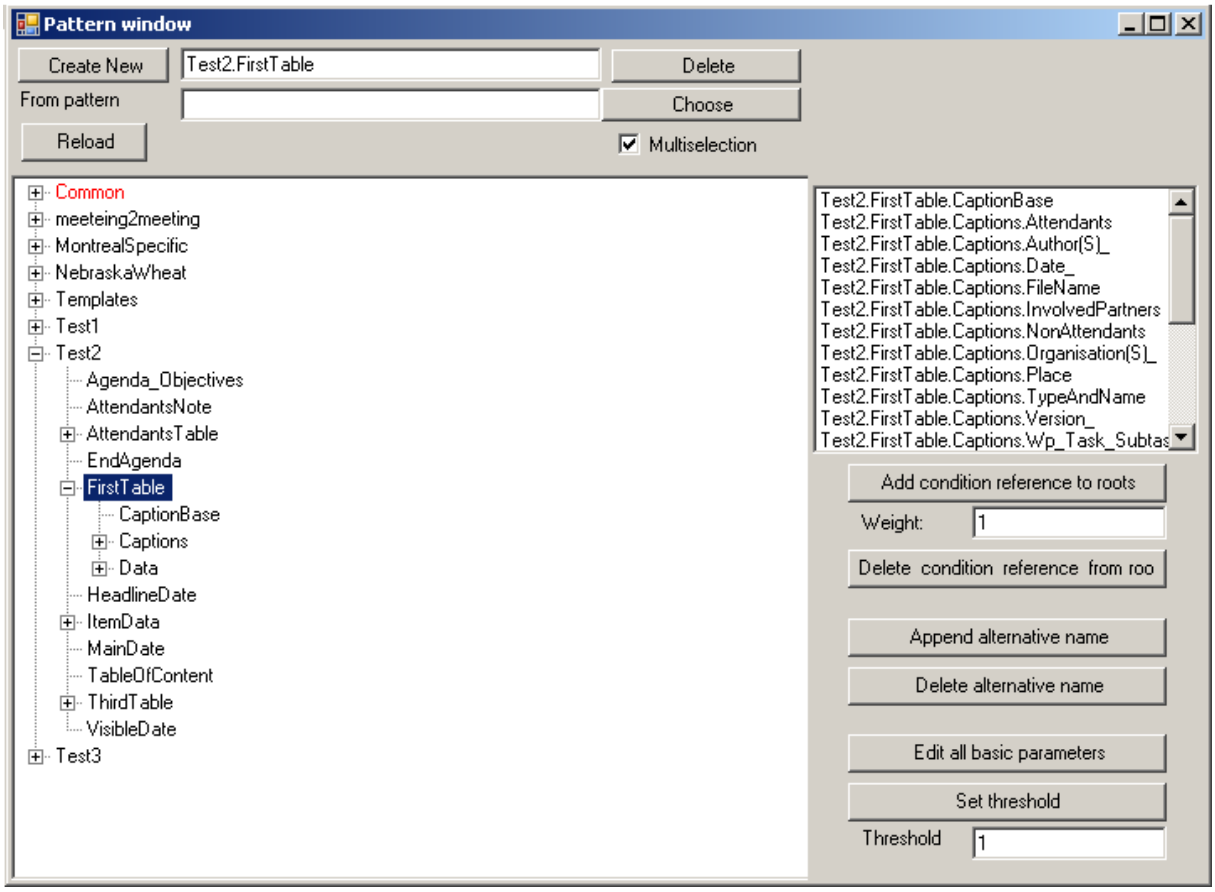
Element je pak vypsán ve tvaru:

75-77 (20) NebraskaWheat.HeadData.BoardMembersPresent (0,9699255) => board members present

75-77 Umístění v dokumentu ve slovech dle toku zleva doprava

(20)	Pořadí vázání (Binding order)
NebraskaWheat…	Celý název vzoru
0,9699255	Ohodnocení elementu
Board…	Vnitřní text elementu (do určité délky)

Okno pro výběr vzorů



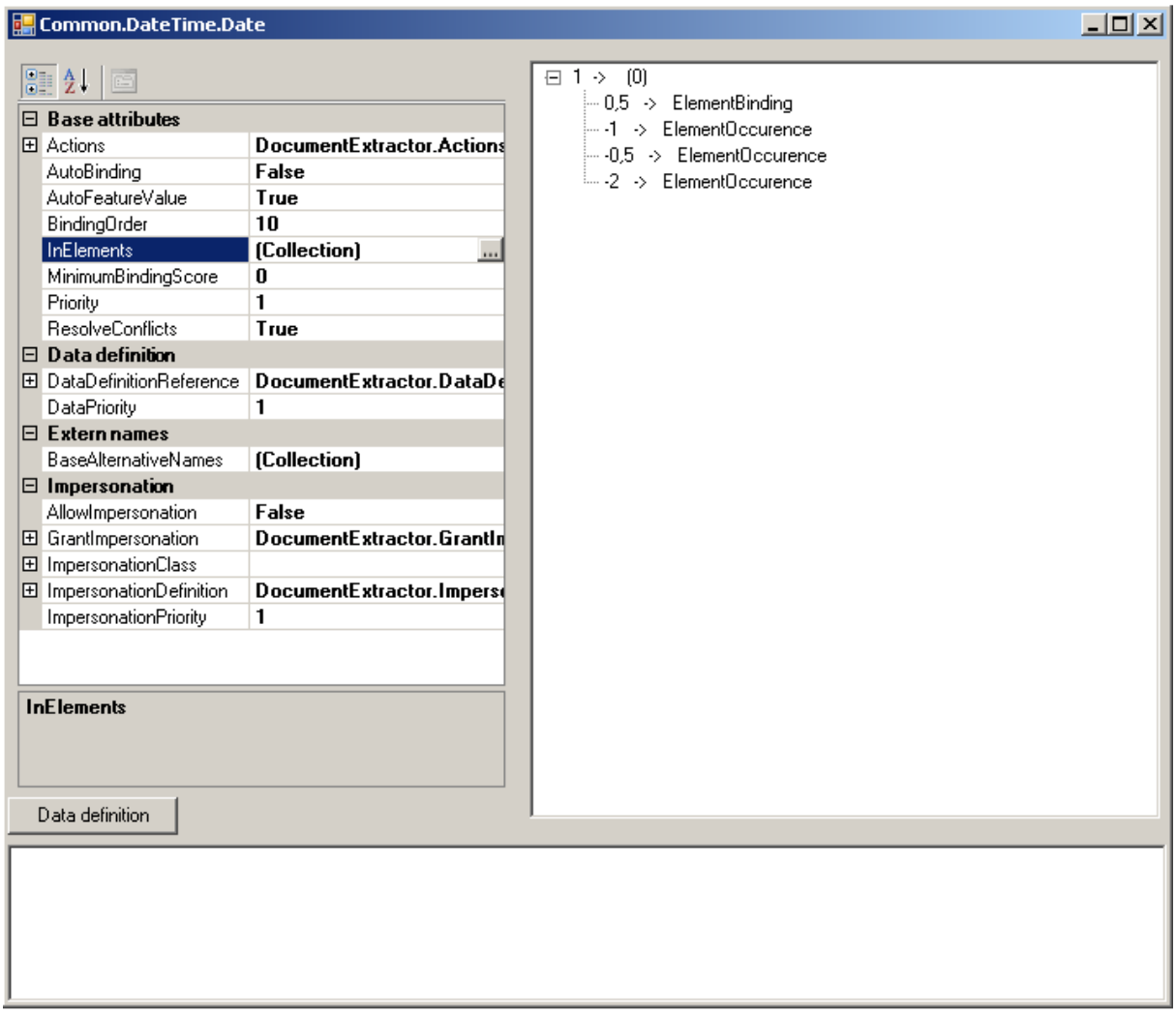
Obrázek 4.3 – okno pro výběr vzorů

Toto okno umožňuje vytvářet vzory, nebo editovat stávající. Jak je vidět, vzory jsou uspořádané ve stromě (jmenné prostory). Je nesmírně důležité si správně rozvhnout vzory do stromů, protože to umožní daleko snazší úpravy. Systém v současné době sice neumožňuje přejmenovat vzory, ale v příští verzi by to tam určitě mělo být. Proto je důležité umístění ve jemnných prostorech pečlivě zadat.

Ve své nejjednodužší podobě stačí dvakrát kliknout na koncový element a zobrazí se okno pro editaci jednoho elementu. To někdy ale nestačí, protože často chceme zeditovat několik základních vlastností naráz pro skupiny elementů – například pořadí vázání (Binding order), nebo minimální vázací ohodnocení (minimum binding threshold), všem elementům přiřadit (ubrat) jeden alternativní název, upravit práh hlavního neuronu, nebo přidat podmínku, která bude platit pro všechny elementy.

Pokud je zapnutá volba multiselection, lze dvojklikem na jmenný prostor vybrat všechny elementy, které patří pod vybraný jmenný prostor. Potom lze nad všemi elementy dělat operace, které byly stručně popsané výše.

Editace vzoru

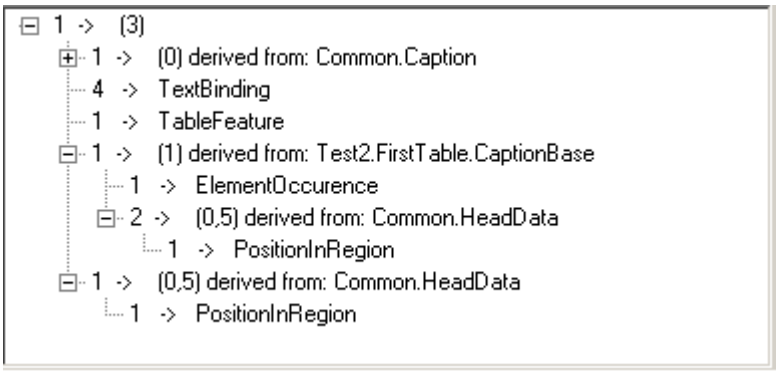


Obrázek 4.4 – editace vzoru

V tomto okně se dá nastavit vše ohledně vzoru, kromě nastavení typu barva apod – to se nastavuje v části nastavení aplikace. Editace vzoru obsahuje pouze informace nutné k extrakci a k převodu do dat.

Obrazovka je rozdělena na 3 části. Nalevo se dají editovat veškeré parametry vzoru a napravo se nachází strom podmínek. Spodní část je vyhrazena chybovému oknu. V určitých případech je uživatel upozorněn na nedostatky v definici – například takový nedostatek, kdy je element vázán na jiný element, který má ale vyšší pořadí vázání – při rozpoznání by v době navázání takový element ještě neexistoval.

Strom podmínek umožňuje upravovat podmínky, přidávat apod. Umí ale také navázat podmínku na zdrojovou podmínku v jiném elementu, například:



Obrázek 4.5 – ukázka stromu rysů

Zde je vidět, že se jedná o zanořené podmínky odvozené od jiných vzorů. To umožňuje znovupoužitelnost a udržovatelnost všech vzorů. Je dobré podobné elementy seskupovat do jednoho jmenného prostoru a přiřadit jim všem odkaz na jiný bázový element, který udržuje všechny společné vlastnosti.

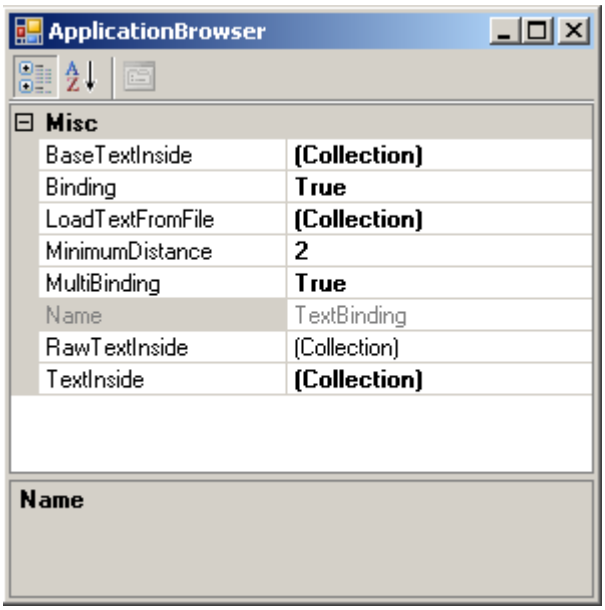
4.11.2 Uživatelské rozhraní pro editaci rysů (vlastností)

Rysů se vyskytuje v projektu veliké množství – v současné době asi 23. Vlastnosti musí být editovatelné pomocí uživatelského rozhraní a také pomocí XML – pro účely otevřenosti a persistence. Navíc musí být možné (i když to není v diplomové práci plně implementované) vytvářet vlastní uživatelské rysy – pro uživatelské rozhraní to funguje, ale bohužel to v současné verzi nefunguje kvůli nedokonalé XML serializaci v .NET, bylo by třeba vyvinout vlastní serializaci.

Uživatelské rozhraní není zcela snadný úkol, protože postavit ho pro 23 rysů je zcela mimo časové možnosti. Z tohoto důvodu je třeba mít k dispozici nástroj pro **automatické generování GUI**.

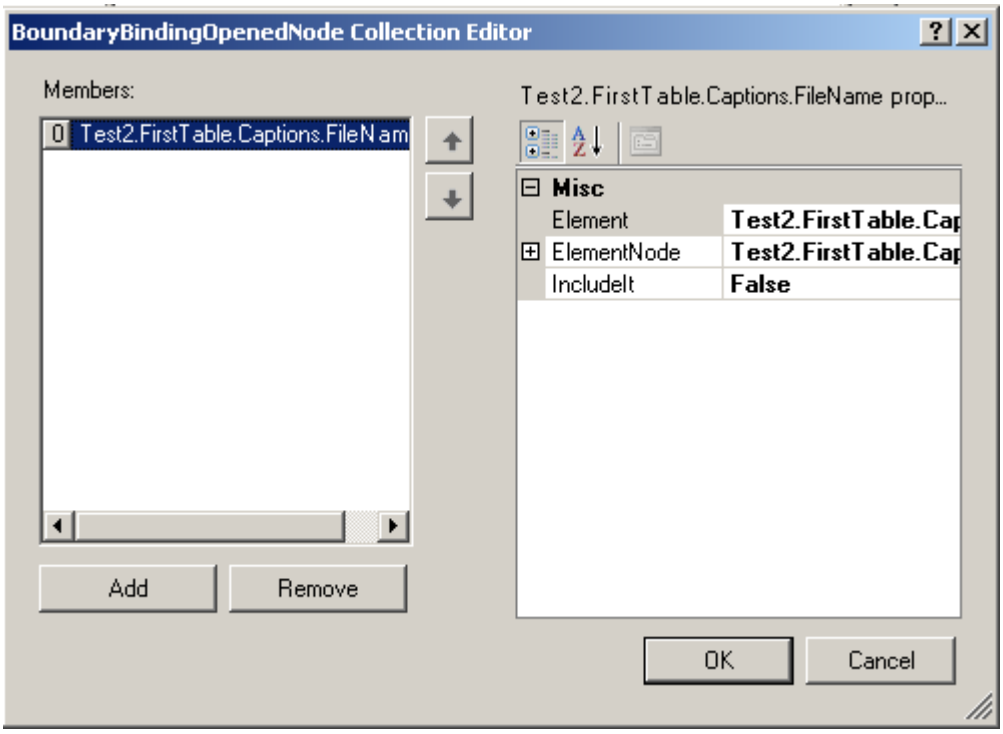
Je pravda, že takto generované uživatelské rozhraní není zcela optimalizované a v některých případech uživatelsky nepřívětivé, ale na druhou stranu umožňuje okamžitě reagovat na změny v definici vlastností a velice snadno přidávat nové, což je při vývoji nepostradatelné.

Automatické generování GUI probíhá na základě definice třídy a tzv. atributů. Pomocí reflexe se pak za běhu zjišťují datové typy, různá omezení, výjimky (na určité věci se třeba postaví speciální dialog) atd.



Obrázek 4.6 – okno prohlížeče objektů

Okno prohlížeče zobrazuje všechny vlastnosti (properties), pokud uživatel neřekne jinak pomocí atributů (např. [Browsable(false)]. Dále zjišťuje zda se jedná o základní datové typy, které se dají editovat přímo, objektové typy (které můžou, ale nemusí mít vlastní editor), nebo o kolekce.



Obrázek 4.7 – prohlížení kolekcí

Zde je vidět editor kolekcí, který spravuje kolekci stejného typu. Pokud uživatel vytváří vlastní rys, stačí, aby se řídil určitými omezeními co se návrhu objektu týče. Za to bude odměněn automaticky vygenerovaným rozhraním. Řešení bylo inspirované [12].

XML Serializace

Xml Serializace opět musí být automatická. Není možné, aby se nad každým objektem rysu sestavoval speciální kód pro zápis a načítání XML. Systém má k dispozici metody SaveXml a LoadXml, který umí načíst i zapsat téměř jakýkoliv objekt, který splňuje určité podmínky. Obsahuje také 2 callback funkce BeforeSerialize a AfterSerialize, které si daná třída může doiplementovat a připravit se na serializaci – někdy je totiž třeba určité parametry dopočítat až za běhu, ty se ukládat nebudou.

Xml Serializace v .NET má však jednu velkou nevýhodu – rysy jsou odvozené od bázové třídy a ta umí poznat konkrétní třídu pomocí atributu class, jenže aby ji serializace využila, potřebuje k tomu Atribut XmlInclude, který se v současné verzi .NET nedá dynamicky přidat. Takže pokud chceme doplnit jiný vzor, nelze to udělat pomocí externí knihovny.

5 Experimenty

Experimenty probíhaly na 65 dokumentech, které si byly podobné. Ne však doslova, byla spousta věcí, která se musela nadefinovat jinak, ale cílem experimentu bylo předvést znovupoužitelnost a obecnost některých definic. V Experimentech jsou diskutovány vzniklé problémy a jejich řešení společně s přibližnou přesností extrakce – není v časových možnostech všechny dokumenty označkovat a přesně zhodnotit. Přesné hodnoty nejsou navíc u ručního klasifikátoru podstatné. Důležité je zhodnotit chybně klasifikovaná data a případně navrhnout řešení. Úspěšnost je tedy jakýsi neformální slovní popis precision a recall.

5.1 Řešení obecných problémů

5.1.1 Agendy

V sadě dokumentů, která byla k dispozici, se agendy příliš nelišily, vždy se jednalo a časový údaj (zejména rozsah) a k tomu příslušný popis. Pokaždé to však bylo zapsáno v jiném formátu, někdy v tabulce, někdy ne. Dobrá možnost vyzkoušet možnosti extrakčního nástroje.
Příklady záznamů:

27.3.2009

Tabulka 5.1 - ukázka

morning: KiWi systém			
09:00	09:45	Welcome & A Look Back (Sebastian)	Sebastian
09:45	10:30	KiWi user experience design, demo of current KiWi system (Szaby, Sebastian)	Szaby
10:30	11:00	coffee break	
11:00	11:45	KiWi data model and architecture (Rolf)	Rolf
11:45	12:30	KiWi transaction management and versioning (Steffi)	Steffi

Tabulka 5.2- ukázka

morning: KiWi systém		
09:00-09:45		Welcome & A Look Back (Sebastian)
09:45-10:30		KiWi user experience design, demo of current KiWi system (Szaby, Sebastian)
10:30	11:00	coffee break
11:00-11:45 KiWi data model and architecture (Rolf)		
11:45	12:30	KiWi transaction management and versioning (Steffi)

28.3.2009

9:00-9:45

Welcome and Look Back (Sebastian)

9:00-9:45

Welcome and Look Back

Sebastian

9:00

Welcome and Look Back

Sebastian

Tabulka 5.3- ukázka

9:00	**9:15**	**Welcome**	**0:15**
9:15	**9:35**	**Project status (coordinator)**	**0:20**
9:35	**10:45**	**WP presentations (WP1, WP2, WP3, WP4, WP5, WP8) 10' each (work performed – next actions)**	**1:10**
10:45	**11:15**	**Coffee break**	**0:30**
11:15	**12:45**	**WP7 plenary and D7.2**	**1:30**
12:45	**13:00**	**Architecture services overview and common UI (or any design guidelines) for the whole project**	**0:15**
13:00	**14:00**	**Lunch break**	**1:00**
14:00	**14:45**	**Architecture services overview and common UI (or any design guidelines) for the whole project**	**0:45**

14:45	**16:00**	**Discussion on the questions of PO**	**1:15**
16:00	**16:30**	**Coffee break**	**0:30**

Tabulka 5.4- ukázka

October 3, 2008		
9:00 a.m.	**Call to Order**	**Tab 1**
	NE Open Meetings Law	
	Roll Call – Planting Report	
	Introduction of Guests	
	Review/Adopt agenda	
	Review of minutes	
	Financial Report	
	Compliance Reports	
9:30 a.m.	**Budget**	
	Review and Update	

Kromě těchto údajů se ještě objevují údaje o Datu, které položkám přísluší. Jak je vidět, záznamy mají různorodý vzhled, někdy jsou v tabulce, někdy ne. V tabulce mohou být na řádku, nebo také na více řádcích.

ItemTime (definice času)

Čas může být obyčejný čas, či rozsah času. Proto je zde ElementBinding nastaven na čas či rozsah. ItemTime má dále rys Visibility, který je velice důležitý, protože se předpokládá, že takto důležitá položka bude dostatečně viditelná. Dále zvyšuje ohodnocení i přítomnost elementu EndOfLine, či EndOfTableRow nalevo, který říká, že časové údaje uvedené na levé straně dokumentu mají vyšší šanci stát se ItemTime. Toto jsou základní podmínky.

HeadLineDate (definice data)

Datum určuje datum záznamu, je obvykle uvedené samostatně a všechny záznamy pod ním (po nové datum) náleží jemu.

ItemRecord (definice záznamu agendy)

ItemRecord je na nadefinování již o něco složitější. Základní vázací podmínka obsahuje ElementBoundaryBinding, který má počáteční element právě ItemTime a koncový, některý z těchto:

ItemTime Je jasné, že jakmile narazíme na nový čas, jedná se o nový záznam.

EndOfDocument	Pokud záznam končí koncem dokumentu, je třeba ho označit za platný.
HeadLineDate	Záznam končí datem.
EndOfTable	Konec tabulky v naprosté většině případů ukončuje ItemRecord.
Caption	Jakýkoliv nalezený popisek (či nadpis) ukončuje záznam (pokud bychom chtěli definovat popisky uvnitř ItemRecord, musíme to udělat až po navázání ItemRecord).

Tato definice byla odzkoušena na spoustě dokumentů a funguje dobře. Co když ale budeme řešit následující problémy:

Některé ItemRecord jsou pro moji sadu dokumentů neplatné.
Je třeba je překrýt elementem platným pro tuto sadu, který se naváže na ItemRecord, otestuje dané podmínky a rozhodne se zda ItemRecord smazat či ne.

Současná definice ItemTime mi nevyhovuje, ale nechci kvůli tomu předělávat původní definici ani dělat nový ItemRecord.
Je možné vytvořit vlastní definici ItemTime a nastavit alternativní název pro původní ItemTime, nový ItemTime se bude pro ostatní elementy jevit jako starý ItemTime.

Ve svých dokumentech mám uvedené i jiné časy, které nesouvisí s ItemTime, extraktor mi je přesto jako ItemTime označí.
Je třeba definovat dokumentově specifické kontrolní elementy ItemTime navázané na původní ItemTime, otestovat a pokud splní podmínky, původní ItemTime smazat.

5.1.2 Hlavičková data

Příklady:

Date:	23/06/2008
Author(s):	Michael Ovelgönne
Organisation(s):	EM-KA
WP/Task/Subtask:	WP 4
Version:	01

Takto vypadá spousta hlavičkových dat. Někdy jsou v přehledné tabulce, jindy pouze ve formě popisků a dat odděné tabelátorem. Jak to vyřešit? Například definovat oblast, ve které se data nachází. Počáteční element bude daný popisek (date, author…) a koncový element jakýkoliv popisek (Caption), či konec řádku tabulky. Tím nejsme vázáni na pořadí elementů ani striktně na tabulku.

Rozpoznání popisku

Popisek se rozpozná jednak podle obsahu textu a jednak podle viditelnosti. Někdy může pomoci o přítomnost okolních popisků, víme-li, že se často nacházejí spolu, můžeme přiřadit vyšší ohodnocení při jejich přítomnosti. Abychom v každé definici nemuseli tyto popisky definovat znovu a znovu, je dobré pro určitou sadu popisků nadefinovat bázový popisek, který bude obsahovat podmínky (vizuální elementy, výskyt ostatních popisků v sadě atd…) a konkrétní popisky budou mít pouze TextBinding (každý se nazývá jinak) a odkaz na bázový popisek, tím všichni zdědí jeho rysy.

Rozpoznání konce datové oblasti

Bývá problematické. Pokud se za datovou oblastí vyskytuje nějaký nadpis, jako ve většině dokumentů, není s tím žádný problém. Ale někdy se za datovou oblastí vyskytuje otevřený text – ve vlastním odstavci. Zde není jednoduché říci, kdy to prohlásit za konec, nebo za pokračování. Někdy může pomoci uživatel pomocí uživatelských elementů, kdy ručně řekne: Tady končí oblast dat, jindy může dopsat ručně nějaký popisek a nadefinovat pro něj element a doplnit ho do všech dokumentů a poslední možností je vyklasifikovat odstavec otevřeného textu jako konec datové oblasti – například mu přiřadit alternativní jméno Common.Caption (což je trochu matoucí, lepší by bylo něco jako Common.EndOfData, ale to by bylo potřeba do koncové podmínky dat tento název přidat). Odstavec s otevřeným textem není takový problém odhalit – na to slouží rys StopWordsDensity. Bylo ověřeno, že mezi daty v tabulkách (například viz. nahoře) a mezi volným textem je velký rozdíl v počtu Stop slov. Problém by mohl být snad jen v datech s otevřeným textem. Ale to už by si musel vyřešit uživatel sám – například i dodatečnou kontrolou dat.

5.1.3 Tabulková data

O těch se zde zmíníme jen velice krátce, protože se dají rozpoznávat s přesností 95-100%. Většina dat byla však zapsaná někdy v tabulce, někdy mimo tabulku, takže byla potřeba volnější definice. Ale data, která byla čistě v tabulce, byla velice přesná a celkem snadná na popis.

5.2 Nebraska Wheat Agenda

8 dokumentů

Typ: Časový seznam s událostmi a osobami v nich zainteresovanými.

Znovupoužitelnost: Téměř všechny předchozí definice

Úspěšnost: Vysoká

V první fázi asi 90%, po úpravách které trvaly asi 10 minut vzrostla asi na 95%. Do úspěšnosti se nezapočítaly nenalezené osoby Person, protože zde záleží na velikosti databáze jmen. Většina osob ale byla nalezena s přibližnou úspěšností 95%.

Problémy a řešení:

Chybí HeadLineDate

U některých položek chybí HeadLineDate - není totiž uvedeno a tím pádem se vztahuje k MainDate. Otázka zní, zda je to dobře, nebo špatně, to záleží už na uživateli. Není problém MainDate označit i jako HeadLineDate, nebo tuto informaci zohlednit při převodu do databáze...

Wednesday, March 10 (Working Lunch Provided)

Někdy je HeadLineDate nenalezeno.

Problém: Mezi datem a následnýma položkama ItemTime je text, snižuje to vidtelnost data.

Možné řešení: Posílit u HeadLineDate vliv tučnosti písma.

March 16 & 17, 2009

Problém: Rozpoznáno pouze March 16

Možné řešení: Povolit znak & jako spojovací text pro rozsah

Thursday, June 11, 2009
1:00 p.m. **Call to Order**

Problém: Nebyl rozpoznán čas.

Možné řešení: Ukázalo se, že na konci data nebyl žádný konec řádku, ani konec odstavce, což je trochu záhadné. Stačilo konec odstavce ručně vytvořit a problém byl vyřešen.

Problémy se záznamy na více řádků v tabulce

Problém: Některé údaje byly v tabulce, přičemž neplatilo pravidlo, že jeden záznam je na jednom řádku.

Možné řešení: Vzhledem k tomu, že tabulková data jsou uspořádaná docela přesně, je možné podmínku konce řádku u ItemRecord vypustit.

Ukončení posledních záznamů ItemRecord

Problém: Ukončovací podmínky pro ItemTime se u poslední nemusí vyhodnotit správně, jelikož ukončovací podmínka je nastavena na ItemTime, HeadLineDate, EndOfTable a CommonCaption.

Možné řešení: Automatické vyhledávání nadpisů tento problém často vyřeší, ale je výpočetně velice náročné, protože musí testovat u každého dostatečně krátkého nadpisu jeho informace o fontu. To trvá přes COM rozhraní dlouho. Horší je to, když seznam ItemRecord nekončí nadpisem, ale nějakým souvislým textem, který bez vizuálních informací, které Word nerad poskytuje, nelze rozpoznat.

Přítomnost časové délky u záznamu

Problém: Časová délka občas může být označena jako ItemTime, přičemž patří pod jeden záznam společně s časovým rozsahem.

Možné řešení: Vytvořit speciální element ItemLength, který bude definován v závislosti na sadě dokumentů a naváže se na ItemTime, vyhodnotí se zda se nejedná spíše o ItemLength a pokud ano (test na MinimumBindingScore), ItemTime se na tomto místě smaže.

Druhá možnost je změnit přímo ItemTime, ale to může poškodit jiné extrakce.

Některé osoby v záznamech nebyly rozpoznány

Tento problém v zásadě nemá řešení, vše závisí na velikosti databáze jmen a na dalších dedukčních schopnostech Named Entity Recognition.

V některých dokumentech to rozpoznalo seznamy osob (například Attendants apod)

Problém: Je možné, že se občas vyskytnul tento text, který byl dobře vidět a byl považován za popisek osob.

Možné řešení: Zpřesnit definici datových popisků by bylo velice pracné a mohlo by to poškodit hotové funkční definice. Lepší je tyto definice pro tuto sadu dokumentů ignorovat a vyškrtnout ze seznamu vzorů. Protože seznamy osob ani jeden dokument obsahovat nemá.

Někdy je ItemRecord záhadně ukončen

Jedná se o častý problém u této sady dokumentů. Tento problém ovšem není chyba systému, nýbrž chybné formátování dokumentu, protože některé dokumenty mají ukočenou tabulku v půlce ItemRecord a dále výpis pokračuje mimo tabulku. Vizuálně není nic poznat, protože tabulka má vypnuté okraje a text pod ní je odsazen tak, aby lícoval. Řešení tohoto problému by mělo spíš být na straně tvůrce dokumentu – takto se dokumenty nepíší. Uživatel může data opravit ručně – označí co považuje za ItemRecord a co ne.

5.3 Nebraska Wheat Meeting

14 dokumentů

Typ: Tabulková data v hlavičce, seznamy osob + poznámky k nim.

Znovupoužitelnost: Většina definic nových – kvůli různým typům osob.

Úspěšnost: Vysoká. 90-100%.

Problémy a řešení:

Různé definice seznamů osob pro různé dokumenty

Problém:

Jednou byla data v klasické formě:

Název role…seznam osob (na každém řádku jedna)

Další název…opět seznam osob.

Podruhé byla v tabulkové formě, vypadalo to asi takto:

	Meeting Minutes
	Nebraska Wheat Board
	March 16 & 17, 2009
	Holiday Inn, Lincoln, NE
Board Members Present	**Guests**
Dan Hughes, Venango – Chairperson	Scott Osler, President NWGA
Chris Cullan, Hemingford	Paul Johnson
Von Johnson, Cambridge	Ken Anderson, Brownfield Network
Rick Larson, Potter	Ashley Colglazier
Brent Robertson, Elsie	Mike Keenan
Delferd Schlake, Blue Springs	Mark Hodges, PGI
	Roger Berry, A-FAN
Ex-Officio Member Present	Joe Parsons, USDA
Dan Duncan, UNL	Dr. Gary Hein, UNL
	Dr. Jens Walter, UNL
Staff Present	Terry Hejny, Nebraska LEAD
Royce Schaneman, Executive Director	Michelle Weber, NE Attorney General Office
Zoe Olson, Public Information Officer	Ken Rahjes, KRVN
Ann Bauers, Staff Assistant	Jeff Noel, Husker Genetics
Cody Dvorak, Intern	Pat Nelson

Z tabulky není problém tato data vyextrahovat. Problém může nastat při spojování osob a jejich rolí – nemůžeme už použít NearestBefore, protože role nejsou už ve směru toku textu, ale ve sloupcích.

Řešení:

Datová položka obsahuje tři informace: Vnitří text (včetně poznámek), jméno osob v záznamu a role osob (Staff Present, Guests…).

Definice pro nadpisy bude stejná. Rozdíl je pouze v tom, jaká pravidla platí pro jednu položku záznamu. Jednou bude mezi nadpisy a jednou bude ve sloupcích pod nadpisy v tabulce. Nadefinovala

se položka AttendantsBlock, která se navázala na obecný element TextBlock (který má význam textového bloku odděleného koncem odstavce, řádku, tabelátoru apod.). AttendantsBlock se vyhodnocuje podle přítomnosti osoby, přítomnosti řídících nadpisů (Staff Present, Guests…) atd.
Dále se vytvořila položka AttendantsBlockCell, která překryla AttendantsBlock a pokud se dotyčný blok nachází v tabulce, AttendantsBlock byl smazán. Je to z prostého důvodu Attendants block vyhodnocuje svojí roli (Guest, Staff Present) od nejblížšího popisku vlevo ve směru toku textu. Ale v tabulce hledáme nejblížší element ve sloupcích, nejbližší element vlevo by dal špatný výsledek, například Terry Hejny by byl pod rolí Staff Present, což je špatně. Oba elementy mají teda stejný klíč definice dat, dokonce i tvar výsledného záznamu. Ale každý si svá data převede jinak. Výsledek vede k tomu, že datová sada je pro oba typy dokumentů stejná.

5.4 Agenda (Kiwi)

9 dokumentů

Typ: Časový seznam s událostmi a osobami v nich zainteresovanými. Někdy v tabulce, někdy mimo tabulku.

Znovupoužitelnost: Téměř všechny předchozí definice.

Úspěšnost: Vysoká, . 90-100%.

Zhodnocení:

V časových záznamech jsou některé dodatečné údaje (například informace zda jde plenary session, nebo parallel session). Pokud je uživatel chce mít v datech, lze to udělat následovně:
Vytvořit nový ItemRecord, navázat na původní ItemRecord a ten původní smazat. Následně vytvořit vlastní definici převodu do dat.
Další možnost je upravit přímo zdrojovou definici dat, což fungovat určitě bude a ostatní dokumenty to neovlivní, na druhou stranu jde o zásah do obecných definic.

5.5 Meeting (Kiwi)

34 dokumentů

Typ: Tabulková data, údaje o dokumentu a účastnících

Znovupoužitelnost: Vlastní definice

Úspěšnost: Téměř 100%.

5.6 Montreal Meeting Agenda

1 dokument

Typ: Klasická agenda

Znovupoužitelnost: Vysoká

Úspěšnost: Vysoká, . 95-100%.

Jediný problém, který mohl nastat, bylo ukončení seznamu agendy. To šlo snadno vyřešit označením nadpisů, které tuto oblast ukončovaly.

6 Závěr

Podřilo se vyvinout systém, který dokáže extrahovat data ze strukturovaných až polostrukturovaných dokumentů, které nejsou generované počítačem. Pro dobře strukturované informace (zejména tabulková data) se úspěšnost blíží ke 100% - podle očekávání.

Pro méně strukturovaná data je úspěšnost kolem 90-97%. Zde záleží hodně na velikosti databáze jmenných entit a na přesnosti rozpoznání data a času či jiných položek. V předchozích tabulkových datech záleželo na správném nalezení tabulkových popisků, v případě méně strukturovaných dat je velice důležité najít záchytné body a správně rozdělit data do oblastí. Problém nastane zejména tehdy, vyskytuje-li se v dokumentech velké množství „matoucích“ záchytných bodů – například hledáme-li čas (či časový rozsah) pro určení začátku záznamu agendy a uvnitř záznamu se opět nachází časy a dobře viditelné. Může dojít k chybnému rozdělení oblastí.

Nedostatky systému jsou zejména v absenci vizuálních informací, které rozhraní COM neposkytuje. Tudíž některé situace nelze řešit jednoduchými prostředky a složité prostředky by byly pro uživatele příliš komplikované a ne zcela přesné.

Dalším nedostatkem je generované uživatelské rozhraní, které sice umožňuje rychlý vývoj, ale není zcela přívětivé pro uživatele. Tento nedostatek je však zcela pochopitelný, protože kompletní uživatelské rozhraní je mimo časové možnosti práce.

Systém byl navržen na dobře strukturovaná data psaná člověkem a tento účel splnil. Pokud má uživatel k dispozici velké množství dokumentů s dobrou podobnou strukturou, je možné data extrahovat s velmi vysokou přesností. Systém ale nebyl testován na špatně formátovaná data. Na ty by bylo třeba dodatečného uživatelského kódu s podporou standardních nástrojů systému pro ulehčení práce.

Literatura

[1] Sunita Savaragi: Information Extraction, Foundations and Trends in Databases, Vol. 1, No. 3 (2007) 261–377, 2008 S. Sarawagi, DOI: 10.1561/1500000003

[2] Ralph Grishman: Information Extraction: Techniques and challenges, Computer Sience Department, New York University, New York, NY 10003, U.S.A

[3] Christopher D. Manning, Prabhakar Raghavan, , Hinrich Schütze: An Introduction to Information Retrieval, Cambridge Univerzity Press, Cambridge, England, 2009

[4] David Nadeau, Satoshi Sekine: A survey of named entity recognition and classification, National Research Council Canada / New York University

[5] Domovská stránka MSDN, [cit. 20.5.2010], Dostupné na msdn.microsoft.com

[6] Sláma Vojtěch: Extrakce dat z popisu zboží. Brno, 2008, diplomová práce, FIT VUT v Brně.

[7] Marek Schmidt: Extrakce sémantických vztahů z textu, diplomová práce, Brno, FIT VUT v Brně, 2008

[8] Jakub Oravec: Klasifikace dokumentů; podle tématu, bakalářská práce, Brno, FIT VUT v Brně, 2008

[9] Kraus Michal: Zjednoznačňování slovních významů. Brno, 2008, diplomová práce, FIT VUT v Brně

[10] Ing. Vladimír Bartík, Ph. D., Dolování z textu a na webu, slidy k předmětu Získávání znalostí z databází, FIT VUT Brno

[11] English StopWords, [cit. 20.5.2010], Dostupné na: http://www.ranks.nl/resources/stopwords.html

[12] Tutorials and resources for Microsoft Property Grid, [cit. 20.5.2010], dostupné na: http://www.propertygridresourcelist.com/