VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

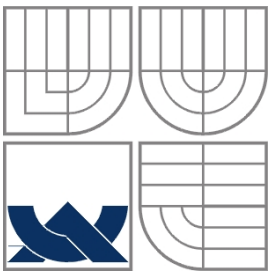
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KNIHOVNA PRO PODPORU VÝVOJE SYSTÉMU RERESEARCH

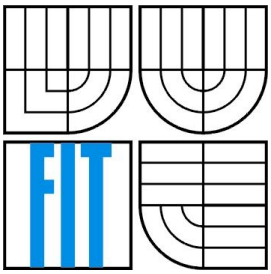
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE STANISLAV HELLER
AUTHOR

BRNO 2011



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

KNIHOVNA PRO PODPORU VÝVOJE SYSTÉMU RERESEARCH

LIBRARY FOR SUPPORT OF RERESEARCH SYSTEM DEVELOPMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

STANISLAV HELLER

VEDOUCÍ PRÁCE
SUPERVISOR

ING. SVATOPLUK ŠPERKA

BRNO 2011

Abstrakt

Vývoj systému ReReSearch je značně zpomalen vzájemnou nekompatibilitou jednotlivých modulů z hlediska programových prostředků, faktem, že vývojáři často opakují již známé chyby a v neposlední řadě slabou komunikací vývojářů mezi sebou. Pro řešení uvedených problémů bylo zapotřebí vytvořit prvek, který by implementoval časté úkony a procesy, sjednotil některá rozhraní v systému a poskytl prostředky pro řešení problémů na vyšší úrovni abstrakce. Výsledkem této snahy je knihovna rrslib, která by měla sloužit všem, kteří se věnují vývoji systému ReReSearch: práci s jeho databází, extrakci, zpracování, analýze a indexaci dat z webu a z lokálních dokumentů nebo jiným částem systému ReReSearch. Používáním knihovny by mělo být docíleno konzistentnějšího, rychlejšího a méně chybového vývoje systému ReReSearch.

Abstract

At this time, the development of the ReReSearch system is significantly slowed down by mutual incompatibility of system modules, by the fact that developers often repeat already known mistakes and of course by poor communication between developers in general. To solve this problem, there was a need to create a component which would implement and unify often performed tasks in development of ReReSearch system and this way to spend time of ReReSearch developers. The result of this effort is so-called “rrslib” – a Python library, which is supposed to be a helper for everyone, who works on parts of ReReSearch project: database, data extractors, web-based agents, crawlers, XML-processing etc. The library should serve for more consistent, faster and more reliable development of ReReSearch system.

Klíčová slova

ReReSearch, Python, knihovna, aplikačně-programové rozhraní, získávání informací, dolování dat, zpracování přirozeného jazyka, skriptování, softwarové inženýrství, vývoj, internet.

Keywords

ReReSearch, Python, library, API, information retrieval, data mining, natural language processing, scripting, software engineering, development, internet.

Citace

HELLER, Stanislav. *Knihovna pro podporu vývoje systému ReReSearch*, bakalářská práce, Brno, FIT VUT v Brně, 2011.

Knihovna pro podporu vývoje systému ReReSearch

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Svatopluka Šperky.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Stanislav Heller
V Brně 15.května 2011

Poděkování

Tímto bych chtěl poděkovat Ing. Šperkovi a Ing. Otrusinovi za odbornou pomoc, kterou mi poskytli při řešení této práce. Díky také patří Bc. Tomáši Lokajovi, který mi pomohl s implementací některých částí knihovny.

Obsah

1 Úvod

V dnešní době, kdy se věda velmi rychle rozvíjí a vědecké disciplíny se stále více specializují, bývá pro začínající studenty a vědce velmi obtížné zorientovat se v problematice a udržet krok s vývojem daného vědeckého oboru. Pro získání kvalitních informací o trendech a moderních postupech lze využít existujících citačních databází, které obsahují velké množství publikací z různých vědních oborů. Tyto databáze však poskytují často pouze základní data o publikacích bez potřebného propojení se souvisejícími vědními obory, postrádají možnost poskytnutí komplexní informace a celkového přehledu o dění v požadované disciplíně.

Uvedenou problematikou se zabývá projekt *ReReSearch* (dále jen RRS), jehož cílem je vybudování znalostního systému, který by byl orientován nejen na sběr a prezentaci dat, ale i na hlubší analýzu, zkoumání souvislostí a vývoje v daném vědeckém odvětví. Systém ReReSearch je vyvíjen ve výzkumné skupině zpracování přirozeného jazyka (NLP Group) na Fakultě Informačních Technologií na Vysokém Učení Technickém v Brně.

Během vývoje systému RRS vznikla potřeba prvku, který by sjednocoval některé úkony a procesy, vytvářel jednotná rozhraní a usnadnil vývoj některých částí systému. Tato potřeba byla ještě umocněna faktem, že vývojáři jsou často z řad studentů nižších ročníků FIT, pro které obtížné obsáhnout šíři problému v krátkém čase, který mají na seznámení s projektem. Proto byl vývoj částí RRS pomalý, nekonzistentní, použitelnost některých skriptů či programů byla na nízké úrovni a kvůli špatnému návrhu některé moduly rychle zastarávaly.

Východiskem tohoto problému je sjednocující pomocný prvek – programová knihovna. Protože naprostá většina modulů systému RRS je implementována v jazyce Python, musí i knihovna poskytovat aplikačně-programové rozhraní v tomto jazyce. Vývoj knihovny *rrslib* byl započat v květnu 2010. Nejdříve měla být knihovna určena pouze pro extraktory[1], později byla podpora rozšířena o databázi, klasifikaci dat, slovníky a formát RRS-XML.

Cílem této práce je analýza opakujících se úkonů v rámci modulů RRS, identifikace algoritmů vykazujících vysokou chybovost, jejich zobecnění a abstrakce. Dalším cílem je kvalitní návrh a implementace knihovny rrslib, která bude implementovat zjištěné úkony a algoritmy a která bude podporovat rychlejší a konzistentnější vývoj dalších modulů systému ReReSearch. Knihovna musí splňovat požadavky na spolehlivost, rozšiřitelnost a musí být kvalitně otestována a zdokumentována. Ideálním konečným stavem je vybudování frameworku pro výstavbu všech typů modulů systému RRS. Vzniklou knihovnu bude pak třeba integrovat do prostředí RRS a využívat poskytovaných rozhraní a podpory v modulech systému RRS.

V následujících kapitole bude popsán stav systému RRS, budou analyzovány některé problémy při jeho vývoji. V dalších kapitolách bude popsáno řešení některých uvedených problémů - návrh knihovny rrslib, její implementace, testování a další možný vývoj.

1 moduly systému ReReSearch, které se zabývají extrakcí dat z webu a z lokálních dokumentů

2 Analýza problému

Tato kapitola pojednává o systému ReReSearch, o problematice jeho vývoje a popisuje okolnosti rozhodování o vzniku knihoven pro ReReSearch. Je zde popsán proces identifikace nejvíce rozšířených a chybových úkonů a obecné zásady při návrhu knihoven a aplikačně-programového rozhraní.

2.1 Systém ReReSearch

Jak již bylo v úvodu této práce řečeno, cílem systému ReReSearch je vybudování znalostního systému, jehož základem jsou vědecké články a publikace, které slouží jako hlavní zdroj informací o dění na akademické půdě. Principem funkce systému je sběr dat o publikacích, autorech, organizacích, projektech atp., budování znalostí nad získanými daty a interaktivní prezentace informací uživateli. Systém by měl být schopen informovat vědce o nových publikacích, konferencích, seminářích a událostech v jeho oboru i v oborech souvisejících, měl by také umožnit sledovat aktivitu a cíle významných osobností zabývajících se danou problematikou a propojit tyto informace s aktuálním děním v oboru. Cílem projektu je tedy poskytnout prostředí pro rychlejší orientaci v oboru výzkumníka.

V současné době existuje několik kvalitních webových služeb, které se specializují na vědecké publikace a na vědu obecně. Mezi nejkvalitnější zdroje věděckých publikací patří CiteSeerX [1], DBLP [2] nebo Google Scholar [3]. Zaměřením blíže systému ReReSearch je portál Microsoft Academic Search [4], který navíc poskytuje informace o organizacích, které se zabývají výzkumem.

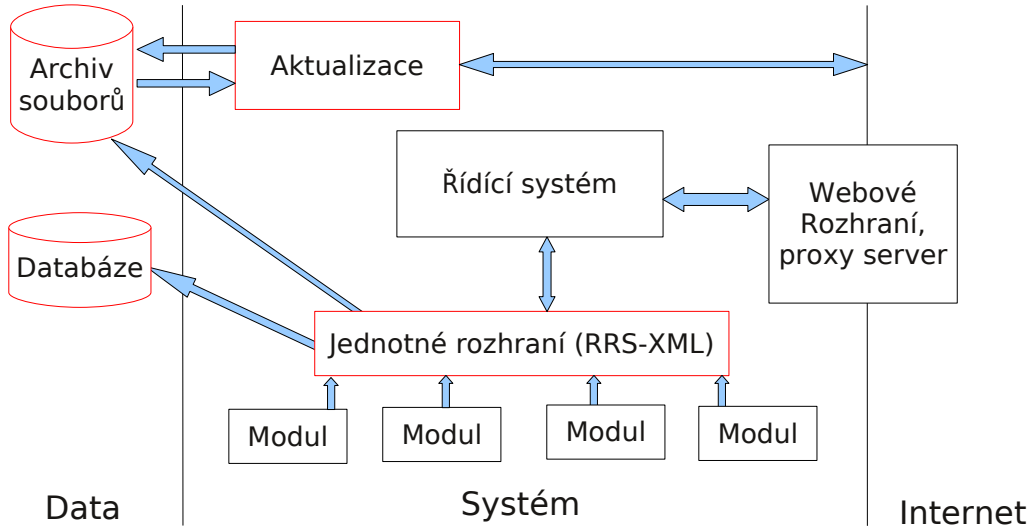
2.1.1 Architektura systému ReReSearch

ReReSearch je navržen jako informační systém postavený na databázi, obsahuje archiv souborů (převážně publikací), a je řízen *řídícím systémem*. Systém obsahuje také aktualizační systém, pomocí kterého bude systém zpracovávat i nejnovější publikace a události, které se na internetu objeví.

Řídící systém je program (démon), který pomocí vzdáleného volání procedur (XML-RPC) spouští *moduly*, obsluhuje jejich činnost, uchovává informace o běhu systému a spouští pravidelné procesy pro údržbu systému. Obsluhuje také požadavky uživatelů a řídí případné chybové stavy a řeší incidenty.

Modul systému RRS je samostatný program nebo skript komunikující s řídícím systémem a který obsluhuje určitou část funkcionality systému (např extrakce dat z článků, převod XML do databáze, extrakce informací o projektech atp.).

ReReSearch obsahuje dvě relační databáze. První z nich je *datová*, ve které jsou uložena veškerá cílová data a informace (publikace, články, osoby atp.). Druhá databáze slouží pro účely *řídícího systému*; jsou zde uchovávána statistická data o chování a konfiguraci modulů RRS, informace o spuštěných úlohách a procesech a také rozšířené informace o průběhu procesů a úloh. Schéma systému je zobrazeno a obrázku 2.1.



Obrázek 2.1 Schéma systému ReReSearch (převzato z [5], obr. 2.11, aktualizováno)

V následujícím textu se budou opakovat některé termíny a proto bude vhodné jejich význam v kontextu systému ReReSearch podrobněji vysvětlit:

- *rozhraní v systému RRS*: tímto rozhraním je nejčastěji *RRS-XML* [5], tedy datový formát XML použitý pro uchovávání extrahovaných informací před vložením do databáze. Dále těmito rozhraními může být myšlen protokol pro komunikaci s řídícím systémem nebo další formáty. Pokud se nejedná obecně o rozhraní, je v textu vždy uvedeno, o jaký formát nebo typ rozhraní se jedná.
- *extraktor*: modul systém RRS, který se zabývá získáváním dat nebo extrakcí informací z webu nebo z lokálních dokumentů.
- *credibility:* parametr dat označující důvěryhodnost informace používaný v systému ReReSearch. Tento údaj vznikl jako reflexe potřeby identifikovat data pocházející z nejistých zdrojů nebo data, která jsou extrahována heuristickými algoritmy, které často nedokáží extrahovat data přesně a spolehlivě.
- *ORM (objektově-relační mapování)*: technika pro napojení objektů reprezentující entity na databázi. Ve většině ORM frameworků platí vztah: 1 objekt = 1 řádek tabulky databáze popř. 1 vztahový objekt nebo asociace = 1 řádek spojovací tabulky.
- *parser:* syntaktický analyzátor. Program nebo část programu, který transformuje vstupní text na hierarchickou strukturu, která je dále využívána pro účely dalšího zpracování.

2.2 Stav vývoje systému ReReSearch

Projekt ReReSearch má za sebou asi 4 roky vývoje a za tuto dobu bylo vytvořeno mnoho modulů a částí. Bohužel, některé tyto moduly jsou v současné době nepoužitelné, protože se prostředí RRS několikrát změnilo a neaktualizované moduly nebo části neodpovídají současným požadavkům.

2.2.1 Obecné problémy

Velmi významným problémem při vývoji RRS je fakt, že studenti, kteří pracují na nějakém modulu nebo kteří se podílí na řešení některé části řídícího systému, často nevidí celý rozsah problému a

nejsou schopni obsáhnout celý kontext, ve kterém se daný problém řeší. Tomuto faktu ještě napomáhá časté střídání studentů na jednotlivých úkolech; nový student se musí seznámit s prostředím, naučit se pracovat s prostředky pro vývoj a aktivním programováním pak stráví zlomek času. Z toho plynou omyly, polovičatá řešení a nefungující části projektů, které pak nemohou být plně implementovány do RRS. Tento problém také zásadně prodlužuje čas řešení projektu a pozdější nasazení modulu do systému.

Nepřehlédnutelnou vlastností systému ReReSearch jsou také časté změny rozhraní, databáze a obecně požadavků na systém. Řešení tohoto problému je nesnadné; částečně jej lze řešit pomocí flexibilních rozhraní a kvalitního návrhu inkriminovaných aplikací tak, aby se dokázaly rychle přizpůsobit novému prostředí. Zásadní část problému ale nespadá do oblasti IT, nýbrž do oblasti managementu a řízení projektu.

2.2.2 Chybovost

Moduly RRS, implementované v rámci obecných projektů, bakalářských nebo diplomových prací, často vykazují vysokou chybovost a nekompatibilitu s okolím systému. Tento problém vychází z nefungující komunikace vývojářů mezi sebou a často chybějící hierarchií odpovědných osob za daný problém. Jestliže mnoho studentů pracuje na stejném úkolu a nespolupracují, zákonitě jsou pak vytvářeny nekonzistentní rozhraní a procesy, jejichž implementace do RRS většinou není možná. V důsledku je pak modul v reálném prostředí nepoužitelný, rychle zastará a případné změny nebo vylepšení znamenají změnu celého algoritmu nebo velké části zdrojového kódu aplikace. Nezřídka se pak stává, že vytvořený modul není nikdy použit a je zcela zahozen.

Mezi hlavní důvody a zdroje chyb v modulech RRS tedy patří:

- neznalost kontextu problému (řešení projektu bez kompletní znalosti okolních procesů)
- nekvalitní návrh řešení
- špatná dekompozice problému na podproblémy
- opakující se chyby, nedostatečná komunikace mezi vývojáři
- nedostatečná flexibilita rozhraní
- nedostatečná znalost implementačního jazyka a jeho chování

Některé z vyjmenovaných problémů by mohla vyřešit programová knihovna obsahující základní rozhraní a algoritmy, které se při implementaci modulů RRS používají nejčastěji. Tím by každý student neřešil problém po svém (a často také špatně), ale existovala by jedna implementace, jejíž zlepšení by se projevilo ve všech modulech, které knihovní funkci používají. Základním úkolem tedy je identifikovat opakující se procesy, analyzovat používaná rozhraní a časté operace nad nimi a tyto algoritmy implementovat do knihovních modulů.

2.2.3 Neexistující nebo nekompatibilní rozhraní

Systém ReReSearch obsahuje několik rozhraní, které spojují jednotlivé části systému. Heterogenita systému na jedné straně usnadňuje implementaci modulu v jiném jazyce, než je Python, ale zároveň přináší mnoho jiných problémů.

Rozhraní v RRS musí být tedy navržena tak, aby byla robustní, odolná proti změnám, flexibilní a promyšlená z hlediska dalších potřeb. Skloubit všechny tyto požadavky v jeden celek je často obtížné a tak i kvalitní rozhraní je výsledkem mnoha kompromisů.

Prvním druhem rozhraní v RRS je rozhraní modul-řídící systém. Moduly RRS jsou v současnosti více samostatnými programy, než moduly napojenými na řídící systém. Je to samozřejmě způsobeno

faktem, že řídící systém ještě nebyl implementován. Přímým důsledkem je potom fakt, že vývojáři modulů si tak musí vytvářet vlastní rozhraní pro ovládání. V některých případech jsou tyto rozhraní implementována do funkcionality modulu tak nešťastně, že přechod na jiné rozhraní znamená zásadní změnu modulu.

Dalším typem rozhraní formát pro komunikaci *řídícího systému* s *moduly* a databází. Jako prostředek pro úschovu dat generovaných modulem a databází byl již v minulosti zvolen formát *RRS-XML*, který je založen na značkovacím jazyku XML. Tento formát byl navržen v rámci bakalářské práce Iva Dlouhého ([5], str. 14).

Posledním identifikovaným rozhraním je webové rozhraní systému ReReSearch, jehož podoba ještě není určena ani navržena. Na serveru athena3 [6] je spuštěna prozatímní verze, která zatím využívá fulltextového vyhledávání. Toto rozhraní je přímo připojeno na databázi, ale vzhledem k některým požadavkům na systém bude nutné webové rozhraní striktně oddělit od modelu (koncept MVC), který bude dále požadavek distribuovat dalším modulům (např. Odpovídání na otázky atp.).

2.3 Analýza nejčastějších procesů a úkonů v rámci RRS

Na základě analýzy dosavadních modulů systému ReReSearch, které se zabývají extrakcí a zpracováním dat a jejich následným uložením do databáze RRS, bylo identifikováno velké množství typů úkonů a procesů.

Základním problémem při výstavbě extraktorů byla chybějící platforma pro kvalitní rozpoznávání požadovaných entit a jejich vztahů v textu. Těmito entitami jsou např. názvy publikací, konferencí, workshopů, rozpoznávání jmen autorů, projektů, prezentací, organizací apod. V systému RRS chyběla také podpora pro normalizaci textu a pokročilou manipulaci s textovými soubory a s textem obecně.

Pro webové extraktory chyběla v RRS podpora pro rychlé stahování a parsování webových stránek, crawling a základní operace na webovým dokumentem (získávání rámců, url-ping apod.). Další částí je již inteligentní rozpoznávání informací, které jsou přítomny na webové stránce: rozpoznávání datových regionů a záznamů, rozpoznávání navigace na stránce, rozpoznávání jazyka stránky atp. Jistě je také potřebný klasifikátor webových dokumentů rozdělující stránky podle kategorie a zaměření (v rámci RRS).

Z hlediska databáze zde byl potřebný model databáze a *ORM vrstva*, která by plně abstrahovala modul pracující s databází od SQL a která by podporovala veškeré automatické procesy při vkládání a úpravě dat v databázi. Vzhledem k potřebě implementovat také jednotné API pro rozhraní bylo rozhodnuto, že bude nutné vytvořit programové prostředky pro práci s formátem RRS-XML, konvertory z databázového modelu na XML a naopak.

Kategorizací uvedených úkonů bylo dosaženo přehlednějšího výsledku analýzy a tyto kategorie také byly základem pro návrh knihovních balíčků:

- manipulace s internetem, jeho protokoly a webem
- extrakce dat z webových dokumentů
- extrakce dat z lokálních dokumentů, identifikace entit a vztahů mezi nimi
- manipulace s databází RRS
- klasifikace dat, třídění
- manipulace s formátem RRS-XML

Později při implementaci extrakčních modulů knihovny bylo nutné vytvořit samostatnou kategorii pro slovníky a API pro přístup k nim.
Pro vzájemnou kompatibilitu, rozšiřitelnost a robustnost modulů by bylo ideální, kdyby knihovna fungovala jako framework pro výstavbu modulů RRS. Této vlastnosti se dá docílit jedině propracovaným návrhem základních částí modulu, jeho komunikace s řídícím systémem a plnou podporou oboru, ve kterém modul funguje a který ovládá.

2.4 Programové knihovny

Tématem této kapitoly jsou obecné vlastnosti programových knihoven, jejich výhody a nevýhody a také teorie návrhu a výstavby aplikačně-programového rozhraní, které je jediným prostředkem pro komunikaci knihovních funkcí s okolními systémy.

2.4.1 Obecné vlastnosti knihoven

Programové knihovny slouží pro programátory jako abstrakce problému, který je již vyřešen a jehož řešení (vlastní implementace) lze v dané oblasti snadno použít. Programátor pak ke knihovním funkcím přistupuje prostřednictvím aplikačně-programového rozhraní jako k černé skříňce (tzv. black box) a očekává, že se funkce bude chovat přesně podle dokumentace bez znalosti vnitřní implementace. Hlavní vlastností knihoven je tedy *zapouzdřenost*, *spolehlivost a znovupoužitelnost*.
Velmi důležitou vlastností knihoven by měla být rozšiřitelnost. Programátoři často potřebují u některé funkce další rozšíření, které by jim poskytlo vyšší komfort nebo lepší možnost jejího ovládání. Špatně navržené knihovny se pak velmi nesnadno rozšiřují a dalšími zásahy se implementace a zdrojový kód knihoven znepřehledňuje.

Knihovny by se daly rozdělit podle účelu na obecné (vestavěné) knihovny jazyka a na aplikačně-specifické knihovny. Ty podporují nějakou aplikaci, program, rozhraní nebo manipulaci s určitým druhem dat nebo procesů. Typickým představitelem problému pro výstavbu knihoven jsou síťové protokoly, datové struktury nebo například vysoké matematické funkce a počty. Dále by se daly knihovny rozdělit podle implementačního jazyka nebo podle podpory objektovosti (např C-stdlib je typická neobjektová knihovna versus Python xml.dom.minidom je striktně objektová knihovna).

2.4.2 Aplikačně-programové rozhraní

Aplikačně-programové rozhraní (dále jen API) je prostředek pro komunikaci a interakci funkcí, tříd a metod s okolními systémy. API je nejčastěji součástí vestavěných knihoven programovacích jazyků, knihoven a specifických systémů, které jsou poskytovány pro volné využití. Součástí API u neobjektových jazyků jsou deklarace funkcí, struktur a konstant pro využití v systému, který API používá. U objektových jazyků je to souhrn všech dostupných veřejných tříd, jejich metod, parametrů a vestavěných konstant. Nezanedbatelnou součástí API je jeho dokumentace.

Návrh kvalitního a jednoduchého API je po vlastním rozvržení, architektuře a sestavením hierarchie funkcionalit nejsložitější částí návrhu celé knihovny. Při návrhu API je vhodné využívat návrhových vzorů a také využívat možnosti implementačního jazyka. Pro návrh kvalitního API neexistuje zaručený postup, který vede k dobrým výsledkům; vždy je to výsledek kombinace úrovně abstrakce problému, kompromisů při implementaci a vlastní invence programátora.

Jako příklad návrhu dobrého a špatného API lze uvést následující situaci:

Je třeba vytvořit API pro jednoduchý přehrávač CD. (Pro jednoduchost uvažujme pouze základní funkce přehrávání bez ladění hlasitosti atp.). V příkladu budeme uvažovat možnost objektového programování s notací syntaxe jazyka Python a s přidanou deklarací typů parametrů metod.

Příkladem špatně navrženého API by mohla být množina těchto tříd a metod:

```
class CompactDisc – hlavní výkonná třída
CompactDisc.play(int track) – přehrání skladby na CD
CompactDisc.stop() - ukončení přehrávání
```

Dobře navržené API lze demonstrovat na následující množině tříd a jejich metod:

```
class CompactDisc – jednoduchý obal kolem dat představující hudbu na CD
class CDPlayer – přehrávač CD
CDPlayer.insert_disc(CompactDisc disc) – vložení disku
CDPlayer.play(int track) – přehrávání skladby
CDPlayer.stop() - ukončení přehrávání
CDPlayer.eject_disc() - vyjmutí disku z přehrávače
```

Výhod druhého řešení oproti prvnímu je několik:

1. Lépe reprezentuje logiku věci: CD nepřehrává samo sebe, ale přehrávač přehrává disk. V prvním případě by se sice dalo říci, že je logické, že přehraji CD, čemuž odpovídá metoda play(), tedy funkcionalita je stejná, ale lépe je znázornit přehrání tím, že pomocí přehrávače přehraji nějaký disk, jako je tomu v případě druhém. Přísná objektová logika je mnohdy u objektově orientovaných API opomíjena. Při výstavbě složitých systémů je logické uspořádání a hierarchie často důležitější, než krátký zápis kódu nebo několik ušetřených milisekund ve vnitřním cyklu programu.
2. Vyhovuje více zadání: bylo třeba navrhnout přehrávač CD, nikoliv přehrávání CD.
3. Z hlediska rozšiřitelnosti je je druhé řešení na podstatně vyšší úrovni: vytvořením zvláštní třídy pro disk je možné v případě potřeby možné přidat atributy disku a zároveň přehrávač může mít nové metody (např. změnit hlasitost přehrávání) nebo atributy (výrobce, model). Přidáním metody pro změnu hlasitosti u třídy reprezentující kompaktní disk by se pak celé rozhraní ještě znepřehlednilo. Z podobných úvah pak lze odvodit výslednou kvalitu navrženého rozhraní.

Kvalitnímu a propracovanému návrhu API se vyplatí věnovat značnou část času vývoje knihovny, protože knihovní funkce a moduly často pak při hierarchickém uspořádání pomáhají samotnému programátorovi, který pak používá své vlastní API a může značně ušetřit čas, síly i prostředky.

2.4.3 Python a jeho verze

Protože převážná část funkcionality jádra systému RRS je napsána v jazyce *Python* [7], bude při návrhu knihovny pro RRS počítáno s tímto jazykem jako implementačním.

V začátcích vývoje byla knihovna vytvářena pro verzi Python 2.6.2 a 2.5.5. Na těchto verzích byla knihovna i testována. Postupně byla přidána i podpora pro Python 2.7 a verze 2.5.5 již není testována, tedy pro tuto verzi není zaručena plná funkcionalita knihovny.

Pokud by bylo třeba implementovat modul v jazyce C s použitím Python/C API, bude pravděpodobně použito API verze 2.7. Implementace takových modulů je jistě náročnější, než modul v interpretovaném Pythonu a proto k tomuto kroku bude přistoupeno jen pokud nebude jiného východiska.

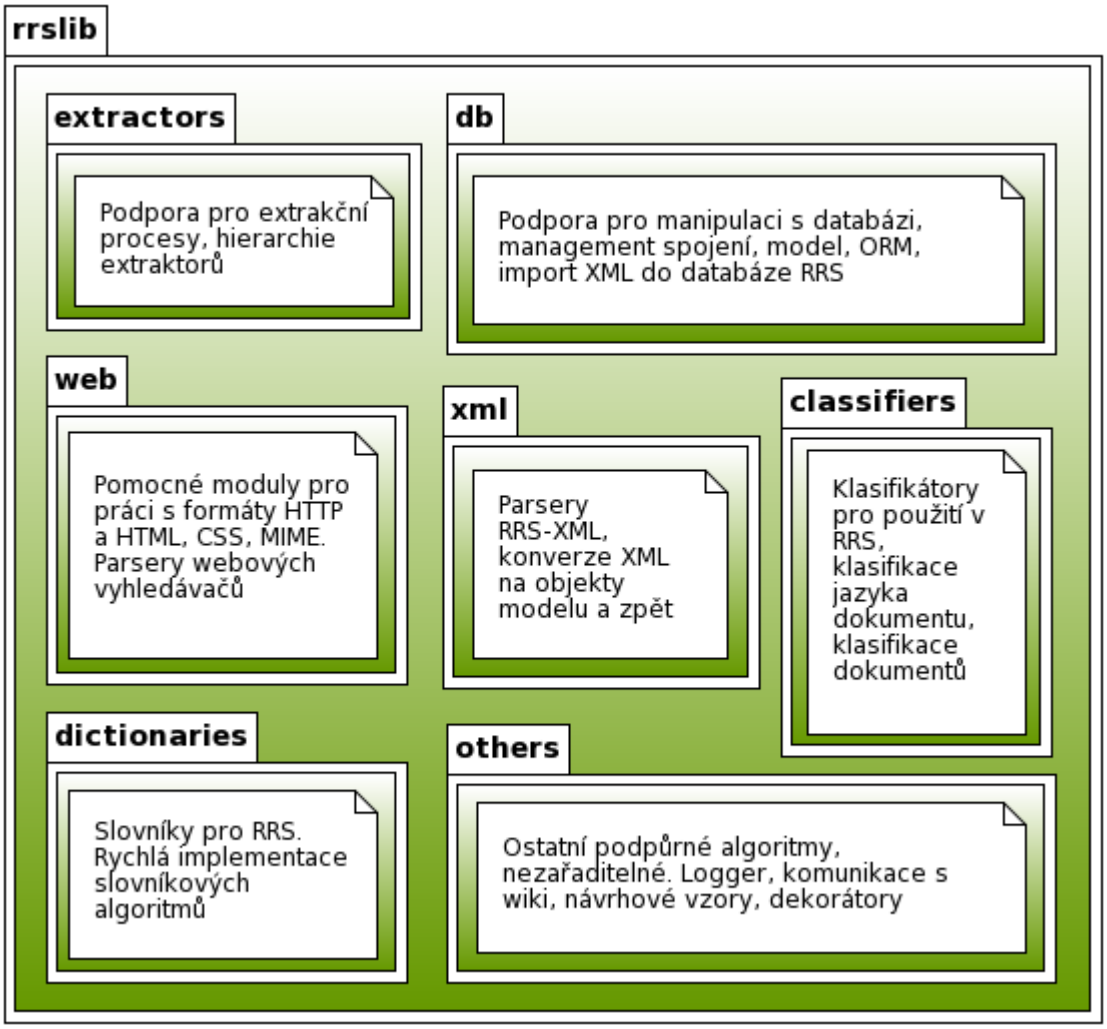
3 Návrh knihovny

V této kapitole bude popsán základní návrh architektury knihovny, jejích vrstev a balíčků. Návrh plně vychází z předchozí analýzy (kapitola 2.2) a používá identifikovaných kategorií jako základu pro tvorbu kostry architektury knihovny.

Jako pracovní název knihovny byl zaveden pojem „*rrs library*“, zkráceně „rrslib“. Kvůli úspoře místa je v dalším textu výhradně používán kratší výraz *rrslib*.

3.1 Architektura

Knihovna rrslib je navržena jako pythonovský balík obsahující podbalíčky sémanticky souvisejících modulů, jejichž podpora se týká stejné oblasti. Návrh rozdělení je zobrazen na obrázku 3.1.



Obrázek 3.1. Architektura knihovny rrslib

Jednotlivé balíčky budou diskutovány v kapitole 3.3.

Knihovní moduly budou implementovány v jazyce Python. Pokud by bylo třeba implementovat náročné a extrémně rychlé algoritmy, bude použit jazyk C, resp. Python/C API [8].

3.2 Zásady při návrhu API pro rrslib

Aplikačně-programové rozhraní pro každý modul je navrženo tak, aby bylo jednoduché a rozšiřitelné. Smyslem knihovny není maximální kompatibilita a vysoká parametrizovatelnost knihovních funkcí, ale jednoduchost použití bez důkladné znalosti používaného algoritmu, při dostatečné možnosti přizpůsobit běh knihovních funkcí vlastním potřebám. Použití by mělo být aplikačně-specifické, tedy cílem není vytvořit knihovny, které by bylo možné distribuovat, ale knihovny, které lze pohodlně použít pro účely systému ReReSearch.

Veškeré knihovní funkce jsou navrženy hierarchicky, knihovna tedy používá často své vlastní funkce k vytváření modulů s vyšší úrovní abstrakce daného problému.

Názvy modulů jsou voleny malými písmeny bez podtržítek. U názvů modulů bylo třeba vyřešit dilema, zda volit spíše krátké obecné názvy a nebo názvy specifičtější delší. Byla zvolena střední cesta: pokud je třeba, je použito delšího názvu, ve většině případů je modul označen jedním slovem vyjadřujícím hlavní funkci nebo oblast podpory. Byla snaha vyhnout se názvům modulů, které by kolidovaly s vestavěnými knihovnami jazyka Python.

Pro vyšší konzistenci knihovny byl zvolen prefix „RRS“ pro všechny třídy, které nejsou použitelné v jiném projektu, než v RRS. Třídy a funkce, které tento prefix neobsahují, jsou obecně použitelné, tedy jejich funkcionalita není pevně navázána na schéma databáze RRS, na řídící systém nebo jiné specifické prvky systému ReReSearch.

Veškeré třídy a funkce dodržují pravidla daná v Python Enhancement Proposals 8 [9], tedy názvy tříd jsou psány CamelCase, jména metod a funkcí jsou lowercase_underscored. Každý modul má definovány základní konstanty:

`__modulename__` - jméno modulu
`__author__` - seznam všech autorů modulu
`__email__` - seznam e-mailových kontaktů autorů
`__date__` - datum poslední úpravy modulu

Každý modul, třída, veřejná metoda a funkce mají dokumentační řetězec, který obsahuje komentář a vysvětlení použití funkce.

Všechny tyto zásady potom pomohou při generování dokumentace programem *epydoc* [10], který rozpoznává dokumentační řetězce a vytvoří ze zdrojového kódu modulu přehlednou stránku dokumentující obsah a jeho použití.

3.3 Podpora vývoje jednotlivých oblastí

V této kapitole budou podrobněji popsány jednotlivé kategorie[2], které jsou uvedeny v kapitole 3.1. Navržené rozdělení balíčků na sémanticky související části později při implementaci nových modulů působilo mírné potíže: některé moduly lze zařadit do dvou kategorií. Tento problém byl vyřešen tím, že byla zvolena vždy kategorie, ke které se modul vztahoval větší částí.

3.3.1 Internet a web

Prvotním úkolem oblasti internetu a webu je zajistit pohodlné a rychlé stahování webových stránek a dokumentů. Tento úkol by měl být řešen jedním modulem, který bude obsahovat rozhraní pro vícevláknové stahování z internetu, popř. rekurzivní crawler prohledávající webové stránky odkazované z jiných. Dále je nutné zajistit některé vysokoúrovňové funkce nad protokolem HTTP – např. url-ping nebo získávání jména souboru podle URL. Dále by měla knihovna poskytovat funkce pro operace se zdrojovými kódy v jazyce HTML: hledání rámců[3] na stránkách, dekódování HTML entit, získávání kódování stránky, získávání odkazů na stránce atp. Velmi užitečné z hlediska extrakce dat bude také parser jazyka CSS (kaskádové styly), který pak bude možné využít pro získávání parametru viditelnosti textu na stránce.

Další velmi významnou složkou knihovny budou parsery webových vyhledávacích strojů. Tyto vyhledávače jsou v RRS používány často pro vyhledání odkazu na webovou stránku autora, publikace, konference aj. Bude nutné zajistit jednotné rozhraní pro všechny vyhledávače a vybrat několik nejvýkonnějších vyhledávacích strojů na internetu, jejichž výsledky budou zpracovávány. Velkou výzvou je vyhledávač *Google*, který pro zpracování výsledků neposkytuje žádné API a který se brání strojovému zpracování částečně tím, že výsledky poskytuje technologií *AJAX* (Asynchronous Javascript And XML); není tedy vytvořen odpovídající HTML kód, který by byl odeslán klientovi, ale je interpretován Javascriptový kód, který se dotazuje serveru na výsledky a vytváří se pouze DOM v operační paměti počítače. Pro řešení tohoto problému by bylo třeba buď implementovat interpret jazyka Javascript, což by bylo časově velmi náročné a nebo použít existující jádro některého z prohlížečů, simulovat dotaz od klienta, interpretovat odpověď serveru a načít výsledek do formy HTML. Toto řešení se zdá být snazší a spolehlivější, než vytvářet interpret jazyka Javascript. Ostatní vyhledávací stroje (Bing, Ask, Altavista, Yahoo) poskytují své výsledky v HTML a tedy vytvoření parseru výsledků vyhledávání je triviální.

3.3.2 Extraktory

Jak již bylo uvedeno v kapitole 2.2, základním prostředkem pro extrakci dat v rámci RRS je vyhledávání a rozpoznávání entit a vztahů mezi nimi. K tomuto účely by měl sloužit samostatný modul, který bude na základě vzorů a regulárních výrazů, popř. s použitím slovníků vyhledávat entity v textu. Tento problém je poměrně komplexní a bude nutné počítat i s limity některých metod. Modul pro *extrakci entit z textu* bude představovat základ pro celou hierarchii extraktorů v knihovně rrslib.

Na základě extrakce entit bude nutné pomocí přídavných algoritmů a postupů vystavět extraktor zaměřený na získávání informací z textů referencí a citací. Tento problém je esenciální pro celý

2 oblasti systému RRS, které knihovna bude přímo podporovat

3 části stránek, které jsou pevně umístěny na stránce a jejichž zdrojový kód není v kódu stránky obsahující rámce zobrazen. HTML značka rámce je `<frame>` nebo `<iframe>`

systém RRS a bez kvalitního rozpoznávání entit v referencích nebude nikdy ReReSearch systémem srovnatelným se službami jako Microsoft Academic Search, DBLP nebo CiteSeerX.

Dalšími extraktory v rrslib budou systémy pro rozpoznávání částí textových (popř. PDF) dokumentů. Tyto dokumenty lze vždy rozdělit na celky představující hlavičku, abstrakt, obsah a reference/literaturu. Výstupem tohoto modulu by tedy měl být text rozdělený na uvedené části a důvěryhodnost výsledku (angl. *credibility* – vysvětleno v kapitole 2.1). Do oblasti extraktorů také jistě bude patřit modul podporující webové extraktory. Tento by měl automaticky rozpoznávat datové regiony a záznamy na webové stránce a měl by také umět získávat navigaci a URL odkazovaných stránek.

Poslední složkou oblasti extraktorů budou pomocné moduly pro normalizaci textu (řešení unicode, sjednocování znaků, řešení slitků[4]) a knihovna užitečných regulárních výrazů.

Spojením výše uvedených extraktorů by bylo vhodné implementovat vysokoúrovňové extraktory pro automatickou extrakci dat z webových stránek (např. seznamy publikací, konferencí atp.) a z lokálních dokumentů (navazuje na práci Tomáše Lokaje [11]).

3.3.3 Databáze a XML rozhraní

Zásadním problémem při vývoji systému ReReSearch je často se měnící databázové schéma. Moduly RRS pak po změně nejsou připraveny na aktuální změny a nejsou často schopny ani testovacího běhu, dokud není modul upraven. Je tedy nutné v knihovně vytvořit jednotné rozhraní pro manipulaci s daty v rámci RRS. Proto bylo rozhodnuto o potřebnosti *modelu databáze* a *ORM*, které by poskytlo pohodlnější práci s entitami databáze a zpřehlednilo veškerou manipulaci s daty v rámci modulů systému RRS.

Před vytvořením nového modelu a ORM databáze bylo nutné rozhodnout, zda nepoužít nějaký již existující ORM framework, který by obsahoval potřebnou funkcionalitu. Vzhledem k častým změnám, které databáze RRS postihuje, se zdálo být účelné navrhnout vlastní objektový model databáze a poté implementovat mapovací vrstvu, které by byl model vytvořen tzv. „na míru“. Aby bylo možné udržet aktualizovanost ORM vrstvy, je třeba model automaticky generovat ze schématu databáze a tím je umožnit při každé změně databáze vytvoření nového modelu v relativně krátkém čase. Model je třeba vybavit také silnými typovými kontrolami[5] a je nutné zajistit oboustranné provázání objektů, aby objekt dokázal poskytnout veškeré informace o topologii entity. Kvůli silné návaznosti modelu na import dat do databáze a navíc na formát XML, který po objektech požaduje také specifické vlastnosti, bylo zavrženo použití ORM frameworku a byla započata práce na návrhu vlastního modelu a databázové vrstvy s vlastním ORM.

Návrh modelu databáze spojený s ORM vrstvou je inspirován aplikacemi založenými na MVC (model-view-controller). Tato vrstva by měla poskytnout

- dostatečnou abstrakci dat na straně modulů RRS
- prostor pro změny v databázi: úpravou knihovního modulu budou přizpůsobeny všechny moduly, které model databáze používají

Tento model budou moduly RRS používat na své straně naprosto odděleně od databáze, pomocí konvertorů je převedou do XML, které bude na výstupu modulu. Toto XML pak bude zpracováno

4 český název pro ligaturu – termín označující spojení dvou znaků do jednoho písmena. V kontextu strojového zpracování je nutné řešit slitky, které se vyskytují v tabulce znaků UNICODE.

5 Python je jazyk silně a dynamicky typovaný - typ je tedy svázán s hodnotou a ne s proměnnou. V modelu je ale třeba vytvořit mechanismus, který by vázal typ s proměnnou, aby byly hodnoty nesprávného typu odstraněny ještě před kontrolou integritních omezení v databázi.

dalšími moduly, které budou data z XML vkládat do databáze RRS. Model musí být schopen akceptovat i jiné atributy, než ty, které odpovídají aktuálnímu schématu databáze: musí být tedy připraven na práci s neznámými atributy a typy.

V knihovně musí také existovat rozhraní pro připojení k databázi RRS. Toto rozhraní musí obsahovat nejen přímé spojení na databázi a příslušné obslužné rutiny, ale také management spojení s databází a a jistou úroveň abstrakce nad operacemi s databází. Velmi vhodné bude tedy implementovat základní třídu představující obecnou databázi *PostgreSQL*[6] s managementem spojení a dále s jejím použitím vystavět ORM vrstvu nad modelem databáze.

3.3.4 Slovníky

Při návrhu a implementaci extrakčních algoritmů byla zjištěna potřeba speciálních slovníků, které pomáhají extraktorům nacházet vhodné kandidáty pro označení entity spadající do určené entitní množiny. Protože tyto slovníky mohou být potenciálně velmi velké (desítky tisíc položek), bylo třeba kvalitně navrhnout formát uložení těchto slovníků a implementovat rychlé algoritmy pro hledání ve slovnících, popř. specializované algoritmy nad slovníky přesně pro potřeby extraktorů.

Tyto slovníky by měly být tedy implementovány s použitím protokolu *pickle*. Tento protokol poskytuje možnost ukládání objektů jazyka Python do textových řetězců pomocí speciálního protokolu. Ve standardní knihovně jazyka Python existují dvě implementace tohoto protokolu: knihovna *pickle*, která je implementována v jazyce Python a knihovna *cPickle*, která je vytvořena v jazyce C a je několikanásobně rychlejší. Pro implementaci slovníků v rrslib byla převážně kvůli rychlosti vybrána knihovna cPickle. Veškeré struktury (převážně seznamy a slovníky) budou pomocí této knihovny uloženy do souborů, které pak budou v průběhu hledání ve slovníku načítány do operační paměti. Protože překladové slovníky budou velmi velké, bude nutné řešit i otázku umístění dat v operační paměti. Ideálně by se měly načítat pouze ty data, která jsou aktuálně třeba aniž by tím byla zásadně omezena rychlost. Protože právě rychlost bude hlavním ukazatelem kvality slovníků, bude nutné použít pro hledání klíčů nebo hodnot ideálně algoritmů se složitostí *O(1)* (hashování) nebo *O(log(n))* (binární vyhledávání).

3.3.5 Klasifikátory

Přestože velké množství klasifikačních algoritmů je přímo zabudováno v některých extraktorech, je třeba vytvořit ještě samostatné obecné klasifikátory pro použití v dalších modulech nebo programech. Původní záměr byl implementovat obecné metody a techniky neuronových stí, bayesovské teorie, Support Vector Machine a další techniky. Vzhledem k tomu, že v RRS bylo třeba implementovat jiný druh klasifikátorů – převáže cílené klasifikace dokumentů, jazyka nebo třídění publikací, bylo od tohoto záměru upuštěno a byly navrženy klasifikátory založené na různých algoritmických technikách pro účely systému ReReSearch:

- klasifikace typu dokumentu (publikace, prezentace aj.)
- klasifikace úspěšnosti převodu PDF dokumentu na text
- klasifikace jazyka dokumentu
- klasifikace typu publikace (inproceedings, inbook, misc, techreport atp.)

Tyto klasifikátory budou pravděpodobně založeny většinou na regulárních výrazech, vzorech nebo pravděpodobnostních modelech.

6 databáze v systému ReReSearch je typu PostgreSQL

4 Implementace knihovny

Tato kapitola popisuje postup při implementaci knihovních modulů, zavedení prostředků pro správu verzí, vytvoření testovacího prostředí a v podkapitolách jsou vybrané moduly podrobněji popsány. Knihovna rrslib byla implementována v jazyce Python verze 2.6.2 se zpětnou podporou pro Python 2.5.5 a pozdější vývoj se přesunul na verzi 2.6.6 a nakonec 2.7.

4.1 Prostředky pro vývoj a distribuci

Základním kamenem každého vývoje software je systém pro správu verzí. Původně v návrhu byl vybrán systém SVN, ale brzy byla knihovna migrována na systém *git*. Git má velkou výhodu oproti SVN: není nutné mít spuštěného démona, který běží na serveru a který by bylo nutné po každém pádu serveru restartovat. Byl tedy založen git repozitář a zavedeny pravidla pro práci s repozitářem tak, aby si vývojáři při zakládání nových verzí nekomplikovali vzájemně práci. Repozitář knihovny rrslib se nachází na serveru minerva1 na sdíleném diskovém poli přístupném ze serverů athéna, minerva, merlin a z počítačů pcnlp v adresáři projektu *rrs_library*.

Pro distribuci knihovny bylo založeno prostředí pro tvorbu balíčků. Z podporovaných typů balíčků byl vybrán RPM pro Red Hat/Fedora Linux (popř. CentOS, Mandriva, OpenSUSE atd.) a pro ostatní je nutná instalace ze zdrojových kódů z balíku .tar.gz. Pro automatizaci této operace byl vytvořen skript, který vytváří balíčky z poslední revize knihovny z git repozitáře. K vlastní knihovně jsou přibaleny také instalační soubory. Jestliže jsou vytvářeny balíčky, bylo nutné pro knihovnu zavést odpovídající systém verzování a označení verzí.

Ve své současné podobě je název verze založen na datu vydání knihovny a je ve formátu:

```
0.yy.mm.dd-rr
```

kde první nula značí fakt, že knihovna není dokončena,
`yy` – dvě poslední číslice roku vydání,
`mm` – číslo měsíce,
`dd` – čislo dne,
`rr` – vydání (release)

Pro udržení funkcionality knihovny během vývoje byl vytvořen automatizovaný systém testování. Tento mechanismus je podrobněji popsán v kapitole 6.

4.2 Adresářová struktura knihovny, balíčky

Knihovna je umístěna ve složce rrslib, která reprezentuje celou knihovnu a také hlavní balík. Dále je knihovna členěna na podbalíčky, jak bylo uvedeno v návrhu (viz kapitola 3.1). Každý balíček obsahuje __init__.py soubor, který definuje složku jako balíček, který má být pythonem registrován.

Výpis balíčků a jejich modulů je tedy následující:

rrslib/

__init__.py – obsahuje definici podbalíčků. Obsahuje informaci o verzi knihovny.

classifiers/

documentinfo.py – klasifikace typu dokumentu (nikoliv tématu nebo obsahu textu)
language.py – identifikace jazyka textu
pdf2textcredibility.py – klasifikace úspěšnosti převodu PDF na text pomocí *pdftotext*

db/

dbal.py - abstraktní databázová vrstva a ORM
model.py – objektový model databáze
modelcreator.py – skript pro převod SQL schématu na model databáze
xmlimport.py – import dat uložených ve formátu RRS-XML do databáze RRS

dictionaries/

rrsdictionary.py – rozhraní pro užití knihovních slovníků
rrsdictcreator.py – skript pro vytváření knihovních slovníků
složky slovníků

extractors/

articlemetaextractor.py – extrakce dat z textových dokumentů
bibtexparser.py – parser formátu BibTeX a převodník do objektů modelu
citationentityextractor.py – extrakce entit z referencí a citací
documentwrapper.py – parser dokumentů, rozdělení na části dokumentu
entityextractor.py – extrakce entit z textu
normalize.py – normalizace textu
rrsregex.py – knihovna regulárních výrazů
webmetaextractor.py – webové extraktory

others/

daemon.py – implementace démona
logger.py – logování pro moduly RRS
pattern.py – implementace základních návrhových vzorů, dekorátory
progressbar.py – grafický prvek pro příkazovou řádku, vhodné pro testovací účely
wiki.py – programové rozhraní pro úpravu NLP-wiki

web/

crawler.py – rozhraní pro stahování webových dokumentů s využitím threadingu
mime.py – nadstavba nad vestavěný modul *mimetools* s využitím threadingu
separsers.py – parsery webových vyhledávacích strojů (Bing, Altavista, Ask, Yahoo)
sequencewrapper.py – rozpoznávání opakujících se sekvencí na webové stránce
httptools.py – různé nástroje pro práci s HTTP protokolem
htmltools.py – abstarkce HTML dokumentu, práce s jazykem HTML
csstools.py – parser CSS a příslušné struktury pro popis pravidel kaskádových stylů

xml/

sax.py – vybrané SAX handlery (podpora pro xmlconverter)
xmlconverter.py – konvertory z objektů modelu do XML a nazpět

4.3 Vybrané moduly knihovny rrslib

Kvůli omezenému rozsahu této práce není možné popsat veškeré implementované moduly knihovny (a ani by to nebylo nijak účelné). Byly proto vybrány moduly reprezentující svoji kategorii (balíček) a ty jsou v následujících kapitolách podrobněji popsány. Tyto popisy nejsou zcela vyčerpávající, některé algoritmy jsou popsány podrobněji, u některých bylo za důležitější považováno důkladné vysvětlení principu a souvislostí.
Názvy podkapitol jsou zvolené tak, aby popisovaly obecný účel modulu a také cestu pro import modulu v jazyce Python – tedy tak, jak lze modul v praxi importovat a dále používat.

4.3.1 Objektový model databáze – rrslib.db.model

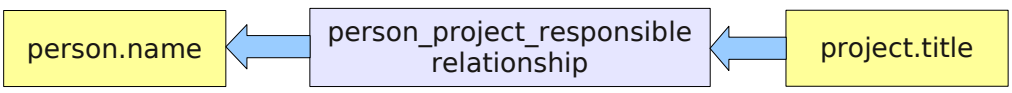
V knihovně rrslib existuje modul rrslib.db.modelcreator, který po spuštění vygeneruje ze SQL schématu generovaného návrhovým programem SQLPowerArchitect [12] nový model (v jazyce Python). Konverze SQL do tříd Pythonu je netriviální a vzhledem k tomu, že model neobsahuje informace o databázových typech a integritních omezení (což považuji za největší problém implementovaného modelu), musí se model spoléhat na konvenci názvů tabulek, spojovacích tabulek a cizích klíčů z databázového schématu, což se jeví jako nedostatečně robustní řešení.

V modelu databáze jsou tedy k dispozici následující typy:

- základní objekt (odpovídající záznamu v tabulce, jmenná konvence: RRS+název tabulky)
- vztahový objekt (odpovídající vztahu 1:N nebo N:N, jmenná konvence: RRSRelationship + názvy spojovaných tabulek)
- pomocné objekty (reprezentující některé významnější prvky, např. e-mail, datum podporující formáty zavedené v RRS-XML)

Základní objekty nesou informaci o vlastní entitě (např. publikace, osoba, projekt) . Základní vztah 1:N je modelován přímým vložením objektu do jiného objektu. Vztah N:N je modelován na každé straně (u každé entity) seznamem vztahových objektů, do kterých jsou vkládány vztahové objekty. Tyto vztahové objekty představují záznam ve spojovacích tabulkách a obsahují případné atributy vztahu.

Ilustrací použití modelu by mohla být následující situace vycházející z reálných potřeb některých modelů systému ReReSearch:
Je třeba vytvořit strukturu, která ponese informaci o osobě „John Smith“, která je zodpovědná za projekt „ReReSearch“.
Vzhledem k faktu, že vztah mezi projektem a zodpovědnou osobou je v databázi RRS modelován tabulkou `j_person_project_responsible`, je nutné vložit do osoby (případně do projektu - tyto dva přístupy jsou ekvivalentní) vztahový objekt, který bude později reprezentovat záznam ve spojovací tabulce.



Obrázek 4.1 Propojení dvou objektů modelu databáze vztahovým objektem

Tato struktura by zapsána v jazyce Python za použití modelu databáze RRS vypadala takto:

```
from rrslib.db.model import *
# vytvoření entity osoby
pers = RRSPerson()
pers['first_name'] = "John"
pers['last_name'] = "Smith"
# vytvoření entity projektu
proj = RRSProject(title="ReReSearch")
# vytvoření vztahového objektu a přiřazení
resp = RRSRelationshipPersonProjectResponsible()
resp.set_entity(proj)
pers['project_responsible'] = resp
```

Objektový model je využíván převážně v modulech, které extrahují data nebo s daty jinak pracují. Modul pracuje pouze s objekty a vztahovými objekty a vytváří z nich strom. Tento strom je poté serializován konvertory do formátu XML (viz. kapitola 4.2.3).

4.3.2 Abstraktní databázová vrstva – rrslib.db.dbal

Cílem tohoto modulu je poskytnout podporu pro komunikaci s databází RRS. Modul rrslib.db.dbal obsahuje abstrakci databáze PostgreSQL, třídu pro parametrizovatelné SQL dotazy a vlastní ORM – abstrakci databáze RRS, která pracuje s objekty modelu.

Základním kamenem této vrstvy je třída `PostgreSQLDatabase`, která je nejnižší úrovní abstrakce v tomto modulu. Obsahuje základní metody pro práci s databází a z hlediska abstrakce je jen o malou úroveň výš, než třída `psycopg2.Connection`, která je již implementací obsluhy databáze. Třída `PostgreSQLDatabase` navíc přináší několikaúrovňové logování událostí, nastavení Unicode, obsluhu kurzorů a databázových sekvencí.

Další částí modulu dbal je třída `FluentSQLQuery`, která není součástí hierarchie vrstev poskytující určitou úroveň abstrakce, ale která vytváří programové rozhraní pro parametrizování SQL dotazů. Tato třída je inspirována třídou `dibiFluent` z knihovny dibi vytvořené Davidem Grundlem [13]. Jedná se o SQL dotaz, který není vytvářen jako řetězec, ale jako objekt, jehož chování upravují volané metody objektu. Tím se vytváří možnost upravovat dotaz na základě běhu programu a jeho parametrů. Třída tedy obsahuje metody pro vytváření a úpravu dotazu (tedy klasické SQL parametry jako SELECT, WHERE, LIKE atp.) a některé další funkce (navracení posledního vloženého ID, fulltextové hledání atp.). Zatím chybí podpora agregačních funkcí a funkce JOIN. Objekt třídy `FluentSQLQuery` obsahuje spojení na databázi a je volatelný. Voláním objektu jako metody je dotaz proveden a navrací výsledek.

Příkladem použití parametrizovatelného dotazu `FluentSQLQuery` je tato ukázka:

```
# vytvoření dotazu
query = FluentSQLQuery()
query.select("*").from_table("person")
query.where("first_name=", "John").or_("last_name=", "Jack")
query.order_by("id", DESC).limit(5).offset(10)
# provedení dotazu
query()

# získání dat
print query.fetch_all()
```

Poslední implementovanou vrstvou je již dokončení ORM, tedy vrstva, která pracuje již pouze s objekty a na jejich základě vytváří SQL dotazy, které posílá nižší vrstvě. Tato nejvyšší vrstva je reprezentována třídou `RRSDatabase`, která již přijímá objekty modelu a z nich vytváří pomocí `FluentSQLQuery` SQL dotazy, které zpracovává nejnižší vrstva – `PostgreSQLDatabase`. Třída poskytuje metody pro vkládání entit, vztahů a metody pro jejich úpravu.

Tak jaké v každém ORM, i zde bylo nutné řešit optimalizaci SQL dotazů a jejich množství. Nezřídka se stává, že vysoká úroveň abstrakce, která se u ORM očekává, vede na zbytečné množství dotazů, které nejsou sloučeny do jednoho, optimalizovaného. Dalším problémem, který bylo nutno vyřešit, byl způsob úprav již existujících dat v databázi, pokud na vstupu jsou data, která jsou rozšířením již stávajících. Byly navrženy čtyři způsoby chování ORM vrstvy při příkazu typu „update“:

- přepsání existujících dat daty novými
- přidání pouze atributů, které nejsou v databázi přítomny (jsou NULL)
- rozhodnutí na základě důvěryhodnosti dat (parametr credibility)
- inteligentní kombinace výše uvedených metod (neimplementováno)

Použití ORM je přímočaré:

```
from rrslib.db.dbal import *
# inicializace databáze
db = RRSDatabase(schema="data", logs=NO_LOGS)
# vytvoření objektu, který je třeba uložit do db
person = RRSPerson(first_name="Adam", last_name="Dvořák")
# vložení objektu do databáze, automaticky se vloží záznam do metadat
# o původu informace – modul publicatiom_text_data
person = db.insert(person, "publication_text_data")
# nyní bylo osobě přiřazeno ID, které je možné vytisknout
print person['id']
```

V současnosti chybí implementace metod pro filtrování výběru objektů a metody pro mazání záznamů z databáze.

4.3.3 Konvertory modelu do RRS-XML a zpět – rrslib.xml.xmlconverter

Formát RRS-XML hraje v systému ReReSearch velmi významnou roli z hlediska spolupráce systémů, které jsou vzájemně nekompatibilní z hlediska programovacího paradigmatu nebo jazyka. XML slouží v RRS jako datový nosič, tedy jako výstup z mnoha modulů, které data generují nebo upravují a také jako vstup pro moduly, které dále XML ukládají do databáze nebo filtrují.
Každý modul systému ReReSearch, který extrahuje nebo upravuje data, pracuje s objektovým modelem databáze. Na výstupu je vždy vytvořený objektový strom, který je pak pomocí konvertoru serializován na XML. XML je pak uloženo do souborů a později je možné XML znovu načít a zpracovat (např. importovat data do databáze).

Modul rrslib.xml.xmlconverter obsahuje dva typy konvertorů:

1) Model2XMLConverter – konverze objektů na XML
2) XML2ModelConverter – konverze XML na objekty

Nejčastěji používaný je první typ konvertoru, protože ten se vyskytuje na straně modulů RRS, druhý typ je zatím použit pouze v modulu pro import XML do databáze RRS (viz kapitola 4.2.4). Oba konvertory používají pro zpracování formátu XML metodu SAX (Simple API for XML), která je

šetrnější vůči systémovým prostředkům, než metoda DOM. Bylo by značně neefektivní vytvářet DOM z vstupního XML, když by byl vytvořen další objektový model (nyní již model databáze RRS).

Zajímavostí při konverzi modelu na XML byly takzvané odkazy na entity; v některých případech je třeba v XML popsat kruhovou závislost nebo vztah objektů modelu, který nelze popsat stromovou strukturou formátu XML. Tyto případy se pak řeší vytvořením štítku entity (atribut. „label“), který funguje jako odkazovaný prvek. Pokud je třeba vytvořit odkaz na tuto entitu v jiné části XML, je vytvořen element popisující entitu stejné třídy, které je vložen pouze odkaz (atribut „ref“) s hodnotou štítku (těmito hodnotami jsou hexadecimální řetězce o délce 6 znaků), jak je tomu v v následujícím příkladu:

```
<?xml version="1.0" encoding="utf-8" ?>
<output>
  <citation>
    <publication label="5ac97f">
      <title value="FAST TCP: From Theory to Experiments"/>
      ...
    </publication>
    <reference>
      <content value="[77] C. Jin, D. Wei. FAST TCP: From Theory to
Experiments, Submitted for publication in IEEE Communications Magazine,
Internet Technology Series, April 1, 2003."/>
      <publication role="referenced" ref="5ac97f"/>
    </reference>
    <content value="(Fast TCP, [77])"/>
  </citation>
</output>
```

Problémem je zpracování odkazů na entity při použití SAX: v době, kdy je třeba vytvořit odkaz, není k dispozici hodnota štítku odkazované entity, protože štítek nebyl vytvořen. Při použití SAX není možné zpětně přistupovat k entitám, které již byly zapsány, protože SAX zpracovává XML proudově. Řešením je vložení každému elementu štítek, který je v konvertoru mapován na objekt a při výskytu odkazu na entitu se na dané místo modelu pak vloží již namapovaný objekt. Toto řešení obchází nepříjemnou uvedenou vlastnost proudového zpracování XML (nemožnost návratu zpět ve zpracovávaném dokumentu), ale přidává do některých elementů redundantní informaci, která nebude nikdy využita.

4.3.4 Import XML do databáze RRS – rrslib.db.xmlimport

Základním problémem při extrakci dat nebo při získávání znalostí z dat již existujících je formát přenosu dat do databáze a pravidla, podle kterých se tento přenos odehrává. V mnoha systémech pracujících s databázemi není třeba zvláštních pravidel pro *zjednoznačňování*[7], která by definovala shodnost vstupních entit s daty, které již v databázi existují na základě shodných nebo podobných atributů. Důvodem může být orientace systému – například procesně orientovaná databáze zaznamenávající úpravy článku na webové stránce nepotřebuje řešit shodnost článků nebo osob, které článek upravují. Naproti tomu ve znalostních systémech typu ReReSearch musí být na tuto vlastnost brán velký zřetel; každý vstup musí být kontrolován pro případnou shodu s již existujícím záznamem v databázi. Proto bylo nutné vytvořit knihovní modul, který by unifikoval pravidla pro porovnávání dat v databázi RRS, definoval procesy, které jsou potřebné pro normalizaci a porovnávání a vytvořil rozhraní pro import dat do databáze s inteligentním algoritmem zjednoznačňování entit.

7 proces nacházení identických entit v databázi

V systému RRS jsou výstupní data z extraktorů ukládána do souborů ve formátu RRS-XML. Těchto souborů bývají na výstupu jednoho extraktoru až tisíce. Import dat do databáze tedy znamená získat data z XML souborů, zpracovat je a inteligentně uložit do databáze.

Během návrhu samotného systému musely být řešeny tyto podproblémy a situace:

- jak zpracovat XML tak, aby každá entita měla vazby na všechny entity, se kterými sémanticky souvisí
- řešení cyklických závislostí entit
- odstranění nevalidních nebo nevyužitelných dat
- inteligentní vyhledávání entit na základě společných atributů
- předzpracování a doplnění chybějících dat (normalizované názvy a jména, MIME[8] typy atp.)
- provázání vstupních dat tak, aby v databázi byla kompletní informace
- způsob úpravy stávajících záznamů
- ukládání průběhu importu dat (logování)
- optimalizace, výkon

Hlavním úkolem bylo vyřešit porovnávání jednotlivých entit z XML s již existujícími záznamy v databázi. Pro tento účel byla vytvořena programová tabulka pravidel, která jsou parametrizovatelná a několikaúrovňová: entity nejsou porovnávány jen na základě svých vlastních atributů, ale i cizích klíčů, popř. jejich atributů. Úroveň, do které lze data dohledávat, je samozřejmě do jisté míry omezena množstvím vstupních dat – pokud v XML nejsou data do požadované úrovně přítomna, není možné je s ničím porovnat.

Pro ilustraci problému bude nyní vhodné popsat nejčastější situace a odpovídající dotazy:
Jsou dána vstupní data ve formátu XML obsahující smyšlenou publikaci „Scientific portal generator – new approach“, jejíž autorem je doc. Pavel Smrž. Tato publikace byla prezentována na konferenci NLDB. Původ těchto dat není v tomto příkladě důležitý.

Odpovídající data v RRS-XML jsou tedy uložena takto:

```
<?xml version="1.0" encoding="utf-8" ?>
<output>
    <publication>
        <title value="Scientific portal generator – new approach"/>
        <person author_rank="1" editor="False">
            <first_name value="Pavel" />
            <last_name value="Smrž" />
            <full_name value="Pavel Smrž" />
        </person>
        <event>
            <title value="International Conference on Applications of
                          Natural Language to Information Systems" />
            <acronym value="NLDB" />
        </event>
    </publication>
</output>
```

Dotazy na databázi RRS tedy budou odpovídat tomu, jak by se v přirozeném jazyce každý člověk intuitivně ptal, kdyby neznal žádné dodatečné informace:

- známe publikaci s názvem „Scientific portal generator – new approach“ ? Pokud ne, tak...
- ...známe událost „International Conference of Applications of Natural...“ ? Pokud ne, tak...
- ...známe alespoň takovou událost, která by měla alespoň akronym „NLDB“? Pokud ano, tak..

8 Multipurpose Internet Mail Extensions – internetový standard umožňující přenášet e-mailové zprávy s přílohou v různých formátech. Je využíván i protokolem HTTP.

- ...známe osobu se jménem Pavel Smrž/Pavel Smrz nebo s povrchovou formou P. Smrz, Smrz Pavel atp.? Pokud ano...
- ...známe publikaci, kterou by napsala osoba se jménem Pavel Smrž nebo alespoň s příjmením Smrž/Smrz a která byla zároveň prezentována na události „Interantional Conference ...“ nebo „NLDB“?
- jestliže je více takových publikací, název které se podobá nejvíce řetězci „Scientific portal generator – new approach“?
- a tak podobně...

Tyto dotazy samozřejmě je nutné zformulovat v jazyce SQL. Podobné porovnání je zřetelné z obrázku 4.2. Výkonný algoritmus by měl potom dokázat na základě uvedených porovnání sjednotit nebo naopak přidat korelující data:

- jestliže známe publikaci s názvem „Scientific portal generator – new approach“, potom je nutné této publikaci přiřadit autora Pavel Smrž.
- Pokud se již v databázi tento autor vyskytuje, je nutné pouze vytvořit spojení mezi publikací a existující osobou.
- Pokud se osoba v databázi vyskytuje v jiné formě (příjmení „Smrž“, jméno. „P.“ atp.), je nutné vytvořit novou povrchovou formu jména a přiřadit jméno publikaci.
- Jestliže na publikaci není navázána konference se jménem „International Conference on Applications of Natural Language to Information Systems“, pak je nutné tento záznam vložit do databáze a vytvořit spojení mezi publikací a konferencí.

```
<?xml version="1.0" encoding="utf-8" ?>
<output>
    <publication>
        <title value="Scientific portal generator – new approach"/>
        <person author_rank="1" editor="False">
            <first_name value="P." />
            <last_name value="Smrž" />
            <full_name value="Pavel Smrž" />
        </person>
    </publication>
</output>
```

```
<?xml version="1.0" encoding="utf-8" ?>
<output>
    <publication>
        <title value="Scientific portal generator – new approach"/>
        <person author_rank="1" editor="False">
            <first_name value="Pavel" />
            <last_name value="Smrž" />
            <full_name value="Pavel Smrž" />
        </person>
        <event>
            <title value="International Conference on Applications of
                          Natural Language to Information Systems" />
            <acronym value="NLDB" />
        </event>
    </publication>
</output>
```

Obrázek 4.2 Ekvivalence atributů entit v RRS-XML na jejichž základě je určena shodnost porovnávané entity.

Implementace uvedených algoritmů hledání a přiřazování včetně dalších podprocesů (normalizace, předzpracování atp.) je implementována v modulu rrslib.db.xmlimport. Třídní diagram hlavních tříd modulu je k dispozici v příloze A. Hlavní třída `RRSXMLImporter` poskytuje rozhraní pro inteligentní import XML nebo objektů modelu do databáze RRS, k čemuž využívá výše popsaných (a mnoha dalších) postupů. Importér využívá modulu rrslib.xml.xmlconverter pro konverzi XML na objekty, se kterými poté pracuje a které do databáze ukládá pomocí ORM vrstvy.

4.3.5 Extrakce opakujících se sekvencí na webových stránkách – rrslib.web.sequencewrapper

Extrakce dat z webových stránek má oproti zpracování lokálních dokumentů značnou výhodu: přítomnost formátovacích značek ve zdrojovém kódu. Tímto se z nestrukturovaného textu stává semistrukturovaný text a extrakce z takových dokumentů je do jisté míry značně zjednodušena. Bohužel, přístupnost nástrojů pro tvorbu webových dokumentů (WYSIWYG editorů) a množství amatérů a poloprofesionálů, kteří tvoří webové stránky bez hlubšího porozumění problematiky sémantického HTML – to vše staví trendu automatické extrakce dat z webu velké překážky. Mnohdy je tak webový dokument nejen že nestrukturovaný, ale často i chybně strukturovaný, což je pro webový extraktor matoucí.

Esenciálním problémem při extrakci dat z webových stránek v modulech RRS je rozpoznání struktury stránky; rozpoznání navigace (je-li jaká), obsahu, popř. získání struktury obsahu stránky. Rozpoznávání těchto částí stránek je častý proces u těch modulů systému ReReSearch, které extrahují data ze stránek, u nichž není předem známa struktura stránky; jsou to například moduly pro:

- extrakci referencí ze stránky s publikacemi autora
- extrakci výzkumných zpráv ze stránky projektu
- extrakcí programu ze stránky konference

Všechny tyto typy stránek mají jedno společné: jedná se o extrakci dat ze seznamů, tabulek nebo obecně sekvencí.

V ideálním případě je webový dokument dobře hierarchicky vystavěn a jsou v něm správně používány sémantické značky. Zpracování takového dokumentu je pak jednodušší, protože struktura odpovídá obsahu stránky a lze lépe rozpoznat jednotlivé datové regiony a záznamy. Pokud je webový dokument nesprávně vytvořen a pokud programátor nevyužívá sémantických možností jazyka HTML, extrakční algoritmy poté selhávají a stránka je strojově obtížně zpracovatelná. Příkladem dobře strukturovaného webového dokumentu může být stránka s publikacemi Jamese T. Hynese [14], náhled její části je zobrazen na obrázku 4.3.

Publications List

James T. Hynes is author or co-author of over 200 research articles (this list is not exhaustive). Over 325 invited lectures and seminars.

In order to gain access to the publications made available here please send us an e-mail asking for the password.

Publications since 1995

1. Arnulf Staib, James T. Hynes and Daniel Borgis, "Proton Transfer in Hydrogen-Bonded Acid-Base Complexes in Polar Solvents", *J. Chem. Phys.* **102**, 2487-2505 (1995)
2. Roberto Bianco and James T. Hynes, "VB Resonance Theory in Solution. I. Multi-state Formulation", *J. Chem. Phys.* **102**, 7864-7884 (1995)

Obrázek 4.3 Náhled části stránky dobře strukturovaného webového dokumentu

Tato stránka je v HTML kódu zapsána takto:

```
<div class="content">
  <h1>Publication List</h1>
  <p>James T. Hynes is author or co-author of over 200 research articles...</p>
  <p>In order to gain access to the publications made available here please...</p>
  <h2>Publications since 1995</h2>
  <ol>
    <li class="bib">
      <span class="author">Arnulf Staib, James T. Hynes and Daniel Borgis</span>
      ", "
      <span class="title">
        <a href="publications/staib.1995.jcp.102.2487.pdf">Proton Transfer ...</a>
      </span>
      ", "
      <span class="journal">J. Chem. Phys</span>
      <span class="volume">102</span>
      ", "
      <span class="page">2487-2505</span>
      <span class="year">(1995)</span>
      ...
    </li>
    <li class="bib">
      <span class="author">Roberto Bianco James T. Hynes</span>
      ", "
      <span class="title">
        <a href="publications/bianco.1995.jcp.102.7864.pdf">VB Resonance Theory in.</a>
      </span>
      ...
    </li>
```

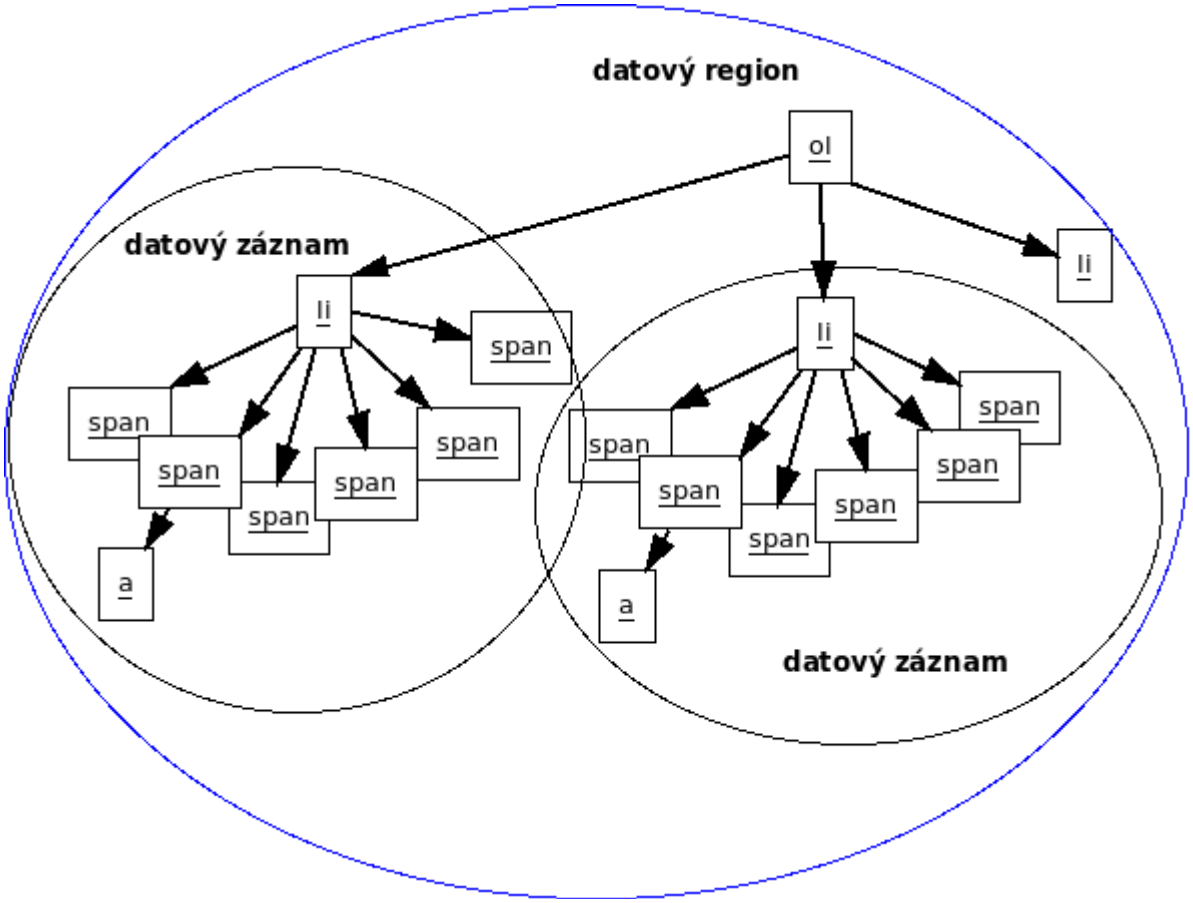
Obrázek 4.4 HTML kód dobře strukturovaného webového dokumentu (vztahuje se k obrázku 4.3)

Na uvedeném příkladu je plně zřetelná struktura představující datový region – oddělovačem je v tomto případě nadpis „Publications since 1995“ ohraničený elementem `<h2>` a datové záznamy jsou v číslovaném seznamu (element `<ol>`), každý záznam je ohraničen elementem `<li>`. Tento případ ilustruje obrázek 4.5. Datový záznam v tomto případě představuje posloupnost značek:

`li, span, span, a, span, span, span, span`

Jak již bylo řečeno, nejvíce potřebné pro extrakci dat ze stránek je rozpoznání navigace (pro crawling[9]) a struktury obsahu stránky, kde se jedná nejčastěji o rozpoznání opakujících se sekvencí. Tyto sekvence nemusejí být nutně tabulky ohraničené značky `<table></table>` nebo seznamy `<ol>/<ul>` s položkami `<li>`, popř . definiční seznamy `<dl>` s položkami `<dt>` a `<dd>`. Seznamem může být jakákoliv sekvence stejných (často nesémantických) značek – tedy `<div>` a `<span>` v kombinaci s jinými značkami. Není proto možné spoléhat se na to, že stránka bude vytvořena přesně podle sémantiky HTML značek a je nutné tomuto faktu extrakci přizpůsobit.

Existuje mnoho přístupů, jak řešit problém rozpoznání datových regionů a záznamů na webových stránkách. Je možné z posloupnosti elementů vytvořit prefixový strom (tzv. prefix-tree) a získat z něj opakující se sekvence nebo jiným „přesným“ algoritmem rozpoznat opakující se struktury z HTML značek. Tyto přístupy však naprosto selhávají na stránkách, na kterých jsou seznamy vytvářeny ručně a nepravidelně. Často totiž seznam má strukturu záznamů velmi podobnou – liší se např. v jednom elementu nebo je jeden element přidán; autor potřebuje přidat nějakou informaci nebo naopak některou upravit a proto dochází k této změně. I tak nepatrná změna je pro algoritmy vyvozující z přesné struktury HTML stromu nežádoucí a výsledek je velmi špatný. Naopak na webových stránkách, kde jsou seznamy generovány z databáze pomocí skriptovacího jazyka (cyklus vytiskne vždy stejnou sekvenci) jsou tyto algoritmy velmi přesné a dobře fungující. Tyto stránky jsou bohužel statisticky v menšině a proto je nutné se zaměřit i na špatně strukturované stránky.



Obrázek 4.5 Struktura diskutované webové stránky s publikacemi a rozdělení datového regionu na záznamy.

9 proces automatizovaného stahování webových stránek na základě odkazů nalezených na již stažených stránkách. Využívá se ve webových vyhledávačích pro hledání, třídění a indexaci webových stránek.

Pro účely extrakce dat v systému ReReSearch je nutné zpracovat co největší procento webových stránek, ze kterých je třeba získat informace. Proto bylo nutné navrhnout a vytvořit algoritmus, který úspěšně zpracuje i nepřesné struktury a správně rozpozná datový region skládající se ze „skoro stejných“ záznamů.

Je velmi vhodné si uvědomit způsob, jakým uživatel (člověk) rozpoznává tyto datové regiony a záznamy. Člověk vnímá webovou stránku graficky – to je nejdůležitější rozdíl oproti strojovém zpracování. Datové záznamy jsou pro člověka nejčastěji grafické struktury, které jsou vzájemně velmi podobné a mají podobu seznamu nebo tabulky. Naproti tomu datové regiony jsou člověkem nejčastěji rozpoznávány podle nadpisů nebo zvýrazněných textů nad souborem záznamů. Pokud by se podařilo vytvořit algoritmus tak, aby bral v potaz grafickou strukturu webové stránky, mělo by to velmi pozitivní vliv na úspěšnost vyhodnocení stránky a extrakce dat.

Algoritmus extrakce seznamů (opakujících se sekvencí) z webových stránek navržený pro knihovnu rrslib je založen na základě rozpoznání *stejných HTML značek v jedné úrovni* zanoření stromu objektového modelu dokumentu (DOM). Tyto značky (elementy) jsou označeny jako datové záznamy. Již ale není brána v potaz struktura nižších elementů daného záznamu (potomci elementu představující záznam) a tedy změna uvnitř záznamu algoritmus nijak neovlivní. Tím by měl být splněn požadavek na univerzálnost algoritmu a schopnost adaptability na hůře strukturované webové stránky.

Tento způsob získávání sekvencí však přináší několik problémů, které bylo nutno řešit:

1. opakované značky se mohou vyskytovat v několika úrovních – kterou sekvenci vybrat?
2. jak rozpoznat, zda se jedná o více datových regionů a nikoliv již o seznam záznamů?
3. jak neztratit informaci o struktuře záznamu (některé informace mohou být důležitější než jiné)?
4. jak řešit, když je každý záznam „obalen“ elementem s jinou značkou?

Řešením prvního problému je aplikace dvou heuristik, které algoritmu pomáhají při výběru nejpravděpodobnějším sekvence představující datový záznam. První heuristikou je tzv. *koeficient počtu potomků elementu reprezentujícího záznam*. Tento parametr ve skutečnosti reprezentuje nejnižší možný počet záznamů v regionu a zároveň nejvyšší počet částí (tzv. *chunk* – viz níže) záznamu. Druhou heuristikou je *minimální délka textu v záznamu*. Tato heuristika je triviální, ale funguje velmi dobře. Podle typu stránky se dá totiž očekávat, že záznamy budou obsahovat určité množství textu: např. v seznamu publikací autora bude mít reference pravděpodobně alespoň 30 znaků. Tímto lze velmi dobře rozpoznat, zda se jedná o část záznamu nebo o záznam celý.

Druhý problém je řešen již sofistikovanějším způsobem; aby bylo možné rozpoznávat datové oblasti, bylo nutné zjistit na stránce nadpisy, které nejčastěji tyto oblasti uvozují. První ideou bylo sledovat HTML značky typu header: `<h1>` - `<h6>`, ale tato metoda se vzhledem k neukázněnosti webových vývojářů ukázala jako neúčinná. Proto byl zaveden pojem „*viditelnost textu*“ (angl. visibility) a byl navržen algoritmus získávání tohoto parametru pro každý element obsahující text. Algoritmus bere v potaz CSS styly (velikost, barvu, styl písma atp.) i vlastní sémantiku HTML značek, z čehož vypočte výslednou relativní viditelnost textu na webové stránce. Tento parametr je pak základem pro třetí heuristiku: určení nadpisů na stránce se děje na základě *minimální hodnoty viditelnosti nadpisů.* Nadpisem je text s vysokou hodnotou viditelnosti (dáno koeficientem) a podle něj je určen příslušný datový region.

Kromě informace o datových regionech a záznamech lze ještě samotný záznam strukturovat na části, které často představují nějakou entitu. Na obrázku 4.4 jsou dobře znatelné tyto části v elementech `<span>` - jsou to jména autorů, rok vydání atp. Velmi často jsou v samotných záznamech některé části zvýrazněné jinou barvou, stylem případně dekorací písma a představují viditelnější a často

důležitou informaci. Pro extrakci dat je toto implicitní rozdělení autorem velmi důležité, protože již není třeba náročně rozpoznávat entity v textu.
Poslední situace, kdy stránka je tvořena ručně a každý záznam je „obalen“ jinou značkou a stránka je celkově nestrukturovaná, není v rámci navrženého algoritmu řešitelná. Často tyto stránky nejsou graficky na dobré úrovni a struktura záznamů není ani člověku patrná. Automatická extrakce na těchto stránkách sice selhává, nutnost extrakce dat na těchto stránkách je však diskutabilní.

Algoritmus implementovaný v knihovním modulu rrslib.web.sequencewrapper by tedy zmíněnou strukturu z obrázku 4.4 rozpoznával takto:

1. parsování kaskádových stylů (CSS) na webové stránce
2. výpočet parametru „viditelnost“ pro každý element HTML obsahující text
3. nalezení nejvíce stejných značek v jedné úrovni - ve třetí úrovni (na obrázku 4.5) značka `<span>`.
4. aplikací první heuristiky (koeficient synů elementu) se úroveň zanoření posune o jednu výše
5. elementy `<li>` a jejich potomci jsou nyní považováni za nositele dat záznamu
6. odstraněny jsou záznamy které jsou příliš krátké (pravděpodobně špatně rozpoznané)
7. aplikací druhé heuristiky jsou označeny nejviditelnější texty na stránce za nadpisy a veškeré záznamy za nimi jsou spolu s nadpisem považovány za datový region
8. jsou rozpoznány části záznamů – elementy `<span>` pod každým záznamem
9. při heuristické analýze výsledku jsou pak porovnány poměry region:záznam:část a podle toho je celý algoritmus iterován od bodu (4) s upravenými koeficienty.

Použití modulu je demonstrováno následujícím příkladem:

```
from rrslib.web.sequencewrapper import *
from rrslib.web.crawler import GetHtmlPage
# inicializace stahovace a extraktoru
downloader = GetHtmlPage()
seq = HTMLSequenceWrapper(childcoef=6.0, headercoef=4.0, mintextlen=30)
# url stranky, kterou chci parsovat
url = "http://www.chimie.ens.fr/hynes/publications.php"
# stazeni stranky
if not downloader.get_page(url)[0]:
    print "Chyba pri stahovani stranky."
    exit()
# navraceni stazene stranky rozparsovane pomoci lxml
element_tree = downloader.get_etree()
# extrakce: vstupem je lxml.etree.ElementTree stazene stranky, vystupem
# objekt ParsedHTMLDocument s regiony, zaznamy a chunky
document = seq.wrap_h(element_tree, url)
# vystup v xml
print seq.get_xml()
```

Výstup zpracování diskutované webové stránky modulem `sequencewrapper` ve formátu XML je přiložen v příloze B. Je zřetelné, že algoritmus popisuje celou strukturu dokumentu; navigaci, obsah rozčleněný na datové regiony a jejich záznamy včetně částí záznamů a jejich relativní viditelnosti na webové stránce. Tímto se z modulu `sequencewrapper` stává univerzální a robustní pomocník pro extrakci dat z webových stránek.

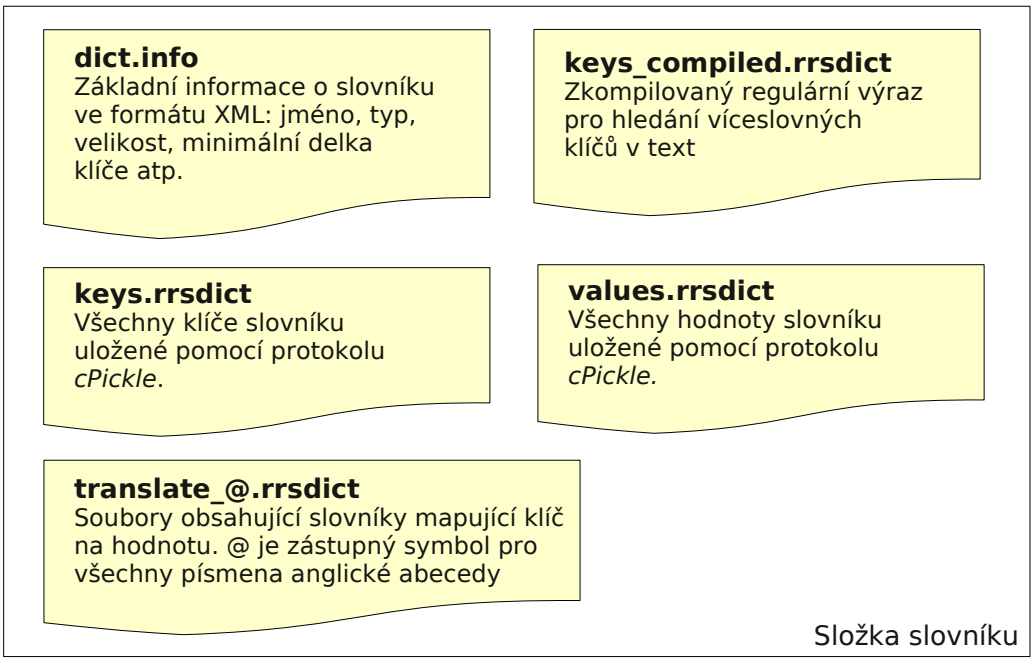
4.3.6 Slovníky pro extraktory v RRS – rrslib.dictionaries.rrsdictionary

Extrakce dat z lokálních dokumentů a z webových stránek je v systému ReReSearch založena převážně na hledání vzorů, sekvencí nebo základních částí dokumentů, které jsou dále zpracovávány dalšími extrakčními algoritmy. Dále se vyskytuje extrakce založená na regulárních výrazech a

vybraných klíčových slovech. Velmi výraznou pomocí extrakčním algoritmům pak mohou být specializované slovníky, které obsahují buď seznam názvů entit, které extraktor hledá jako klíčová slova a nebo slovníky mapující některé klíče na asociované hodnoty popř. na jejich seznam.

Ve své prvotní podobě tyto slovníky pro extraktory existovaly ve formě textového souboru, který byl načten do operační paměti, prohledávání hodnot slovníku bylo sekvenční a neexistovaly metody pro vyhledávání klíčů slovníku v textu. Tento přístup byl velmi neefektivní a použití tohoto typu slovníků bylo pro reálnou extrakci dat z textu z časových důvodů téměř nemožné.

Modul rrslib.dictionaries.rrsdictionary představuje řešení tohoto problému: byl navržen nový formát uložení slovníků, nové rychlejší rozhraní pro přístup k hodnotám a také algoritmy pro vyhledávání klíčů slovníku v textu. Slovníky byly rozděleny na dva typu: seznamy a překladové slovníky. V některých případech není třeba překladového slovníku a stačí seznam entit nebo názvů vztahujících se k určitému tématu. Bylo vytvořeno aplikačně-programové rozhraní pro používání těchto nových slovníků tak, aby vyhovovalo požadavkům na jednoduché použití a funkčnost. Existuje jedna základní třída `RRSDictionary`, která obsahuje veškeré metody pro získávání dat ze slovníku. Dále v modulu jsou přítomny konstanty představující jednotlivé slovníky.

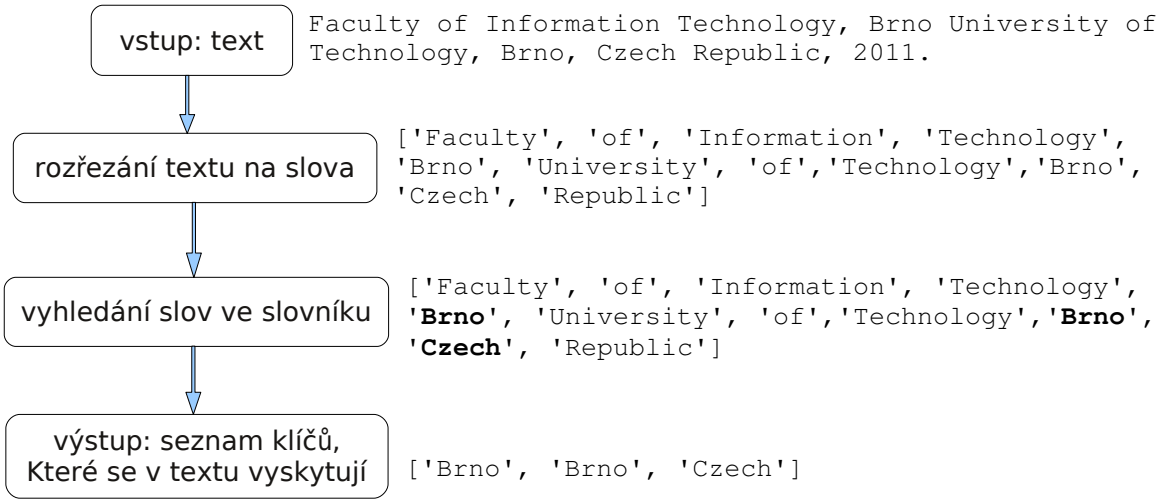


Obrázek 4.6 Architektura slovníků v knihovně rrslib

Implementace uložení slovníků je založena na serializaci objektů do souboru, kterou zajišťuje knihovna *cPickle*. Slovník a jeho soubory jsou uloženy ve složce, jejíž název reprezentuje jméno slovníku. Architektura této složky a obsažených souborů je zřetelná ze obrázku 4.6.: složka obsahuje soubor se statickými informacemi o slovníku, dále seznamy klíčů, popř. hodnot uložené pomocí protokolu pickle, zkompilované regulární výrazy a v případě překladového slovníku soubory translate_a.rrsdict – translate_z.rrsdict obsahující slovníky klíčů začínající na dané písmeno. Tyto slovníky jsou rozděleny kvůli úspoře dat načítaných do paměti. Při inicializaci slovníku nejsou načtena do operační paměti žádná data a teprve v případě potřeby jsou načteny klíče a parciální slovníky (translate_@.rrsdict) jen od toho písmena, které je aktuálně třeba. Často je totiž třeba vyhledat jen několik slov a u jednoduchých extrakčních algoritmů je velká pravděpodobnost, že celý slovník nebude využit.

Vyhledávání klíčů (metoda `contains_key()`) a hodnot (metoda `contains_value()`) ve slovníku je implementováno binárním vyhledáváním. Překlad klíče na hodnotu je dán implementací vestavěných pythonovských slovníků (dict), které jsou implementovány jako dvouúrovňové hashovací tabulky.

Vyhledávání jednoduchých (jednoslovných) klíčů v textu (metoda `text_search()`) bylo implementováno také pomocí *binárního vyhledávání*. Víceslovné klíče není možné vyhledávat tímto způsobem a proto je vyhledávání implementováno pomocí regulárních výrazů, které jsou ve složce slovníku přítomny. Tento způsob je z hlediska výkonu neefektivní a bude velmi vhodné tento problém později řešit a vytvořit algoritmus založený na bisekci, který by tento případ řešil rychleji.



Obrázek 4.7 Algoritmus hledání klíčů slovníku v textu pomocí dělení textu a binárního vyhledávání. V tomto případě je uvažován slovník s geografickými názvy (měst, států atp.)

Bylo provedeno testování jednotlivých metod vyhledávání klíčů a hledání v textu. Testování probíhalo na stroji s procesorem Intel Core2Duo 2,53GHz a s 2GB paměti. Tabulka 4.1 obsahuje srovnání těchto metod a jejich parametrů:

n – počet klíčů slovníku
k – počet slov v textu, ve kterém jsou hledány slovníkové klíče

metoda	význam	implementace	složitost	test [s]
contains_key()	hledání klíče	binární vyhledávání	$O(\log_2(n))$	$4{,}36 * 10^{-5}$
text_search()	hledání klíčů v textu	binární vyhledávání	$O(k*\log_2(n))$	$5{,}14 * 10^{-4}$
text_search()	hledání klíčů v textu	regulární výrazy	$O(k)$	$2{,}79 * 10^{-3}$
translate()	překlad klíče na hodnotu	hashovací tabulka	$O(1)$	$7{,}91 * 10^{-5}$

Tabulka 4.1 Srovnání metod pro vyhledávání ve slovnících

Uvedené hodnoty byly naměřeny na překladovém slovníku o velikosti 33187 klíčů. V případě hledání v klíčů v textu (metoda `text_search()`) obsahoval text vždy 10 slov, z nichž některá byla obsažena ve slovníku. Velmi zajímavé jsou výsledky vyhledávání klíčů v textu; přestože teoretická časová náročnost hledání s využitím regulárních výrazů je lineární, je binární vyhledávání při dané velikosti slovníku násobně rychlejší.

Naměřené hodnoty jsou z teoretického hlediska uspokojivé, avšak v kontextu extrakce dat jsou stále velmi vysoké. Velmi významnou položkou v časové složitosti, která v tabulce není uvedena, je doba načítání regulárních výrazů ze souboru. Bylo změřeno, načítání regulárního výrazu trvá až 2,89 sekundy. Pokud jsou připočteny desítky až stovky volání metody `text_search()` v některých extraktorech, je zjevné, že se jedná již o časově velmi náročné procesy. Také proto by bylo vhodné modul rrslib.dictionaries.rrsdictionary vhodné experimentálně implementovat v C (s použitím Python/C API), odstranit používání regulárních výrazů a výkonnostně srovnat s existujícím řešením.

4.3.7 Klasifikace jazyka dokumentu – rrslib.classifiers.language

Jako příklad klasifikátoru poskytovaného knihovnou byl vybrán modul rrslib.classifiers.language, který slouží pro identifikaci jazyka textu. V systému ReReSearch se extraktory často pohybují v multilingválním prostředí, kterým je například web. Často je tedy třeba rozpoznat jazyk webové stránky nebo textu a dle toho uzpůsobit další extrakční procedury. Pro podporu tohoto rozhodování byl vytvořen klasifikátor `LanguageIdentifier`, který obsahuje algoritmus klasifikace textu a pravděpodobnostní rozhodování. Součástí modulu jsou také modely 14 evropských a světových jazků.

Existuje mnoho možností a algoritmů, jak klasifikovat jazyk textu. Nejčastějším přístupem je vytvoření pravděpodobnostního modelu jazyků a poté vytvořit stejným algoritmem model textu a poté některou z klasifikačních metod porovnat dané modely. Pro vytvoření modelu se mezi jinými nejvíce vyskytují techniky založené na počítání N-gramů - shluků písmen, částí slov (tzv. N-gram counting), metoda založená na nejčastějších slovech v jazyce (tzv. Most Common Words) nebo technika vycházející z identifikace unikátních shluků písmen, které se v jiných jazycích nevyskytují (T. Dunning, [15]).

Možností klasifikace pravděpodobnostních modelů je také celá řada. Nejpoužívanějšími metodami jsou Markovovy modely, rozhodování na základě aplikace Bayesovské teorie nebo techniky založené na relativní entropii a sdílené vazební informaci ([16], str. 5). Další nepříliš používanou technikou je metoda Monte Carlo (tamtéž, str. 6), která byla použita v implementaci knihovního modulu rrslib.classifiers.language.

Jako model jazyka byl zvolen pravděpodobnostní model využívající nejčastější slova v jazyce - tzv. *Most Common Words*. Těmito slovy jsou nejčastěji předložky. Klasifikace je provedena metodou *Monte Carlo*: jsou vybírány náhodné vzorky (slova) a ty jsou klasifikovány pro každý jazyk zvlášť. Normalizovaná suma pravděpodobností potom odpovídá pravděpodobnosti, se kterou je daný text napsán v daném jazyce. Výhoda tohoto přístupu je v tom, že výpočetní složitost je konstantní vůči velikosti textu, tedy *O(1)*. Počet vzorků vybíraných metodou Monte Carlo je omezen shora na 5000 a zdola na počet písmen textu. Tímto je způsobem je zajištěno, že čas pro klasifikaci jazyka textu se pohybuje řádově ve desítkách až stovkách milisekund. Modul poskytuje také programové rozhraní pro získání dodatečných informací o jazyku, jako např. nativní jméno nebo zkratky jazyků definované standardem ISO 639-1 a ISO 639-3 [17].

5 Dokumentace

Pro účely použitelnosti musí být každá knihovna vybavena kvalitní dokumentací. Tato dokumentace je specifická: největší část obsahuje popis aplikačně-programového rozhraní a malá část je většinou věnována pozadí vývoje knihovny, požadavky na instalaci a použití, popř. další instrukce.

Nejinak je tomu v případě knihovny rrslib. Dokumentace byla vytvořena na interní wikipedii výzkumné skupiny NLP Group [18]. Tato dokumentace obsahuje:

- obecné informace o knihovně (umístění, odkazy ke stažení apod.)
- tutoriál pro zprovoznění knihovny
  - zprovoznění na různých typech OS
  - řešení případných problému při instalaci knihovny
  - první kroky při používání
- pokyny pro vývojáře knihovny
  - práce s repozitářem git
  - balíčkování
  - testování knihovny, automatizace
- schéma knihovny
- tematicky rozčleněné sekce knihovny
  - popis funkcionality modulů
  - aplikačně-programové rozhraní (API)
  - příklady použití (ukázky kódu v jazyce Python)
  - řešení problémů

V dokumentaci aplikačně-programového rozhraní jsou vždy uvedeny dostupné třídy a metody dané knihovny včetně parametrů a ke každé důležité funkci (metodě) je přidáno několik příkladů kódu demonstrujících její vlastnosti a použití. Forma dokumentace založená na příkladech se ukázala jako velmi vhodná i pro studenty, kteří s jazykem Python teprve začínají.

Kromě ručně psané dokumentace je k dispozici také automaticky generovaná dokumentace programem *epydoc* [10]. Tento program vytváří na základě vstupních zdrojových kódu webový portál (na výstupu je série HTML stránek), který je velmi přehledný a propracovaný. Aby byla tato dokumentace přístupná i studentům, kteří nejsou členy výzkumné skupiny, byla umístěna na osobní stránky na serveru *merlin* [19].

6 Testování

Každý softwarový produkt, tedy i programová knihovna, při velkém množství změn a dlouhém vývoji potřebuje systém testování. Tento systém nejen že odhalit problémy a chyby již při vývoji, ale zvláště při opravách a změnách ve zdrojových kódech později po uvedení výrobku na trh. Pokud jsou tyto testy kvalitně vytvořené, dokáží odhalit například chyby, které vznikly při opravě jiné chyby. Při velkém množství zdrojových kódu a ve složitých systémech je pak testování neodmyslitelnou částí vývoje každého softwarového produktu. Také proto firmy specializující se na vývoj software investují značné prostředky do oddělení pro testování (tzv. Quality Assurance), které udržuje produkt na požadované úrovni.

6.1 Git pro testy

Testy modulů knihovny rrslib jsou skripty v jazyce Python, které testují jednotlivé aspekty algoritmů a případy použití knihovny. Protože i testy mohou obsahovat chyby a mohou se měnit, protože se mění knihovní modul, bylo nutné zavést i zde systém správy verzí – tedy opět git. Složka s repozitářem rrslib se nachází na serveru minerva1 na sdíleném diskovém poli přístupném ze serverů athéna, minerva, merlin a z počítačů pcnlp ve složce projektu *rrs_library*.

Architektura repozitáře je velmi jednoduchá – veškeré testy jsou umístěny do jedné složky „tests“, odkud jsou spouštěny. Vzhledem k současnému i očekávanému množství knihovních modulů není třeba vytvářet složitější adresářové struktury.

6.2 Implementace unit testů

Testy knihovny rrslib jsou klasické unit-testy, každý se zaměřuje na jeden modul knihovny. Bohužel, knihovna nebyla vytvářena systémem test-driven development, protože bylo nutné implementovat v relativně krátkém čase mnoho modulů a uvést je do provozu. V současné době je napsána necelá polovina potřebných testů pro knihovnu rrslib. Tento fakt je odůvodněn nemožností obecně testovat některé heuristické algoritmy a také tlak na další vývoj podpůrných modulů namísto implementace testů.

6.3 Automatizace testování

Vzhledem k malým rozměrům tohoto projektu není možné zavést rozsáhlejší systém testování. Důvodů je hned několik – neexistují potřební lidé, kteří by psali, opravovali a vyhodnocovali testy a není k dispozici ani dostatečné zázemí pro řízení tohoto procesu. Není proto možné vytvářet složitější struktury softwarového inženýrství jako např. tzv. testplány. Je také velmi diskutabilní, jestli na tak malém projektu by nebylo zavádění podobných struktur přínosné. Vhodnějším způsobem testování se jeví jednoduchá automatizace na bázi existujících unit-testových skriptů, které budou pravidelně automaticky spouštěny. K testování knihovny je použit testovací framework nose. Byl vytvořen skript, který spustí testování poslední revize knihovny a výstup ukládá do XML souborů, které lze pak automaticky načíst na nlp-wiki.

7 Závěr

V rámci této bakalářské práce byla navržena a vytvořena knihovna pro podporu vývoje systému ReReSearch. Z hlediska podpory se knihovna zaměřuje na extrakční a klasifikační algoritmy, podporu práce s webem, databází a s formátem RRS-XML. Vytvořením a použitím knihovny byla zlepšena kvalita a úspěšnost některých modulů v rámci systému RRS. Díky knihovně byly zprovozněny některé důležité procesy v systému RRS: extrakce dat z lokálních dokumentů, import extrahovaných dat do databáze a mnoho dalších. Knihovna je používána také jako základ některých bakalářských prací a projektů v rámci výzkumné skupiny NLP Group.

Knihovna rrslib byla navržena s perspektivou dalšího vývoje, byly tedy založeny prostředky pro správu verzí, testování a jeho automatizace, balíčkování a opět jeho automatizace. Na nlp-wiki byla vytvořena dokumentace a tutoriál pro zprovoznění knihovny, popis API jednotlivých modulů a také pokyny pro vývojáře knihovny.

7.1 Možnosti dalšího vývoje knihovny

Současná verze knihovny rrslib obsahuje mnoho chyb a nedostatků. Převážná část těchto chyb lze vyřešit vytvořením *kvalitnějších testů* a pravidelným testováním během vývoje. V knihovně ale existuje také několik chyb vycházejících z nekvalitního návrhu, často způsobených změnou prostředí nebo požadavků na knihovní moduly (např. modul `entityextractor` je navržen převážně pro extrakci entit z referencí, ale není obecně použitelný pro extrakci z volného textu). Aby bylo možné tyto chyby odstranit, je třeba definovat nové cíle a účely modulu, vytvořit nový návrh a důsledně přepracovat modul tak, aby byl zpětně kompatibilní se staršími systémy a zároveň poskytoval funkcionalitu, která byla vytvořena novým návrhem a zpracováním.

Značný prostor pro zlepšení je v implementaci slovníků a slovníkových algoritmů. Hlavním cílem je zrychlení procesu načítání slovníku do paměti a vyhledávání víceslovných klíčů ve volném textu. Bude vhodné experimentálně implementovat část slovníkových algoritmů v jazyce C a porovnat výkonnost obou nového a současného řešení.

Knihovní moduly pro podporu extrakce dat z textu a z lokálních dokumentů by měly být uzpůsobeny pro obecnější úkoly. Bude třeba vytvořit lépe strukturované algoritmy pro rozpoznávání a extrakci entit z textu a zobecnit některé algoritmy v oblasti normalizace názvů entit.

Vývoj knihovny v oblasti extrakce dat z webových stránek by měl směřovat cestou univerzálního základu pro webové extraktory. Jistě by bylo vhodné rozšíření aktuálního modulu rrslib.web.htmltools tak, aby bylo možné z parsované stránky zjistit i typ webové stránky. Pro tento účel bude třeba vytvořit klasifikátor, který bude rozlišovat typy webových stránek, které jsou zajímavé z hlediska systému ReReSearch. Těmito typy budou pravděpodobně kategorie:

- webová stránka konference
- osobní stránka autora
- stránka fakulty
- stránky univerzity
- stránka se seznamem publikací daného autora
- stránka se seznamem publikací z fakulty/univerzity

- stránka jedné publikace
- domovská stránka projektu
- stránka s výzkumnými zprávami projektu (tzv. deliverables)
- a jiné

Jisté vylepšení by bylo třeba i v parsování CSS, protože aktuální implementace CSS parseru nedokáže zpracovat kontextuální selektory ani selektory pro výběr předků nebo potomků kaskádového stylu. Bylo by vhodné do parametru *viditelnosti* textu na webové stránce počítat také kontrast barvy textu vůči pozadí a jiné detaily.

V oblasti podpory databáze by měla knihovna poskytovat kompletní ORM vrstvu. Je tedy třeba implementovat funkce pro načítání dat z databáze a vytvořit filtry pro tento výběr. Bylo by také vhodné zavést možnost přidávání vlastních tříd modelu pro aktuální běh ORM vrstvy. Tuto funkci by využívali převážně studenti pracující na bakalářských a diplomových pracích, které se zabývají rozšířením současné databáze RRS. Tímto by byla studentům poskytnuta možnost používat knihovní funkce bez změny modelu databáze, který je aktuálně používán v systému ReReSearch. Začnou pozornost si také zaslouží model databáze, který by měl v budoucnu poskytovat informaci o integritních omezení.

V knihovně prozatím úplně chybí modul s funkcemi podporující klientskou stranu řídícího systému. Tento modul by měly importovat všechny moduly systému RRS a měly by používat jeho funkcí pro komunikaci s řídícím systémem. Sjednocením aplikačně-programového rozhraní XML-RPC serveru do jednoho modulu bude zjednodušena případná změna nebo upgrade rozhraní.

Literatura

[1] CiteSeerX [online]. 2010 [cit. 2011-04-15]. Scientific Literature Digital Library and Search Engine. Dostupné z WWW: <http://citeseerx.ist.psu.edu/>.

[2] LEY, Michael. DBLP [online]. 2011 [cit. 2011-04-15]. The DBLP Computer Science Bibliography. Dostupné z WWW: <http://www.informatik.uni-trier.de/~ley/db/>.

[3] Google Scholar [online]. 2011 [cit. 2011-04-15]. Google Scholar. Dostupné z WWW: <http://scholar.google.cz/>.

[4] Microsoft Academic Search [online]. 2011 [cit. 2011-04-15]. Dostupné z WWW: <http://academic.research.microsoft.com/>.

[5] DLOUHÝ, Ivo. *Řídicí systém projektu ReReSearch*, bakalářská práce, Brno, FIT VUT v Brně 2010, str 14., 18. Dostupné z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=10577>.

[6] Webové rozhraní systému ReReSearch [online]. 2011 [cit. 2011-03-24]. Dostupné z WWW: <http://athena3.fit.vutbr.cz:30101/databrowser/www/>.

[7] Python Programming Language [online]. 2011 [cit. 2011-03-24]. Dostupné z WWW: <http://www.python.org/>.

[8] Python v2.7.1 documentation [online]. 2.7. 24.3.2011 [cit. 2011-03-24]. *Python/C API Reference Manual*. Dostupné z WWW: <http://docs.python.org/c-api/>.

[9] VAN ROSSUM, Guido; WARSAW, Barry. Python Developer's Guide [online]. 5.7.2001 [cit. 2011-03-24]. *PEP 8 - Style Guide for Python Code*. Dostupné z WWW: <http://www.python.org/dev/peps/pep-0008/>.

[10] Epydoc [online]. 2008 [cit. 2011-03-24]. *Automatic API Documentation Generation for Python*. Dostupné z WWW: <http://epydoc.sourceforge.net/>.

[11] LOKAJ, Tomáš. *Extrakce metadat z vědeckých článků*, bakalářská práce, Brno, FIT VUT v Brně. 2010. Dostupné z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=10474>.

[12] SQL Power [online]. 2010 [cit. 2011-03-26]. *Data Modeling & Profiling Tool: SQL Power Architect*. Dostupné z WWW: <http://www.sqlpower.ca/page/architect>.

[13] GRUNDL, David. Dibi [online]. 2011 [cit. 2011-03-26]. *Dibi is Database Abstraction Library for PHP 5*. Dostupné z WWW: <http://dibiphp.com/cs/>.

[14] HYNES, James T. Personal homepage of James.T. Hynes [online]. 2011 [cit. 211-03-26]. *Hynes Group Publication List.* Dostupné z WWW: <http://www.chimie.ens.fr/hynes/publications.php>.

[15] DUNNING, Ted. *Statistical Identification of Language*. [online]. 10.3.1994, [cit. 2011-03-24]. Dostupný z WWW: <http://citeseer.ist.psu.edu/viewdoc/download?doi=10.1.1.48.1958&rep=rep1&type=pdf>.

[16] POUSTMA, Arjen. *Applying Monte Carlo Techniques to Language Identification*. [online]. 2001, [cit. 2011-03-24]. Dostupný z WWW: <http://www.xs4all.nl/~ajwp/langident.pdf>.

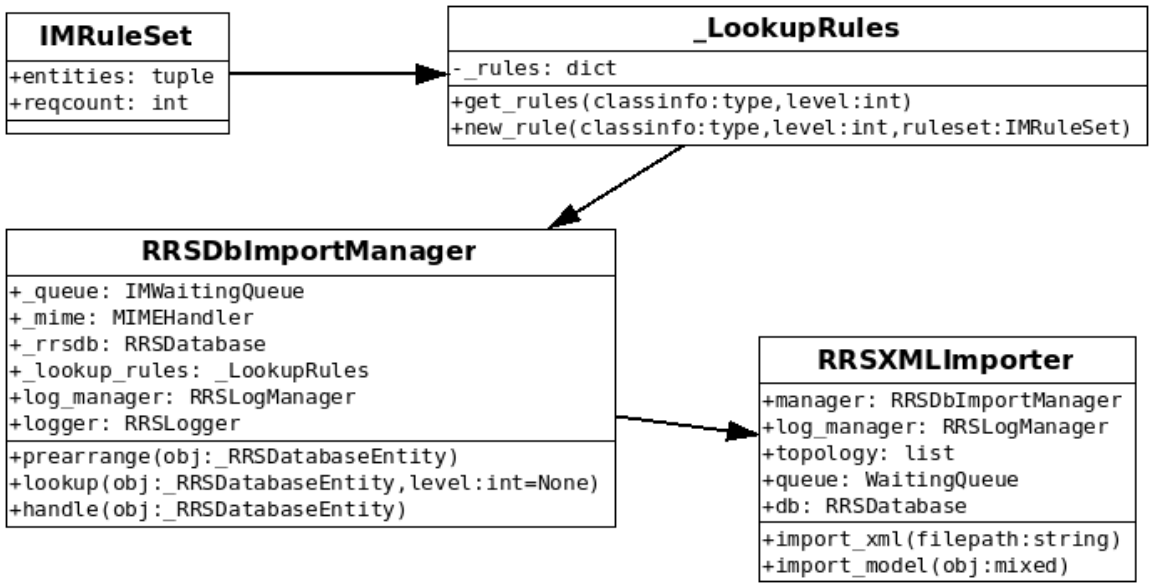
[17] ISO 639-3:2007. *Codes for the representation of names of languages* [online]. 2007, 3, [cit. 2011-03-26]. Dostupný z WWW: <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39534>.

[18] HELLER, Stanislav. *Dokumentace knihovny RRS library* [online]. 2011 [cit. 2011-03-29]. Dostupné z WWW: <https://merlin.fit.vutbr.cz/nlp-wiki/index.php/Rrs_library>.

[19] HELLER, Stanislav. *RRS Library API Documentation* [online]. 2011 [cit. 2011-03-29]. Dostupné z WWW: <http://www.stud.fit.vutbr.cz/~xhelle03/nlp/rrslibdoc/>.

Příloha A

Diagram základních tříd modulu rrslib.db.xmlimport.



Příloha B

Výstup modulu rrslib.web.sequencewrapper ze stránky http://www.chimie.ens.fr/hynes/publications.php ve formátu XML:

```
<?xml version='1.0' encoding='utf-8'?>
<document base="http://www.chimie.ens.fr/hynes/publications.php"
title="Hynes Group Publication List">
  <menu/>
  <sequence-area>
    <header visibility="52.4908">Publications since 1995</header>
    <entry>
      <text>Arnulf Staib, James T. Hynes and Daniel Borgis, "Proton
Transfer in Hydrogen-Bonded Acid-Base Complexes in Polar Solvents", J.
Chem. Phys. 102, 2487-2505 (1995)</text>
      <chunks>
        <chunk visibility="1.0">Arnulf Staib, James T. Hynes and Daniel
Borgis</chunk>
        <chunk visibility="1.5"
link="http://www.chimie.ens.fr/hynes/publications/staib.1995.jcp.102.2487.
pdf">"Proton Transfer in Hydrogen-Bonded Acid-Base Complexes in Polar
Solvents"</chunk>
        <chunk visibility="2.0">J. Chem. Phys.</chunk>
        <chunk visibility="2.3303">102</chunk>
        <chunk visibility="1.0">2487-2505</chunk>
        <chunk visibility="1.0">(1995)</chunk>
      </chunks>
    </entry>
    <entry>
      <text>Roberto Bianco and James T. Hynes, "VB Resonance Theory in
Solution. I. Multi-state Formulation", J. Chem. Phys. 102, 7864-7884
(1995)</text>
      <chunks>
        <chunk visibility="1.0">Roberto Bianco and James T. Hynes</chunk>
        <chunk visibility="1.5"
link="http://www.chimie.ens.fr/hynes/publications/bianco.1995.jcp.102.7864
.pdf">"VB Resonance Theory in Solution. I. Multi-state Formulation"</chunk>
        <chunk visibility="2.0">J. Chem. Phys.</chunk>
        <chunk visibility="2.3303">102</chunk>
        <chunk visibility="1.0">7864-7884</chunk>
        <chunk visibility="1.0">(1995)</chunk>
      </chunks>
    </entry>
    ...
    ...
  </sequence-area>
</document>
```

Příloha C

Metriky kódu knihovny rrslib. Tyto metriky se vztahují k odevzdané verzi knihovny; v repozitáři gitu bude pravděpodobně k dispozici nová verze s jinými metrikami.

Počet řádků zdrojového kódy knihovny rrslib v jazyce Python: 25081
Počet řádků zdrojového kódu obslužných skriptů: 276
Velikost archivu s knihovnou a instalačními skripty (tar.gz): 35,436 MB

Příloha D

Obsah přiloženého CD.

src-lib/	Zdrojové soubory knihovny rrslib
src-dist/	Zdrojové soubory skriptů pro tvorbu a údržbu balíčků
dist/	Balíčky pro instalaci rrslib (tar.gz, RPM)
doc-lib/	Elektronická verze dokumentace knihovny generovaná programem epydoc
doc-thesis/	Zdojový soubor (.odt) a PDF verze bakalářské práce
README	Stručný popis obsahu CD