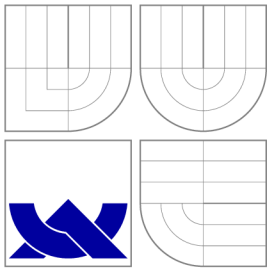
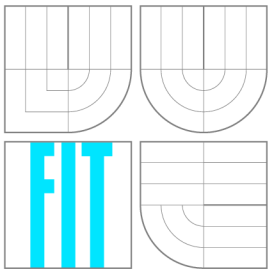


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

RÁMEC PRO EXTRAKCI INFORMACE Z WWW

FRAMEWORK FOR INFORMATION EXTRACTION FROM WWW

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. FILIP BRYCHTA
AUTHOR

VEDOUCÍ PRÁCE — Ing. RADEK BURGET, Ph.D.
SUPERVISOR

BRNO 2009

Zadání práce

1. Seznamte se s technologiemi pro publikaci informací na WWW.
2. Prostudujte existující metody extrakce informace z WWW dokumentů.
3. Navrhněte modulární systém pro extrakci strukturovaných záznamů z HTML dokumentů a jejich další zpracování.
4. Implementujte navržený systém.
5. Implementujte moduly pro ukládání dat do relační databáze a XML.
6. Po dohodě s vedoucím implementujte modul realizující zvolenou metodu extrakce informace.
7. Zhodnoťte dosažené výsledky a navrhněte pokračování projektu.

Abstrakt

Prostředí webu se postupně vyvinulo v nejrozsáhlejší zdroj dokumentů v elektronické podobě, takže by bylo velice výhodné, informace v těchto dokumentech zpracovávat automaticky. To však není jednoduchý úkol, protože většina dokumentů je napsána v HTML (Hypertext Markup Language), který neumožňuje definovat sémantiku dat v těchto dokumentech. Cílem této práce je vytvořit modulární systém pro extrakci informací z HTML dokumentů a jejich další zpracování. Dalším zpracováním se myslí ukládání získaných informací například do XML souboru nebo do relační databáze. Modularita systému umožňuje využití různých extrakčních metod a různých metod pro uložení získaných dat. Díky tomu je systém použitelný pro mnoho různých úloh.

Abstract

Web environment has developed into the largest source of electronic documents, so it would be very useful, to process this information automatically. This is however not a trivial problem. Most documents are written in HTML (Hypertext Markup Language), which does not support semantic description of the content. The goal of this work is to create modular system for information extraction and further processing of this information from HTML documents. Further processing of information means to store this information in XML document or relational database. System modularity makes it possible to use various information extraction and storing methods, thus the system can be used for various tasks.

Klíčová slova

extrakce informací, wrapper, World Wide Web, XML, HTML, detekce znakové sady, java class loader

Keywords

information extraction, wrapper, World Wide Web, XML, HTML, charset detection, java class loader

Citace

Filip Brychta: Rámec pro extrakci informace z WWW, diplomová práce, Brno, FIT VUT v Brně, 2009

Rámec pro extrakci informace z WWW

Prohlášení

Prohlašuji, že jsem tento semestrální projekt vypracoval samostatně pod vedením pana Ing. Radka Burgeta, Ph.D.

.....

Filip Brychta

25. května 2009

Poděkování

Chtěl bych tímto poděkovat vedoucímu práce Ing. Radku Burgetovi, Ph.D, za cenné připomínky a pomoc při řešení problémů.

Obsah

Kapitola 1

Úvod

Extrakce informací z WWW je v dnešní době stále velkou výzvou. Prostředí webu se postupně vyvinulo v nejrozsáhlejší zdroj dokumentů v elektronické podobě, takže by bylo velice výhodné informace, obsažené v těchto dokumentech, zpracovávat automaticky. Tyto dokumenty jsou dostupné v mnoha různých formách, nejčastěji však jako HTML. HTML (HyperText Markup Language) je značkovací jazyk bez pevně dané struktury, což znesnadňuje automatické zpracování počítači. Data v takovýchto dokumentech postrádají sémantiku, tu si k nim přiřazuje až člověk. Snahou o odstranění tohoto nedostatku je například sémantický web, kde jsou data ukládána a strukturována podle standardizovaných pravidel. Sémantický web se ale zatím nedočkal většího rozšíření. I přes snahy o vytvoření alternativ k HTML, se stále počet HTML dokumentů velmi výrazně zvyšuje. Z tohoto důvodu je nutné mít k dispozici technologie, které umožňují automatické zpracování informací obsažených v takovýchto dokumentech.

Automatické zpracovávání informací z webu by ušetřilo mnoho času uživateli, který je dnes nucen procházet stránky manuálně. Každý by jistě ocenil aplikaci, která by dokázala sjednotit informace z různých míst na webu a zobrazila je v přehledné podobě například v jednom souboru XML nebo je uložila do databáze. Tohoto se snaží dosáhnout tzv. wrappery, které transformují vybrané části HTML dokumentu do XML formátu. Dokumenty ve formátu XML jsou mnohem vhodnější pro automatické zpracování. Na soubor XML dokumentů je možné pohlížet jako na databázi a pro dotazování využít dotazovacích jazyků jako jsou XML-GL [6], XML-QL [10] a XQuery [7].

Extrakce informací (získávání znalostí) se hojně využívá v databázových systémech. Ty mají velkou výhodu oproti HTML především v tom, že jsou strukturované a je možné využívat dotazovacích jazyků jako je SQL. V dnešní době je k dispozici obrovské množství dat a je nutné mít prostředky pro přeměnu těchto dat na užitečnou informaci. Tyto informace pak mohou být využívány například pro rozhodování analytiků apod. Techniky využívané pro získávání znalostí z databází je možné využívat i pro extrakci informací z WWW. Tato oblast zahrnuje mnoho dalších vědních oborů jako je například zpracování přirozeného jazyka, získávání znalostí z čistého textu, strojové učení apod.

Cílem této práce je vytvořit rámec pro extrakci informace z WWW. Jedná se o modulární systém, který umožní snadné používání různých extrakčních metod a ukládání získaných dat do XML souborů nebo do databáze. Jednotlivé extrakční metody představují moduly, které systém dynamicky načítá a spravuje. Dalším typem modulů, se kterými systém pracuje, jsou moduly starající se o ukládání získaných dat. Díky této modularitě se jedná o velice pružný systém, který je možné využít pro mnoho různých úloh.

První polovina tohoto textu se věnuje popisu používaných přístupů a technologií. Tato část má sloužit pro uvedení čtenáře do problematiky. Při psaní této části bylo čerpáno ze zdrojů uvedených na konci práce. První kapitola obsahuje úvod do extrakce informací. Jsou zde popsány přístupy využívané pro extrakci informací z databází, z čistého textu a z WWW. Druhá kapitola se věnuje popisu prostředí WWW a technologií spojených s WWW jako je HTTP, HTML, XHTML, CSS, sémantický web a tvorba dynamických stránek. Třetí kapitola rozšiřuje úvod do extrakce informací z WWW uvedený v první kapitole. Jsou zde podrobně popsány wrappery, jejich rozdělení a přístupy pro jejich automatickou tvorbu. Dále jsou zde popsány alternativní metody extrakce informací, které modelují logickou strukturu dokumentu. Druhá polovina textu se už věnuje vlastnímu systému. První kapitola z této části obsahuje popis systému a analýzu, kde jsou řešeny zásadní problémy. Druhá kapitola obsahuje návrh systému a třetí implementaci. Zhodnocení práce je uvedeno v závěrečné kapitole. Jako příloha je uveden manuál a příklady využití systému.

Tato práce navazuje na semestrální projekt, který řešil teoretickou část a základní návrh. Ze semestrálního projektu byly převzaty první čtyři kapitoly uvedené v této práci. Tyto kapitoly byly na mnoha místech rozšířeny. Základní návrh ze semestrálního projektu byl přepracován.

Kapitola 2

Úvod do extrakce informací

V dnešní době máme k dispozici obrovské množství dat a potřebujeme mít prostředky pro získání užitečných informací z těchto dat, ať už jsou uloženy v databázi, jako prostý text nebo jsou dostupné na webu v různých formátech. V následující části je popsán úvod do problematiky z pohledu klasických databází, čistého textu a WWW.

2.1 Získávání znalostí z databází

Získávání znalostí z databází lze chápat jako výsledek přirozeného vývoje databázové technologie [35]. Databázová technologie vznikala od 60. let minulého století. V té době vznikly síťové a hierarchické databáze. V 70. letech se začalo pracovat na nejúspěšnějším typu databází, čímž byly relační databáze, které se postupně prosadily po celém světě. V 80. letech se objevily další typy databází jako jsou objektově-relační databáze, objektové databáze a další a zároveň došlo k velkému nárůstu dat uložených v databázích. To vedlo v 90. letech k rozvoji datových skladů. Ty mají sloužit k shromažďování a přípravě dat pro podporu rozhodování. Datové sklady poskytují různé pohledy na data, agregační funkce a další funkce pro analýzu dat. Takováto analýza se označuje jako OLAP (Online Analytic Procesing).

Tento přístup stále nestačil což vedlo k rozvoji získávání znalostí z databází. Můžeme říci, že získávání znalostí z databází je extrakce (neboli dolování) zajímavých (netriviálních, skrytých, dříve neznámých a potenciálně užitečných) modelů dat a vzorů z velkých objemů dat. Tyto modely a vzory reprezentují znalosti získané z dat [35].

Mezi základní úlohy získávání znalostí z databází patří získávání asociačních pravidel, shluková analýza, klasifikace a predikce. Asociační pravidla se využívají například při analýze nákupního košíku. Úkolem je nalézt takové produkty, které se prodávají velmi často společně. Pokud tedy zákazník v e-shopu vloží do košíku nějaký produkt, automaticky mu jsou nabídnuty produkty z pravé strany asociačního pravidla. Takové pravidlo může vypadat následujícím způsobem: *kupuje(X,'myš')* $\Rightarrow$ *kupuje(X,'podložka pod myš') [podpora=2%, spolehlivost=60%]*. Takovéto pravidlo říká, že ve dvou procentech všech sledovaných nákupů zákazník nakoupil myš společně s podložkou pod myš a v šedesáti procentech případů, kdy

zákazník koupil myš, koupil také podložku pod myš. Dalším příkladem využití vydolovaných asociačních pravidel je rozmístění zboží v supermarketu (často kupované zboží vedle sebe nebo naopak na druhé straně prodejny). Další používanou metodou je shluková analýza, ta pracuje tak, že vytváří shluky prvků s podobnými vlastnostmi. Příkladem využití této metody může být cílená reklamní kampaň. Z nashromážděných dat o zákaznících a jejich nákupech se vytvoří shluky zákazníků s podobnými zájmy a pak je jím rozeslána nabídka produktů, které by je mohly zaujmout. Pokud proces obrátíme a soustředíme se na odlehlé hodnoty, můžeme snadno odhalit podezřelé transakce v bankovnictví a podobně. Úkolem klasifikace je zařadit vstupní data do určitých tříd (konečný počet diskrétních hodnot). Klasifikovat můžeme například elektronickou poštu, zda se jedná o spam, či nikoli. Dalším příkladem využití klasifikace může být odhad úvěruschopnosti žadatele o úvěr. Na základě získaných dat o žadateli je automaticky rozhodnuto, zda úvěr poskytnout, či neposkytnout. Další používanou metodou je predikce, ta slouží k odhadu nějaké spojité hodnoty. Využít lze například na burzách při odhadu cen akcií a podobně.

2.2 Dolování z textu

Motivace pro dolování z textu je jednoznačná. Existuje obrovské množství prostého textu v nejrůznějších podobách (knihy, časopisy, vědecké články, internetové diskuze apod.). Dostat se k těmto materiálům v elektronické podobě není velký problém. Problém nastává, pokud chceme z tohoto nestrukturovaného materiálu získat pouze ty části, které obsahují pro nás zajímavé informace.

Dolování z textu se oproti získávání znalostí z databází liší především v tom, že se nevyhledává v strukturovaných tabulkách, ale v prostém textu. Dalším velkým rozdílem je vysoká dimenzionalita rysů a řídkost rysů. Vysokou dimenzionalitou je myšleno velký počet atributů (pokud jsou jako rysy zvolena slova, tak je to obecně počet slov v přirozeném jazyce). Oproti databázím, kde je maximální počet sloupců obvykle několik desítek, je to obrovský rozdíl.Řídkost rysů souvisí s vysokou dimenzionalitou. Tím, že existuje velké množství dimenzí, je v běžném dokumentu využívaná pouze velmi malá část (z celkového počtu slov v přirozeném jazyce se v jednom dokumentu vyskytuje pouze zlomek). Nejčastěji se používají následující rysy dokumentů: znaky (jde o úplnou reprezentaci), slova (je jich mnoho, proto se optimalizují), termy (jsou to slova nebo slovní spojení, nutnost vytvoření slovníku), koncepty (slova, která se nemusejí přímo vyskytovat v dokumentu, například téma dokumentu) [5]. Existují dva přístupy k dolování v textu. Tradiční přístup, kde má uživatel dopředu představu o tom, co chce získat a předloží systému množinu dokumentů jako vzorek a nový přístup, kde je větší aktivita na straně systému. Systém uživateli předkládá návrhy, sám provádí filtrování, uspořádávání, shlukování a uživatel pouze takový systém navádí [26].

Jedním z prvních přístupů dolování z textu bylo pouhé zaznamenávání počtu výskytů jednotlivých slov. Tento postup bylo možné využít pro klasifikaci dokumentů, sumarizaci

dokumentů nebo jejich shlukování. Při použití této metody se ovšem úplně vytratil původní kontext i pořadí slov [5]. Příkladem takovéhoto přístupu je vektorová reprezentace článku. Reprezentace TF-IDF (term frequency - inverse document frequency) zohledňuje výskyty jednotlivých slova v textu a současně snižuje jejich důležitost podle množství výskytu v ostatních dokumentech [26]. Pokročilejší metodu představuje automatické konstruování textových klasifikátorů, kde se využívá statistických metod a učení na trénovací množině článků. Trénovací množinou jsou již ohodnocené dokumety, které byly ohodnoceny expertem. Naučený klasifikátor je schopný ohodnocovat nové, zatím neohodnocené dokumenty.

S dolováním z textu je spojena oblast zpracování přirozeného jazyka. Jedním z cílů tohoto oboru je, aby stroj porozuměl přirozenému jazyku. Toto je velmi obtížný a zatím nevyřešený úkol. Přirozený jazyk obsahuje mnoho nejednoznačností a často je potřeba si kontext domýšlet. Pro porozumění je tedy potřeba inteligence, která se blíží inteligenci člověka. V současné době se pro udržení kontextu používají různé ontologie (doménové znalosti). Ontologie je soubor pojmů, jejich významů a vztahů mezi nimi pro určitou oblast.

2.3 Extrakce informací z WWW

Prostředí webu je v dnešní době nejrozsáhlejším zdrojem dokumentů v elektronické podobě. Informace jsou rozmístěny na mnoha různých místech, což vede k nutnosti je efektivně vyhledávat. Existuje celá řada kvalitních vyhledávačů, ale i tak je uživatel nucen nalezené stránky procházet manuálně. Nalezených stránek je většinou velké množství, takže manuální procházení a hledání požadovaných informací je velmi náročné na čas. Málokdy se stane, že všechny zajímavé informace jsou na jednom místě, takže uživatel musí projít mnoho stránek, než všechny potřebné informace shromáždí. Dalším problémem je nalezení informace na samotné stránce. Některé stránky jsou velice nepřehledné, obsahují spoustu reklam, což obojí vede k prodloužení doby potřebné k nalezení požadované informace. Všechny tyto nevýhody jsou motivací pro obor získávání informací z WWW.

Dokumenty na webu jsou mnoha různých typů, nejčastěji však v podobě HTML. Důvodem velkého rozšíření HTML je jeho jednoduchost. Každý uživatel si bez větší námahy může vytvořit a umístit svou prezentaci v podobě HTML dokumentu na webu. HTML je ale jazyk bez pevně dané struktury a datům v takovém dokumentu nepřiřazuje sémantiku. Sémantiku si k datům přiřazuje až člověk. Tyto vlastnosti jsou nevhodné pro automatické zpracování strojem. Snahou o přiřazení sémantiky je Sémantický web.

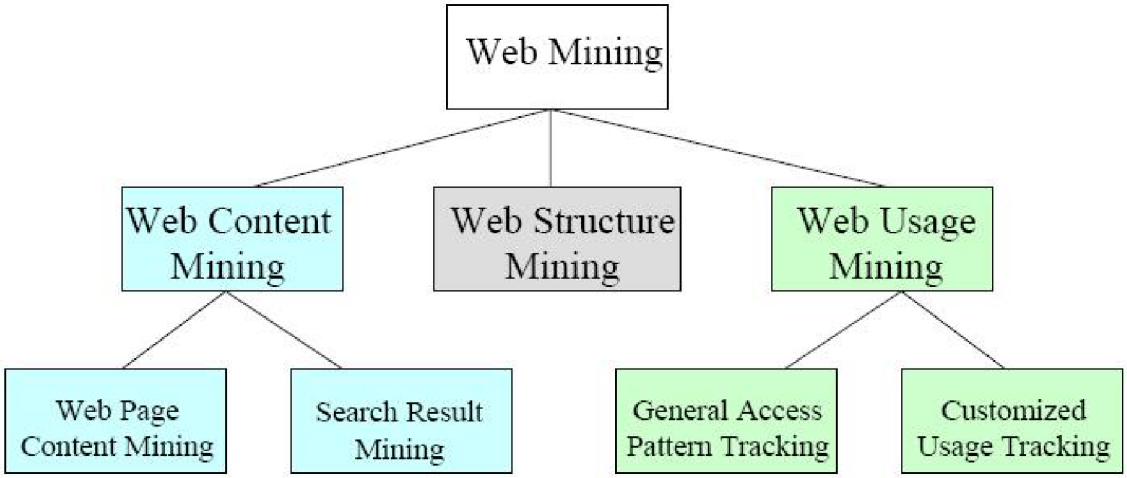
Existují dva přístupy pro získávání informací z webu. V prvním případě uživatel zadá co přesně chce najít a v druhém případě jsou nalezeny všechny dostupné informace na zadané téma. Získávání informací probíhá ve třech krocích [3]. V prvním kroku je nutné nalézt relevantní dokumenty obsahující požadované informace. V tomto kroku je možné využít nejrůznější vyhledávače jako je například Google. V druhém kroku je nutné nalézt a získat požadované informace přímo na nalezených dokumentech. V tomto kroku se využívá některá z metod pro získávání informací. V posledním kroku jsou získané informace uloženy

ve vhodné podobě jako je například dokument XML.

Při získávání informací z webu se využívají přístupy používané pro dolování u klasických databází a při dolování z textu. Oproti klasickým databázím se web liší především v tom, že informace nejsou dostupné ve strukturované podobě, struktura webové stránky je mnohem složitější než struktura databázové tabulky. Zajímá nás i vizuální informace a v neposlední řadě je obrovský rozdíl v rozsahu a dynamičnosti (stále přibývá velké množství nových stránek a na mnoha stránkách probíhají velmi časté změny).

Typy dat na webu, ze kterých může být zajímavé dolovat, jsou uvedeny na obrázku 2.1. Při dolování obsahu webové stránky (Web Page Content Mining) se jedná například o schopnost identifikace obsahu stránky na základě nějakého dotazovacího jazyka nebo o vyhledávání cen produktů na různých místech. Pro vyhledávání informací z obsahu stránky se dají použít wrappery. Nejjednodušším wraperem může být například skript, který pomocí regulárních výrazů vyhledává požadované informace ve zdrojovém kódu stránky a ukládá je například do XML. Existuje mnoho různých typů wrapperů viz. 4.1. Při dolování ve výsledcích vyhledávání (Search Result Mining) jde o shlukování výsledků vyhledávání na základě popisků od vyhledávače. Dolování struktury webu spočívá ve využití odkazů mezi stránkami (přidělování vah stránkám na základě grafu odkazů) a v zachyceni struktury webu na základě oblíbenosti jednotlivých stránek. Jako příklad může sloužit například PageRank od Google. Pro dolování přístupu ke stránkám (Web Usage Mining) je nutné mít k dispozici nějaký log soubor s informacemi o přístupu na stránku, jako je IP adresa, datum a čas nebo typ prohlížeče. V tomto případě můžeme využít stejných přístupů jako u klasických databází, protože log soubory mají strukturovanou podobu. Hledat můžeme zajímavé vzory a trendy (například nalezení stránek, které jsou často navštěvovány současně), což může přispět k vylepšení struktury odkazů na stránce. Pokud analyzujeme uživatelovo chování v čase, můžeme vytvářet adaptivní stránky (stránka se mění automaticky na základě přístupu uživatele). Problémem u těchto přístupů bývá to, že uživatel je často skrytý (proxy server, NAT, atd.).

Dalším stupněm je získávání vizuální informace. Aplikace takovéhoto typu nám mohou sloužit k okamžitému nalezení polohy požadovaných informací na strance, mohou nám zobrazit pouze relevantní části stránky bez reklam a podobně. Vizuální informace tvoří další dimenzi webové stránky. Stránky jsou většinou rozděleny do několika oblasti s různou důležitostí, čehož se dá využít v dalších krocích získávání informací. Proces získávání struktury stránky se nazývá segmentace. Výsledkem segmentace je rozdělení stránky do boxů, se kterými se může dál pracovat. Pro určení vzájemné polohy se používá reprezentace v mřížce. Segmentace stránky je netriviálním problémem, protože k dosažení stejného vzhledu stránky je možné postupovat mnoha způsoby (CSS, Javascript, atd.) a to se musí při segmentaci brát v úvahu. Existují dva postupy pro segmentaci, ze shora dolů a zdola nahoru. V prvním případě se začne se stránkou jako celkem a postupně se tento celek dělí na segmenty. V druhém případě se stránka rozdělí na nejmenší segmenty a ty se postupně spojují, dokud se nedosáhne požadovaného rozdělení. Obecně probíhá segmentace následujícím



Obrázek 2.1: Typy dat na webu (převzato z [5])

způsobem: v prvním kroku během rendrování stránky jsou získány boxy, v dalším kroku určíme, které boxy jsou pro nás zajímavé a oddělíme prázdné místa na stránce. Na získané oddělené oblasti můžeme pak použít existující algoritmy. K segmentaci je možné využít DOM (Document Object Model). Tímto postupem se ale nedosahuje dobrých výsledků, protože DOM je zobrazení založené více na obsahu a nemusí vždy odrážet sémantickou strukturu [5]. Dobrých výsledků je dosahováno algoritmem VIPS (VIsion-based Page Segmentation). Výsledkem tohoto algoritmu je stromová struktura, kde každý uzel odpovídá jednomu boxu. Tento algoritmus se snaží postihnout i sémantiku. Každému uzlu je přiřazena hodnota (stupeň koherence), která udává složitost boxu z hlediska vizuálního vnímání. Díky této hodnotě můžeme dosáhnout požadované granularity.

Kapitola 3

World Wide Web

Worl Wide Web je celosvětová síť vzájemně různě propojených dokumentu. Tento princip, kdy jsou dokumenty vzájemně propojeny pomocí odkazů se nazývá hypertext (pokud se jedná o dokumenty textové). Hypertext je vlastně text, který obsahuje odkaz na jiný text.

Autorem webu je Tim Berners-Lee, který navrhl HTTP (Hypertext Transfer Protocol), HTML (HyperText Markup Language), napsal první webový prohlížeč a koncem roku 1990 spustil první webový server [34]. V roce 1994 založil World Wide Web Consortium (W3C), které dohlíží na další vývoj webu. Hypertext Transfer Protocol, je komunikační protokol používaný pro přenos různých typů dokumentů. Původně to byly pouze dokumenty HTML. Dnes už jich je celá řada. Typy dokumentů určuje standard MIME (Multipurpose Internet Mail Extensions). HTML (HyperText Markup Language) je značkovací jazyk využívaný pro psaní webových stránek. Webové stránky, nebo obecně dokumenty, jsou umístěné na webovém serveru, což je počítač, který poskytuje tyto dokumenty na požádání klienta (webový prohlížeč). Server a klient spolu komunikují prostřednictvím HTTP. Každý dokument je jednoznačně identifikován svým URL (Uniform Resource Locator). Komunikace probíhá následujícím způsobem, klient odešle požadavek na nějaký dokument na server, ten mu buď požadovaný dokument odešle, nebo mu odpoví nějakou jinou zprávou (dokument nenalezen, nedostatečné oprávnění, atd.).

Web zaznamenal obrovský rozmach během devadesátých let a dnes představuje nejrozsáhlejší soubor dokumentů dostupných v elektronické podobě. Podle [**?**, wiki-www-en]ylo v roce 2001 na webu dostupných přibližně 550 miliard dokumentů. Takováto oblíbenost webu je dána především jeho jednoduchostí a dostupností. Vytvoření své vlastní prezentace na webu není pro pokročilejšího uživatele žádný problém.

Existuje mnoho různých technologií využívaných pro tvorbu webových aplikací, od velmi jednoduchých (HTML), až po robustní technologie typu Java EE. Při tvorbě jednoduchých prezentací postačí použití samotného HTML nebo v kombinaci s CSS a javascriptem. Při využití samotného HTML se jedná o statickou prezentaci, která neumožňuje interakci s uživatelem. Interakce s uživatelem lze dosáhnout například použitím javascriptu. Pro náročnější aplikace typu elektronický obchod, je nutné využít dalších technologií, jako

je PHP, ASP (Active Server Pages) od Microsoftu nebo JSP (Java Server Pages) od Sun Microsystems.

V následující části jsou popsány základní technologie využívané ve WWW.

3.1 Hypertext Transfer Protocol

HTTP (Hypertext Transfer Protocol) je komunikační protokol používaný pro přenos různých typů dokumentů v prostředí webu. Spolu s elektronickou poštou je to nejvíce používaný protokol [31].

Je to bezstavový protokol pracující na principu dotaz-odpověď. Klient prostřednictvím například webového prohlížeče pošle na server dotaz, který obsahuje informace o požadovaném dokumentu, informace o prohlížeči apod. Server odpovídá zasláním informací o výsledku dotazu (zda se požadovaný dokument podařilo nalézt, zda má uživatel dostatečné oprávnění pro přístup k dokumentu atd., dále obsahuje informace o samotném dokumentu) a pokud je vše v pořádku, následuje odeslání požadovaného dokumentu. Protože je protokol bezstavový, není možné udržovat kontext (není představa o dříve navštívených odkazech). Tato vlastnost je problémem ve složitějších aplikacích, kde je potřeba kontext udržovat (internetový obchod). Tento problém se řeší využitím dalších technologií.

Komunikace probíhá ve formě otevřeného textu, takže každý kdo komunikaci zachytí, může tento text bez problému číst. Existuje i bezpečnější verze HTTPS, kde probíhá komunikace šifrovaně.

3.2 Hypertext Markup Language

HTML (HyperText Markup Language) je značkovací jazyk určený pro psaní hypertextových dokumentů, které je pak možné prezentovat v prostředí webu. Vychází z dříve definovaného SGML (Standard Generalized Markup Language). Každá verze jazyka obsahuje určitou množinu povolených značek a ke každé značce množinu povolených atributů. Značky mohou být jednoduché a párové. Párové značky obklopují text a určují, jak se s ním bude pracovat (nadpis, odstavec, formátování textu atd.). Mezi jednoduché značky patří například značka `<br>`, která způsobí zalomení řádku (názvy značek jsou obklopeny úhlovými závorkami). Otevírací značka a uzavírací značka spolu s textem, který obklopují se nazývá element. Elementy se mohou vnořovat, proto se jedná z pohledu formálních jazyků o jazyk bezkontextový. Tím, že HTML umožňuje použití nepárových značek, nesplňuje podmínky dobře formovaného (well-formed) dokumentu.

HTML obsahuje značky umožňující definovat strukturu a vzhled dokumentu (odstavce, nadpisy, barvy pozadí, velikosti bloků atd.), značky pro práci s textem (velikost a barva písma, typ písma, font atd.), značky pro tvorbu tabulek, seznamů, značky pro připojení obrázků a mnoho dalších speciálních značek. Zásadní značkou je značka pro definování hypertextového odkazu. Pro definování vzhledu dokumentu, se upřednostňuje používání

kaskádových stylů (CSS). Ty mají oproti definici vzhledu přímo v HTML mnoho výhod viz. 3.4.

Standardy HTML:

- **HTML 2.0 -** říjen 1995, první verze odpovídající syntaxi SGML.
- **HTML 3.2 -** leden 1997, pro svou složitost nebyla tato specifikace nikdy přijata.
- **HTML 4.0 -** prosinec 1997, přidány nové prvky pro tvorbu tabulek a formulářů. Nově standardizovány rámy. Některé elementy byly zavrženy.

  Ve třech verzích:

  - Strict - Nepovoluje zavržené elementy.
  - Transitional - Umožňuje použít zavržené elementy.
  - Frameset - Pro použití rámů.
- **HTML 4.01 -** prosinec 1999, opravuje chyby předchozí verze a přidává některé nové značky. Stejně jako HTML 4.0 ve třech verzích. Strict, transitional a frameset. Po této verzi se mělo přejít na XHTML a vývoj HTML měl být ukončen.
- **HTML 5.0 -** 7. března 2007 byla založena nová pracovní skupina HTML, jejíž cílem je vývoj nové verze HTML. Tato verze už nebude vycházet z SGML, ale jejím základem se stane Web Applications 1.0 a Web Forms 2.0 [33].

3.3 Extensible Hypertext Markup Language

XHTML je nástupcem jazyka HTML, jehož vývoj měl být ukončen verzí 4.01. Narozdíl od svého předchůdce se jedná o dokument XML. Jedním z důvodů vzniku XHTML je možnost snadného parsování takového dokumentu využitím obecných XML prostředků. HTML tuto možnost nenabízí, protože se narozdíl od XHTML nejedná o dobře formovaný (well-formed) dokument.

Cílem první specifikace XHTML 1.0 bylo upravení jazyka HTML tak, aby vyhovoval podmínkám tvorby XML dokumentů a přitom byla zachována zpětná kompatibilita. Jsou definovány tři verze: Strict (stejné jako HTML 4.01 Strict, ale řídí se syntaktickými pravidly XML), Transitional (stejné jako HTML 4.01 Transitional, ale řídí se syntaktickými pravidly XML, podporuje vše co XHTML 1.0 Strict a navíc několik zastaralých elementů) a Frameset (stejné jako HTML 4.01 Frameset, ale řídí se syntaktickými pravidly XML).

Hlavní rozdíly oproti HTML jsou:

- Veškeré elementy musí být uzavřeny. To znamená, že musí být ukončeny i dříve jednoduché značky jako například `<br>`.

- Názvy všech značek a atributů jsou case sensitive a musejí být zapsány malými písmeny.
- Hodnoty atributů musejí být uzavřeny do uvozovek.
- Dokument musí začínat XML deklarací. Její použití není povinné, pokud je dokument kódován v UTF-8 nebo pokud určujeme kódování vyšší protokolem (například HTTP) [30].

Specifikace XHTML 1.1 vychází z XHTML 1.0 Strict. Byla zavedena modularizace, což vede k mnohem větší flexibilitě (použití pro webové prohlížeče, mobilní zařízení atd.). Jsou odstraněny všechny zavržené elementy a definice vzhledu je možná pouze prostřednictvím CSS. Tento typ dokumentu je nutno odesílat jako MIME type application/xhtml+xml.

XHTML 2.0 je stále ve vývojovém stádiu, tato specifikace už nebude zpětně kompatibilní.

3.4 Cascading Style Sheets

CSS (Cascading Style Sheets) je jazyk pro popis způsobu zobrazení dokumentů napsaných některým ze značkovacích jazyků. Nejčastěji je využíván pro stylování webových stránek napsaných v HTML nebo XHTML. Jazyk byl navržen a je udržován standardizační organizací W3C (World Wide Web Consortium).

Hlavním cílem CSS je oddělit popis vzhledu dokumentu od popisu jeho struktury a obsahu. Pod popisem vzhledu si můžeme představit například výběr barvy písma, velikosti písma, typ fontů, ale i layout stránky. Starší verze jazyka HTML obsahují velké množství elementů, které nepopisují pouze strukturu a obsah dokumentu, ale i vzhled dokumentu. V XHTML 1.1 jsou všechny tyto vlastnosti pro popis vzhledu odstraněny a jediný možný způsob pro popis vzhledu je prostřednictvím CSS.

Hlavní výhody CSS:

- **Rozsáhlejší možnosti pro popis vzhledu -** CSS nabízí rozsáhlejší možnosti popisu vzhledu dokumentu než samotné HTML.
- **Konzistentní vzhled -** K dodržení konzistentnosti vzhledu (například nadpisů, seznamů atd.) bylo v HTML nutné vzhled objektu definovat při každém jeho výskytu, což bylo pracné a velmi špatně udržovatelné. Při jakékoliv změně vzhledu bylo nutné najít všechny výskyty objektu a jeho vzhled upravit. Za použití CSS se vše výrazně zjednodušilo. Veškeré definice vzhledu se uloží do zvláštního souboru, který se připojuje k HTML. Předpisy v tomto souboru pak platí pro celý HTML soubor, takže například pro změnu barvy všech nadpisů stačí editovat pouze definici na jednom místě.
- **Oddělení popisu vzhledu od popisu struktury a obsahu**

- **Snadná změna vzhledu -** Pro změnu vzhledu webových stránek stačí editovat jeden soubor. Není nutné vyhledávat jednotlivé elementy v HTML dokumentu.
- **Kratší doba načítání stránky**

Kaskádové styly je možné importovat do html dokumentu přímím vložením do HTML hlavičky, přímím vložením do jednotlivých tagů pomocí atributu *style* nebo připojením externího souboru s kaskádovými styly pomocí příkazu v HTML hlavičce.

Hlavní nevýhodou CSS je stále špatná podpora webových prohlížečů, které se nedrží specifikace CSS. To způsobuje, že různé prohlížeče interpretují CSS různě, což znamená někdy i značné odlišnosti v konečném vzhledu webové stránky. Pro tvůrce webové stránky je obtížné dosáhnout stejného vzhledu v různých prohlížečích. Pro majoritní prohlížeče toho lze dosáhnout použitím různých postupů, které ovšem mohou způsobit, že se dokument s kaskádovými styly stane nevalidním. Jedním z používaných postupů, je vytvoření více různých stylů a přiřazení odpovídajícího stylu v závislosti na použitém prohlížeči. V dnešní době už majoritní prohlížeče v dodržování standardu velmi pokročili. Přesto dosáhnutí stejného vzhledu u netriviálních aplikací ve všech existujících prohlížečích je téměř nemožné.

3.5 Sémantický web

Sémantický web se má stát dalším stupněm dnešního webu. Problémem dnešního webu je to, že postrádá sémantiku a pevně danou strukturu, takže automatické zpracování počítačem je velmi obtížné. Sémantický web se snaží tyto nedostatky odstranit. Informace jsou strukturovány a ukládány podle standardizovaných pravidel, což usnadňuje jejich vyhledání a zpracování [32].Uživatel by tak měl možnost pracovat s webem jako s databází a využívat tak dotazovacích jazyků podobných SQL.

Sémantický web je založen na následujících technologiích: XML, RDF (Resource Description Framework) a OWL (Web Ontology Language). RDF je založen na myšlence přidělovat k jednotlivým zdrojům takzvané trojice (podmět-vlastnost-předmět). Výrok "Obloha má modrou barvu" je v RDF reprezentována trojicí řetězců, kde podmět je Obloha, vlastnost je má barvu a předmět je modrá. Předmětem v prostředí webu je URI (Uniform Resource Identifier). Tímto přístupem přiřadíme každému zdroji požadovanou sémantiku. OWL je jazyk pro popis ontologií. Ontologie je soubor s formálním popisem nějaké problematiky. Může obsahovat popis pojmů, různých vztahů mezi nimi apod. Můžeme chápat jako slovník vysvětlující nějaké výrazy.

3.6 Dynamický obsah

Doba kdy web sloužil především pro prezentace a kdy HTML byl postačujícím nástrojem je dávno pryč. Dnes jsou na webu k dispozici propracované dynamické aplikace. K vytvoření takových aplikací samotný HTML nestačí, proto vznikly další jazyky, díky kterým webové

stránky přestávají být pouhou statickou prezentací, ale umožňují interakci s uživatelem. Příkladem náročnější webové aplikace, která by nešla vyvinout pouze s použitím HTML je elektronický obchod. Prvním problémem, který se musí řešit použitím pokročilejších technologií je udržování kontextu při pohybu uživatele na stránce (například zboží v nákupním košíku). Dále je nutná interakce s uživatelem (přihlášení uživatele, zadávání různých hodnot uživatelem apod.). Další výhodou těchto technologií je dynamické generování obsahu, který se poté odešle ke klientovi.

Tyto jazyky lze rozdělit do dvou skupin. Jedna skupina jazyků se vykonává na straně serveru a klientovy je odeslána už finální podoba dokumentu. Tohoto se dá velmi dobře využít v kombinaci s nějakou databázovou technologií, kdy jsou uživateli odesílány dokumenty generované na základě dat uložených v databázi. Mezi tyto jazyky patří například PHP, ASP od Microsoftu nebo JSP od Sun Microsystems. Druhá skupina je zpracovávána webovým prohlížečem u přímo u klienta (Javascript). Ať už se jedná o jazyky prováděné na straně serveru nebo u klienta, jejich funkcí je především dynamicky měnit obsah stránky na základě interakce s uživatelem.

Z pohledu získávání informací tento přístup představuje problém pouze u jazyků prováděných na straně klienta [3]. U takovýchto jazyku se webové stránky mění až lokálně u klienta a proto tyto změny nemohou být dostupné. U jazyků prováděných na straně serveru problém nevzniká, protože dokument dorazí ke klientovi ve finální podobě. Problémem u dokumentů generovaných na straně serveru je to, že vzniká tzv. skrytý web [3]. Skrytý web je označení používané pro stránky, které nejsou dostupné pro automatické zpracování. Nedostupnost je způsobena tím, že některé stránky jsou odeslány klientovi až po zadání požadovaných dat prostřednictvím formuláře. Dalším příkladem jsou stránky v nějaké zabezpečené části, kam je umožněn přístup pouze přihlášeným uživatelům. Skrytý web je nedostupný nejen pro automatické aplikace dolující informace, ale například i pro indexovací roboty různých vyhledávačů.

Kapitola 4

Používané přístupy pro extrakci informace z WWW

Ačkoliv existují alternativní technologie pro prezentaci dat v prostředí WWW jako je sémantický web nebo web services, které jsou vhodné pro automatické zpracování a získávání informací, tyto technologie se stále nedočkaly většího rozšíření. Je to dáno především větší složitostí oproti klasickému přístupu. Jak již bylo řečeno, jednoduchou prezentaci s využitím HTML dokáže vytvořit skutečně téměř každý, to se o sémantickém webu nebo web services říci nedá. Dalším důvodem, proč se tyto alternativní technologie příliš neprosazují, muže být neochota vývojářů přejít z dobře známých technologií na ty nové. Tyto a mnoho dalších důvodů způsobuje, že počet dokumentů vytvořených klasickými technologiemi se stále prudce zvyšuje, zatímco nárůst dokumentů například v sémantickém webu tak razantní není. Z tohoto důvodu je nutné používat přístupy, které dokáží získávat informace z klasických HTML dokumentů. Základní postupy, které se pro toto využívají jsou popsány v následující části.

Extrakci informací z WWW můžeme rozdělit do dvou oblastí. V prvním případě jde o procházení HTML dokumentů wrappery, které získávají požadované informace a v druhém případě jde o modelování logické struktury dokumentů a zjišťování sémantiky [3].

4.1 Wrapper

Wrapper je program, který na základě určitých pravidel získává relevantní informace z HTML dokumentu a ukládá je ve vhodné podobě (například XML) k dalšímu zpracování [2]. Wrapper pracuje následujícím způsobem: na vstup dostane zdrojový kód HTML dokumentu a na základě pravidel pro extrakci z tohoto dokumentu získá požadované informace a uloží je jako XML. Tyto pravidla mohou říkat například, že text mezi značkami `<b>` a `</b>` je název země a text mezi `<i>` a `</i>` je název hlavního města. Takovýto přístup je samozřejmě naprosto nedostačující. Při změně stránky (například název země bude umístěn mezi `<h3>` a `<h3/>`) přestane wrapper správně pracovat. Změna stránky je obecný problém

u wrapperů. S tím je spojena další nevýhoda. Wrapper pracuje spolehlivě pouze u dokumentů, které mají podobnou strukturu (například nějaký soubor dokumentů generovaných z databáze) což zabraňuje obecnému použití. Pokud chceme například sdružovat informace z více různorodých stránek, musíme pro každou z nich vytvořit speciální wrapper. Pokud se některá ze stránek změní, je nutné tuto změnu promítnout také do wrapperu.

4.1.1 Rozdělení wrapperů

Wrappery můžeme obecně rozdělit do dvou skupin. První skupina, jako například W4F [24] nebo Lixto [2], využívá struktury webové stránky. Druhá skupina (například Cameleon [9]) zpracovává webovou stránku jako prostý text [25]. Další dělení je možné provést podlé míry automatizace vytváření wrapperu. Nejzákladnějším přístupem je ruční programování wrapperu pro každou stránku zvlášť. Při vytváření takového wrapperu je možné použít například regulární výrazy, pomocí kterých definujeme, které části zdrojového kódu webové stránky chceme extrahovat. Takovýto přístup je samozřejmě časově velice náročný a je zdrojem mnoha chyb. Na druhou stranu, programátor má úplnou kontrolu, takže si wrapper může vytvořit přesně podle svých představ. Existují ale i vysoce automatické metody vytváření wrapperů, kde jsou použity metody strojového učení. Mezi ně patří např. SoftMealy [15] nebo algoritmus Stalker [23]. U těchto metod odpadá časově náročné ruční programování, ale na druhou stranu se může zmenšit přesnost extrakce [25]. Dále je možné wrappery dělit způsobem, který navrhl N. Kushmerick v [20]. Tento způsob dělí wrappery do šesti skupin podle způsobu, jakým jsou definovány extrakční pravidla. Těchto šest typů wrapperů, by mělo vyhovovat pro získávání informací ze 70% současného webu. Pro ukázku (obrázek 4.1) tvaru extrakčních pravidel použijeme úsek zdrojového kódu jednoduché webové stránky, která obsahuje informace o počtu obyvatel a hlavních městech v různých zemích světa.

Jednotlivé řádky této tabulky obsahují informace o počtu obyvatel a hlavních městech různých států světa. Řádky označíme jako záznamy. Potom každý záznam je tvořen několika prvky. Jeden záznam je tedy tvořen prvky obsahující název státu, počet obyvatel a název hlavního města.

Dělení wrapperů podle typu extrakčních pravidel:

- **LR (left-right) wraper -** nejjednodušší typ, jednotlivá extrakční pravidla definují jednotlivé prvky v záznamu. Wrapper je potom popsán souborem takovýchto pravidel. Extrakční pravidlo může být popsáno následujícím způsobem: text mezi značkou *<td class="modre">* a *</td>* odpovídá názvu státu. Stejným způsobem jsou definovány pravidla pro ostatní prvky v záznamu. Takto definovaná pravidla by ovšem mohla zahrnout chybně i některé další hodnoty z ostatních tabulek se stejnou strukturou v dokumentu. Tento problém odstraňuje další typ wrapperů.

- **HLRT (head-left-right-tail) wrapper -** u tohoto typu wrapperu jsou přidány další dva parametry pro extrakční pravidla. Parametr head slouží k přeskočení úvodního, nezajímavého textu. Parametr tail slouží k označení konce pro nás zajímavé části.

```
<table><caption>Hlavní města a počet obyvatel</caption>
      <tr>
        <th>Stát</th>
        <th>Počet obyvatel</th>
        <th>Hlavní město</th>
      </tr>
      <tr>
        <td class="modre">Čína</td>
        <td class="cervene">1 313 973 713</td>
        <td class="zelene">Peking</td>
      </tr>
      <tr>
        <td class="modre">Indie</td>
        <td class="cervene">1 095 351 995</td>
        <td class="zelene">Dillí</td>
      </tr>
      .
      .
      .
</table>
```

Obrázek 4.1: Zdrojový kód tabulky zobrazující počty obyvatel a hlavní města u různých států světa

Ve výsledku můžeme těmito parametry říci, kterou část dokumentu prohledávat. V našem případě by parametr *head* odpovídal řetězci *<caption>Hlavní města a počet obyvatel</caption>* a *tail* značce *</table>*. Tento typ odstraňuje problém vznikající u LR wrapperů. Tím, že určíme začátek a konec prohledávání, nemůže se stát, že se chybně vyberou prvky i z jiných tabulek v dokumentu, které mají podobnou strukturu.

- **OCLR (open-close-left-right) -** parametry *open* a *close* definují začátek a konec záznamu. V našem případě by *open* odpovídal značce *<tr>* a *close* značce *</tr>*.
- **HOCLRT (head-open-close-left-right-tail) -** tento typ kombinuje předcházející.
- **N-LR a N-HLRT -** tento typ je určen pro získání dat z vnořených tabulek.

4.1.2 Automatická konstrukce wrapperů

Jak již bylo řečeno v předešlé části, ruční programování wrapperů je časově velice náročné a náchylné na chyby. Proto byly vytvořeny metody, které umožňují automatickou tvorbu

wrapperů. Automatická tvorba wrapperů je založena na metodách strojového učení a probíhá obecně v následujících krocích: nejprve je nutné získat trénovací množinu dokumentů, v dalším kroku na této trénovací množině probíhá učení. Výsledkem tohoto učení jsou pravidla pro extrakci. V dalším kroku je na základě těchto pravidel vybudován wrapper, který se už může použít pro extrakci informací z dokumentů, pro které byl navržen [3].

Model dokumentu

Existuje mnoho různých algoritmů strojového učení, které vyžadují vstupní dokumenty v různých podobách. Pro některé algoritmy je výhodné pracovat s dokumentem jako s čistým textem, pro jiné je vhodnější například stromová struktura. Tyto jednotlivé reprezentace dokumentu jsou označovány jako tzv. model dokumentu [3].

Jak již bylo řečeno, jedním ze způsobů reprezentace je souvislý proud znaků. To znamená, že celý dokument (text i značky) je zpracováván znak po znaku. V takovémto případě jsou pravidla pro extrakci založena například na regulárních výrazech. Pomocí regulárních výrazů můžeme hledat různé zajímavé tvary a kombinace slov (slova končící dvojtečkou, končící vykřičníkem apod.) čímž vlastně definujeme extrakční pravidla.

Další možnou reprezentací dokumentu jsou jednotlivá slova. Pro strojové učení je tento způsob vhodnější než reprezentace pomocí jednotlivých znaků. Ke každému slovu může být připojen výčet atributů, které určují jeho vlastnosti. Značky obklopující toto slovo jsou buď ignorovány, nebo jsou z nich vydedukovány další vlastnosti.

Nejčastěji používaným modelem je však hierarchický model. Takovýto model zachycuje logickou strukturu dokumentu. Mezi hierarchické modely patří například strom, kde jsou jednotlivé uzly tvořeny elementy dokumentu. Pokud se jedná o HTML dokument, kořenovým uzlem je element html, ten dále obsahuje uzly head a body atd. Čistý text obsažený v dokumentu odpovídá listovým uzlům nebo je z modelu uplně vypuštěn. Pokud chceme vytvořit takovýto model, musí být dokument dobře formovaný (well formed). To znamená, že ke každé otevírací značce musí existovat správně umístěná odpovídající uzavírající značka. Pokud se jedná o XHTML, je tato podmínka splněna, pokud o HTML, je nutné tuto podmínku ověřit, a případně dokument upravit. Výhodou takovéhoto modelu je zachycení vzájemných vztahů mezi značkami. Příkladem takovéto reprezentace je DOM (document object model).

Metody automatického učení

V procesu učení pravidel pro extrakci se využívá několika základních přístupů. První z nich je založen na sestavování gramatiky nebo konečného automatu. Problém je definován takto, máme konečnou abecedu Σ, jazyk L nad touto abecedou, množinu vět nad touto abecedou, která patří do jazyka L a množinu vět nad touto abecedou, která nepatří do tohoto jazyka. Cílem je najít gramatiku generující jazyk L. Z pohledu extrakce informací problém nalezení pravidel pro extrakci pro určitou množinu dokumentů odpovídá problému nale-

zení gramatiky, která generuje kód dokumentů. Tento přístup není možné aplikovat na HTML dokumenty přímo, protože máme k dispozici pouze ty dokumenty, které korespondují s větami nad abecedou Σ, které leží v jazyku L, ale nemáme dokumenty korespondující s větami nad abecedou Σ, které neleží v jazyku L a podle [13] není možné vygenerovat gramatiku pouze ze znalosti vět ležících v jazyku L. Řešením tohoto problému může být vytvoření dokumentů, které odpovídají větám nad abecedou a neleží v jazyku L.

Jeden z přístupů využívající tohoto principu je popsán v [11]. Spolu se sestavováním gramatiky, využívá tento přístup i Bayesovské klasifikace. Dokument je reprezentován sekvencí slov. Bayesovský klasifikátor postupně po krocích zpracovává dokument. V každém kroku zpracuje stejný, pevně daný počet slov a odhaduje s jakou pravděpodobnosti jde o data, která nás zajímají. Problémem je odhadnout, ideální počet slov, která se mají zpracovat v jednom kroku a navíc takovýto přístup nebere v úvahu pořadí slov, proto se využívá i gramatika. Jednotlivá slova z dokumentu jsou nahrazena symboly z abecedy, nad kterou je tato gramatika poté generována. Tyto symboly vyjadřují určité vlastnosti slov (například velikost, styl atd.). Všem koncovým stavům v automatu sestaveném na základě vygenerované gramatiky jsou přiřazeny pravděpodobnosti, že řetězec, se kterým jsme se dostali do koncového stavu odpovídá datům, které chceme získat. Pokud řetězec konečným automatem není vůbec přijat, přiřadí se mu velice nízká pravděpodobnost. Výsledkem je potom kombinace pravděpodobností získaných z Bayesova klasifikátoru a z konečného automatu.

Další přístup na principu sestavování gramatiky, který využívá stromovou reprezentaci dokumentu je popsán v [19]. U tohoto přístupu není cílem vygenerovat automat, který přijímá řetězce odpovídající pořadovaným datům (respektive určuje s jakou pravděpodobností je přijmaný řetězec řetězcem, který nás zajíma), ale vygenerovat automat, který přijímá odpovídající stromové modely dokumentů. V prvním kroku jsou vytvořeny stromové modely trénovacích dokumentů, kde jsou označeny data určená k extrakci speciálním symbolem. Poté je vygenerován automat přijímající tyto stromové modely. Tento vygenerovaný automat je poté použit pro extrakci informací. U dokumentů, ze kterých chceme získat informace se postupně nahrazují jednotlivé uzly ve stromovém modelu speciálním symbolem a pokud je model automatem přijat, původní hodnoty, které byly nahrazeny speciálním symbolem jsou hledaná data.

Rozdílný přístup je popsán v [14]. Je založen na stochastických bezkontextových gramatikách, kde je z mnoha vygenerovaných gramatik vybrána ta nejjednodušší. Nonterminály odpovídají základním částem dokumentu. Pro přesnou lokalizaci jednotlivých dat, které chceme získat jsou využity regulární výrazy.

Další přístup [8] je založen na hledání společných schémat dokumentů. Schéma je definováno společnými, neměnnými částmi a společnými částmi, které se dokument od dokumentu liší. Takové části považujeme za zajímavé data.

Další ze základních principů pracuje s skrytými Markovovými modely (Hidden Markov Model).

Základní myšlenka spočívá v tom, že HTML dokument vzniká nějakým stochastickým postupem a cílem této metody je nalezení skrytého Markovova modelu, který by tento postup reprezentoval. Skrytý Markovův model si můžeme představit jako automat, kde mezi stavy přecházíme s určitými pravděpodobnostmi a z každého stavu sou generovány s určitými pravděpodobnosti nějaké výstupy. Při extrahování informací, stavy odpovídají tokenům, které chceme extrahovat. Jednotlivé pravděpodobnosti jsou získány v procesu učení nad množinou trénovacích dat. Cílem je nalezení posloupnosti stavů, která s největší pravděpodobností vedla k vygenerování dokumentu a získání symbolů, které tyto jednotlivé stavy generují. Tohoto přístupu využívá například [12].

Další z principů využívá Relational Learning algoritmy. Opět se předpokládá množina dokumentů s podobnými vlastnostmi, pro které se bude wrapper sestavovat a trénovací množina dokumentů, která je vybrána z této množiny. Data, která nás zajímají a chceme je extrahovat, jsou nejprve popsány logickými predikáty. Poté probíhá učení, jehož výsledkem jsou obecná pravidla pro extrakci požadovaných dat.

Alternativní přístupy

Alternativní přístupy používané k sestavení wrapperů se snaží analyzovat přímo kód HTML dokumentu. Cílem těchto přístupů je odstranit fázi učení, která byla používaná v předešlých přístupech. Tím, že odpadá fáze učení, odpadá i získávání trénovacích dokumentů. Tyto metody využívají především heuristik a často je obtížné určit, pro které dokumenty se hodí. Předpokládá se znalost domény [3].

Heuristik využívá například Ashish [1], který hledá v dokumentu zajímavé slova (tzv. tokeny). Tokeny identifikuje na základě vlastnosti textu a podle značek, které text obklopují a regulárních výrazů, které určují výskyty [3]. Každý token označuje začátek sekce dokumentu. Na základě porovnání velikosti písma a odražení textu začínající každou sekci se určí hierarchie sekcí. K sestavení wrapperu se využívá YACC generátor. Další přístup použitý v [4] hledá společný separátor, který odděluje data v dokumentu.

Dalším z alternativních přístupů je konceptuální modelování. Tento přístup se využívá především pro dolování z čistého textu, ale je možné ho použít i pro HTML [3].

Uživatelem řízená tvorba wrapperu

Úplně odlišného přístupu využívá například Lixto [2]. Sestavování wrappru probíhá na základě spolupráce s uživatelem. Lixto nabízí grafické uživatelské rozhraní, díky kterému i uživatel neznalý HTML může vytvářet wrappery. Výstupem takto vytvořeného wrapperu je XML soubor. Lixto pracuje s deklarativním logickým jazykem Elog.

4.2 Modelování logické struktury dokumentů

Wrappery, popsané v předešlé části, jsou určeny především k získávání informací z HTML dokumentů, kde je formátování dokumentu řešeno přímo v HTLM kódu. Pokud se definice vzhledu stránky z dokumentu vyčlení, například použitím kaskádových stylů, použití wrapperů je problematické. Tím, jak se informace o vzhledu přenáší mimo kód HTML, wrappery přicházejí o zdroj informací, který lze využít pro extrakci zajímavých hodnot. Jako příklad je možné uvést nahrazení značky <*b*>. Při použití této značky, se dalo usuzovat, že se jedná o nějakou důležitou informaci. Pokud je tato značka nahrazená značkou <*span*> a definice vzhledu je umístěna v externím souboru, wrapper využívající přímo kód HTML nezjistí, že se jedná o nějaká významná data. V dnešní době je vyčleňování vzhledu stránky z HTML kódu dobrým programátorským zvykem, takže většina stránek kaskádové styly používá. Výhody použití kaskádových stylů jsou popsány v 3.4. Využití kaskádových stylů má tedy za následek, že užití postupů popsaných v předešlé části je problematické, nebo dokonce nemožné [3].

Dalším problémem u wrapperů popsaných v předešlé části je vysoká specializace. Každý wrapper se hodí pouze pro malou množinu dokumentů. Potřebné informace se však mnohokrát nenacházejí v jednom dokumentu, ale jsou rozmístěné v mnoha různých dokumentech na různých serverech. Při použití wrapperů by bylo nutné napsat speciální wrapper pro každý dokument (pokud by měl i jen lehce odlišnou strukturu) zvlášť. Z těchto důvodu byly vyvinuty postupy, které nevyužívají pouze samotného HTML kódu, ale snaží se najít další vlastnosti stránky, které by bylo možné využít pro získávání informací. Jedná se především o vizuální podobu stránky a logickou strukturu stránky. Zjišťování těchto vlastností stránky je popsáno v následující části.

4.2.1 Model logické struktury dokumentu založený na hierarchii značek

Při modelování logické struktury dokumentu jde o získání modelu, který by vystihoval hierarchickou strukturu dokumentu a vztahy mezi jeho částmi. Tyto modely mají nejčastěji podobu stromu [3]. Při použití tohoto postupu se předpokládá, že hierarchie HTML značek v dokumentu odpovídá logické struktuře dokumentu. Tento předpoklad ovšem často není splněn. Toto nastává velmi často při použití kaskádových stylů. Pro dosažení požadovaného vzhledu se může logická struktura dokumentu lišit od struktury značek ve fyzickém dokumentu.

4.2.2 Model logické struktury dokumentu založený na vizuální analýze

Vizuální analýza dokumentu spočívá v odhalování logických částí dokumentu na základě jejich polohy. Je to netriviální probmlém, protože finální podobu stránky je možné dosáhnout mnoha způsoby. Webová stránka je obvykle rozdělená na logické části jako je například hlavička, menu, patička a část, ve které jsou zobrazována data. Nalezení těchto částí je jedním z cílů vizuální analýzy.

Jeden z přístupů nejprve nalezne v dokumentu podobné vzory kódu a v dalším kroku, na základě sémantiky HTML značky (například značka *<form>* znamená, že se jedná o interaktivní část) určí, o kterou logickou část se jedná [3].

Další přístup využívá zjištění, že stránky s podobným tématem mají podobnou vizuální strukturu a lze odhadnout hranice mezi jednotlivými částmi. V prvním kroku je zjištěno, jak moc si jsou stránky podobné. V dalším kroku jsou detekovány časté vizuální vzory a nakonec je sestaven hierarchický model dokumentu [3].

Kapitola 5

Analýza

V následující kapitole je uveden popis systému a analýza systému.

5.1 Popis systému

Cílem této práce je rámec pro extrakci informací z WWW a jejich další zpracování. Dalším zpracováním se myslí ukládání získaných informací například do XML souboru nebo do relační databáze.

Jedná se o systém implementovaný v jazyce Java (platforma Java SE), který umožňuje snadnou práci s různými extrakčními metodami. Tyto extrakční metody představují moduly, které systém umožňuje dynamicky načítat. Dalším typem modulů, se kterými systém pracuje, jsou moduly pro uložení extahovaných dat. Pro každý typ modulu je definováno rozhraní, které musí moduly implementovat. Systém dále zajišťuje načtení vstupních dokumentů z lokálního disku nebo dokumentů dostupných na WWW prostřednictvím protokolu HTTP. Pro efektivní práci s moduly a vstupními dokumenty systém umožňuje dávkové zpracování zadané prostřednictvím XML souboru. Aplikaci tvoří dvě části. První z nich zajišťuje výše popsanou funkčnost. Tato část představuje knihovnu, kterou je možné znovu použít pro vytváření dalších aplikací. Druhou částí alikace je grafické uživatelské rozhraní. Cílem systému je usnadnit vývoj různých extrakčních metod a práci s nimi.

5.2 Analýza

Tato část obsahuje popis základních části systému a analýzu problémů, které bude nutné řešit.

5.2.1 Získání vstupních dokumentů

První částí systému bude získávání požadovaných dokumentů, ze kterých se budou informace extrahovat. Systém musí umožnit načítání dokumentů z lokálního disku a z WWW.

Prvním problémem je zjišťování použité diakritiky. Detekce diakritiky je důležitým krokem z důvodu správné funkčnosti některých používaných extrakčních metod. Například při využití regulárních výrazů pro hledání požadovaných dat může být použit i řetězec s diakritikou. Pokud by byl vstupní soubor načten za použití špatné znakové sady, tento regulární výraz by požadovaná data nenašel. Podle W3C (World Wide Web Consortium) [28] je doporučován následující postup při detekci kódování:

1. **Detekce z HTTP hlavičky -** Prvním ze způsobů detekce kódování je načtení pole *Content-Type*, které obsahuje parametr *charset* z HTTP hlavičky. Toto pole může vypadat následujícím způsobem: *Content-Type: text/html;charset=UTF-8*. Hodnoty parametru *charset* jsou case-insensitive.

2. **Načtení z elementu meta -** Dalším způsobem jak zjistit použité kódování, je parsováním elementu *head*, který obsahuje element *meta*. Tento element může vypadat následujícím způsobem: *<meta http-equiv='Content-Type' content='text/html; charset=windows-1250'/>*. Hodnoty parametru *charset* udávají použité kódování. Tyto hodnoty jsou case-insensitive. DTD pro HTML 4.01 Strict [27] i pro XHTML 1.0 Strict [29] definují, že jediným povinným atributem elementu *meta* je atribut *content*. Pokud se pro získání hodnoty atributu *content* použije vlastní parser s využitím regularních vyrazů, je nutné s tímto počítat. Dále je důležitá informace, že nezáleží na pořadí atributů.

3. **Použití parametru charset z elementu, který označuje externí zdroj -** V případě, že není možné získat kódování ani z HTTP hlavičky, ani z elementu *meta*, je možné využít atributu *charset* v některých dalších elementech. Například u elementu *a*, tento atribut udává použité kódování v odkazovaném zdroji.

Pokud ani jeden z uvedených postupů nevede k detekci kódování, je možné využít automatické detekce na základě porovnávání jednotlivých znaků. Takováto automatická detekce kódování je založena na využití heuristik a statistik. Jedná se tedy vždy o odhad a není možné se na něj stoprocentně spolehnout. Detektor může pracovat spolehlivě pro určitou množinu kódových sad, ale obecně není možné vytvořit automatický detektor pro všechna používaná kódování. Protože je detekce založena na statistikách, větší šance pro správnou detekci je u souborů obsahujících více znaků (aspoň několik stovek bajtů). Jeden z možných přístupů používaných pro automatickou detekci je popsán v [17]. U vícebajtových kódování se hledají sekvence bajtů, které by odpovídaly legálním výskytům v testovaném kódování a jednotlivé testované znaky jsou porovnávány s často užívanými znaky v textových řetězcích s testovaným kódováním. U jednobajtových kódování se data porovnávají s nejčastějšími tříznakovými sekvencemi, které se vyskytují v jednotlivých jazycích za použití různých kódování. Pokud ani tento postup neuspěje, je nutné použít defaultní kódování, čímž se velice snižuje možnost správného zpracování dokumentu. Uvedený postup lze použít na dokumenty načítané z WWW prostřednictvím protokolu HTTP. Pokud jsou dokumenty načítané

z lokálního disku, první krok nelze použít. Druhý a třetí krok má smysl použít pouze na HTML nebo XHTML dokumenty. U ostatních textových dokumentů, se pravděpodobnost detekce správného kódování výrazně snižuje.

Dalším problémem je detekce typu dokumentu. Pro získávání informací má smysl používat pouze dokumenty s textovým obsahem. Pro zjištění typu lze postupovat podobným způsobem jako při zjišťování kódování. Prvním ze způsobů je načtením pole *Content-Type* z HTTP hlavičky. Toto pole může obsahovat hodnoty specifikované organizací IANA (Internet Assignet Numbers Authority) v [16]. Například hodnota *text\html* určuje, že se jedná o html dokument, hodnota *text\xml* odpovídá xml dokumentu. Pro zjištění, že se jedná o typ s textovým obsahem stačí ověřit, že hodnota uvedená v poli *Content-Type* začíná řetězcem *text*. Dalším způsobem, stejně jako při určování kódování, je parsování elementu *meta*. V tomto případě jde o zjištění hodnoty parametru *content*. Pokud tato hodnota začíná řetězcem *text*, jedná se o typ dokumentu s textovým obsahem. Zjišťování typu z HTTP hlavičky je možné jen u dokumentů získávaných prostřednictvím tohoto protokolu. U dokumentů načítaných z lokálního disku je nutné použít druhy způsob nebo se dokumenty filtrovat na základě jejich koncovky.

Další problém je spojený s načítáním dokumentů z WWW. Pomale spojení nebo nevýkonný server, můžou významným způsobem ovlivnit dobu načítání dokumentu. Z tohoto důvodu je nutné definovat maximální dobu, po kterou se bude na načtení dokumentu čekat. Dále je nutné definovat, které protokoly podporovat a jak se bude postupovat pokud zadaný dokument není dostupný.

5.2.2 Dynamické načítání modulů

O načítání tříd se v jazyku java stará class loader, který je součástí JVM (Java Virtual Machine). Některou z instancí třídy *java.lang.ClassLoader* jsou načteny všechny používané třídy v programu. Jeho úkolem je najít požadovanou třídu a načíst ji do paměti.

Při startu programu se samozaváděcí class loader postará o načtení klíčových tříd jako je *java.lang.Object* a ostatního nezbytného kódu do paměti. Runtime knihovny jsou umístěny v archivu jre/lib/rt.jar. Způsob jakým tento class loader pracuje není dostupný ve specifikaci jazyka, protože se jedná o nativní implementaci. Jakmile jsou načteny tyto základní třídy, ostatní třídy jsou načítány v okamžiku, pokud jsou vyžadovány některou z už běžících tříd. O načítání těchto tříd už se stará některá z instancí třídy *java.lang.ClassLoader* [21].

Pokud chování systémového class loaderu nevyhovuje požadavkům, je možné vytvořit si svůj vlastní odvozením z třídy *java.lang.ClassLoader*. Svůj vlastní class loader lze využít například pro načítání tříd dostupných prostřednictvím HTTP protokolu nebo pro načítání tříd, které jsou umístěny mimo systémovou classpath. Kromě samozaváděcího class loaderu, který se stará o načtení runtime tříd při startu programu, má každý class loader svůj rodičovský class loader. Tím je ten, který ho načetl. Pro správnou funkčnost vlastního class loaderu je nezbytné nastavit správný rodičovský class loader. To je možné provést použitím prázdného konstruktoru, který vytvoří nový class loader a jeho rodičovský je

nastaven na class loader vrácený metodou getSystemClassLoader(). Druhou možností je nastavit rodičovský class loader v konstruktoru přímo následujícím způsobem:

```
public class CustomClassLoader extends ClassLoader{

  public CustomClassLoader(){
    super(CustomClassLoader.class.getClassLoader());
  }
}
```

Při požadavku na načtení třídy nejprve class loader zjistí, zda už třída nebyla načtena dříve, pokud ne, předá požadavek na svůj rodičovský class loader, pokud ten třídu nenalezne, pokusí se třídu nalézt sám. K načtení třídy slouží metoda *loadClass(String, boolean)*, která defaultně pracuje následujícím způsobem:

1. pokud je požadovaná třída už načtena, je vrácena. K nalezení načtených tříd je volána metoda *findLoadedClass(String)*.

2. pokud požadovaná třída není načtena, je volána metoda *loadClass* v rodičovském class loaderu. Pokud je rodičovský class loader null, je použit systémový class loader zabudovaný v JVM.

3. volání metody *findClass(String)*, která defaultně vyvolává výjimku ClassNotFoundException.

4. pokud je třída některým z předešlých kroků nalezena a boolean hodnota je nastavena na true, je volána metoda *resolveClass(Class)*. Tato metoda zajistí, že budou načteny i všechny odkazované třídy a provede verifikaci byte kódu načtené třídy. Pokud verifikace selže, je generována chyba LinkageError.

Pro vytvoření vlastního class loaderu tedy stačí přetížit metody loadClass(String, boolean) a findClass(String) a implementovat požadované chování.

Tento mechanismus předávání požadavku na rodičovský class loader, tedy podle [18] způsobí následující postup při vyhledávání třídy:

1. třída je vyhledávána v tzv. bootstrap třídách. Mezi tyto třídy patří například runtime třídy (rt.jar) nebo třídy z archivu i18n.jar.

2. třída je vyhledávána v lib/ext použitého JRE. Využití tohoto adresáře je možné ve verzích Java 6 a novějších.

3. třída je vyhledávána v *java.class.path*. Tato hodnota je defaultně nastavena na adresář, ve kterém je program spuštěn. Změnit ji je možné při spuštění programu přepínači -classpath nebo -cp a změnou systémové proměnné CLASHPATH.

4. pokud není třída nalezena v žádném z předchozích umístění, je volána funkce *findClass(String)*.

Nyní je potřeba zajistit načtení požadované třídy v metodě *findClass(String)*. Nejprve je nutné požadovanou třídu nalézt a získat její byte code. Umístění třídy není podstatné (lokální disk, web, atd.), důležité je získat její byte code. Jakmile je byte code získán, pomocí metody *defineClass(String, byte[], int, int)* je převeden na instanci třídy *java.lang.Class*. Pokud se nejedná o validní byte code, je generována výjimka ClassFormatError. Dalším krokem je zavolání metody *resolveClass(Class)*. Jakmile je tato třída získána, její instance se vytvoří voláním metody *newInstance()*.

Aby uvedený postup vedl k načtení vlastní třídy, nesmí se název této třídy shodovat s názvem některé systémové třídy. V takovém případě by byla načtena systémová třída. Stejně tak může nastat problém, pokud bude načítána třída umístěná mimo *java.class.path* se stejným názvem jako některá třída dostupná v *java.class.path*. V tomto případě bude načtena třída umístěná v *java.class.path*.

Vlastní class loader s sebou přináší i několik bezpečnostních rizik, kvůli kterým applety nemůžou používat svůj vlastní class loader. Pokud by například vlastní class loader nepředával nejprve požadavek na rodičovský class loader, umožnil by načíst vlastní třídu *java.lang.Object*, která by se tak stala rodičovskou třídou pro všechny ostatní. Dalším příkladem narušení bezpečnosti je načtení vlastního objektu typu *java.lang.SecurityManager*. Další problém nastává i když třídu vyhledává nejprve rodičovský class loader. Pokud například budeme načítat třídu *java.lang.Pokus*, rodičovský class loader ji nenalezne, takže bude načtena vlastním class loaderem. Tato třída potom bude mít přístup ke všem hodnotám a metodám, které nejsou označeny jako private, z balíčku java.lang. Tomuto lze zabránit kontrolou názvu načítané třídy. Pokud začíná řetězcem, který odpovídá systémovým balíčkům nebo jmenným prostorům a požadovaná třída nebyla nalezena rodičovským class loaderem, tato třída nebude načtena [22].

Dalším problémem je, jakým způsobem zajistit, aby bylo možné s načtenou třídou pracovat jako s objektem určitého typu. Metoda *loadClass(String, boolean)* vrátí objekt typu *java.lang.Class* a je tedy nutné novou instanci přetypovat na odpovídající třídu. Tady nastává problém. JVM umožňuje přetypovávat pouze mezi třídami, které mají společný alespoň jeden class pointer. Pokud je stejná třída načtena různými instancemi class loaderu, načtené třídy budou mít rozdílné class pointery a jedinou jejich společnou nadtřídou bude obvykle *java.lang.Object*. To znamená, že pokud třídu načteme systémovým class loaderem a poté za běhu programu načteme třídu znovu vlastním class loaderem, budou tyto třídy považovány za rozdílné a nebude možné dynamicky načtenou třídu přetypovat na třídu v systému už načtenou. Tento případ slouží pouze jako ukázka, v reálném programu by nastat neměl (pokud je vlastní class loader vytvořen postupem uvedeným výše). Vlastní class loader může být vytvořen pro dynamické načítání nových tříd za běhu programu. V takovém případě program nemůže dopředu vědět, které třídy budou načítány. Řešením obou těchto problémů je využití společné nadtřídy, což může být rozhraní nebo abstraktní třída.

Pokud tedy načítané třídy budou implementovat stejné rozhraní, nebudou mít sice stejné class pointery, ale budou mít stejnou nadtřídu, na kterou budou moci být přetypovány. Program do kterého budou třídy načítány musí znát pouze použité rozhraní [22].

Pokud by bylo nutné načítat za běhu programu třídy, které se mění, je nutné pro znovunačtení použít novou instanci class loaderu. V takovém případě se nesmí původní instance dále používat. Jak bylo popsáno výše, pokud je stejná třída načtena dvěma různými class loadery, jsou tyto načtené třídy považovány za rozdílné. Znovunačítání tříd za běhu programu velmi usnadní práci například při testování nějaké extrakční metody. Ta bude moci být upravována a po přeložení znovu načtena a otestována, aniž by bylo nutné provádět znova všechna nastavení.

5.2.3 Moduly pro extrakci informací

Aplikace musí umožňovat dynamické načítání modulů pro extrakci informací, proto je nutné navrhnout obecné rozhraní pro komunikaci s tímto modulem. Toto rozhraní bude tvořit společnou nadtřídu, aby bylo možné jednotlivé třídy načítat tak, jak je to popsáno v předešlé části.

Prvním problémem je způsob, jakým se budou modulu předávat vstupní dokumenty. Metody získávání informací popsané v časti 4.1 mohou pracovat s dokumentem jako s proudem znaků nebo využívají nějaké složitější reprezentace dokumentu (např. strom). Protože má být rozhraní co nejvíce obecné, nejlepším způsobem předávání vstupních dokumentů bude řetězec znaků. Tento jeden řetězec bude tedy obsahovat zdrojový kód všech vybraných dokumentů. Jednotlivé moduly si potom budou moci řetězec upravit do vyhovující podoby. Po předání vstupních dokumentů bude následovat samotná extrakce informací, která bude probíhat plně v režii modulu.

Po dokončení extrakce bude nutné předat získaná data ve vhodné podobě pro další zpracování. Tato struktura by měla odpovídat struktuře popsané v části 4.1. Pokud budeme například extrahovat data z tabulky, kde každý řádek obsahuje několik hodnot, bude nutné uchovávat význam jednotlivých hodnot. Toto je možné vyřešit použitím například pole, které bude obsahovat asociativní pole, která odpovídají jednotlivým řádkům tabulky. Takovýto postup vyhovuje pro jednoduchou tabulku. Pokud půjde o složitější strukturu (například několik vnořených tabulek), budou se muset data předávat v nějaké složitější struktuře.

Dále bude muset existovat způsob, jak informovat o úspěchu nebo neúspěchu prováděné metody. U extrahování informací je jednou z možností informovat o počtu nalezených datových polí.

5.2.4 Moduly pro ukládání dat

Stejně jako v případě modulů pro extrakci informací, musí aplikace umožňovat dynamické načítání modulů pro ukládání dat.

Prvním z nich je modul pro ukládání extrahovaných dat do souboru XML. Opět je nutné navrhnout obecné rozhraní pro komunikaci s tímto modulem. Vzhledem k tomu, že vytváření XML dokumentu bude plně v režii načteného modulu, jediné co bude nutné modulu předat, jsou data získaná některým z modulů pro extrakci informací. Dále by bylo vhodné, aby modul mohl informovat o úspěšnosti provedené operace. Implementovat vlastní moduly bude nutné jen v případě, že extrahovaná data budou mít složitou strukturu. Pro ukládání dat s jednoduchou strukturou bude možné použít nějaký univerzální modul, který bude umět pracovat s pevně danou strukturou dat. V takovém případě bude nutné spolu s daty předat další informace o vytvářeném souboru (jméno souboru a jeho umístění, názvy elementů atd.).

Dalším typem je modul pro ukládání získaných dat do databáze. Stejně jako u předešlých modulů, je nutné navrhnout rozhraní pro komunikaci. Podobně jako při ukládání do XML souboru, jediné co je nutné modulu předat, jsou data, která se mají uložit. Ostatní informace mohou být opět uvedeny přímo ve zdrojovém kódu. Pro větší pružnost by ovšem bylo vhodné, kdyby bylo opět možné použít nějaký univerzální modul, který by umožňoval nastavení parametrů pro připojení k databázi jako je adresa a port serveru, jméno databáze a tabulky do které se mají data uložit a uživatelské jméno a heslo pro připojení k databázi. Takovýto univerzální modul ovšem opět nebude moci být použit pro všechny případy (data se složitou strukturou, různé typy databází, atd.).

U všech typů modulů by bylo pro lepší orientaci při práci s nimi vhodné, aby bylo možné získat jejích popis apod.

5.2.5 Dávkové zpracování

Pro efektivní práci s jednotlivými moduly by měl systém umožňovat dávkové zpracování. Jednou z možností je načtení XML souboru, který by obsahoval postupy, které se mají provést. Struktura takového souboru by měla umožňovat ke každému modulu pro extrakci informací přiřadit všechny potřebné vstupní dokumenty, ať už z lokálního disku nebo z WWW a všechny moduly pro ukládání dat, které se mají použít pro uložení výstupních dat. Dále je nutné zadat umístění jednotlivých modulů, aby mohly být systémem načteny.

Problémem při dávkovém zpracování by mohl nastat v případě, že některé dílčí operace skončí neúspěšně. V takovém případě je potřeba definovat jak se bude pokračovat. Například pokud se nezdaří načtení žádného vstupního dokumentu, nemá smysl pokračovat v provádění. To stejné platí o ukládání výstupních dat. Pokud se například nezdaří načíst jeden ze vstupních dokumentů, je nutné definovat, zda se bude pokračovat nebo se celý postup spustí znova. To stejné platí při ukládání výstupních dat. Pokud se jedna z operací nezdaří, bylo by vhodné zopakovat pouze nezdařenou operaci.

Kapitola 6

Návrh

Tato kapitola se věnuje návrhu aplikace. V první části je uveden diagram balíčků, který zobrazuje základní strukturu aplikace. Následuje blokové schéma, které dělí aplikaci do logických celků, které jsou následně popsány.

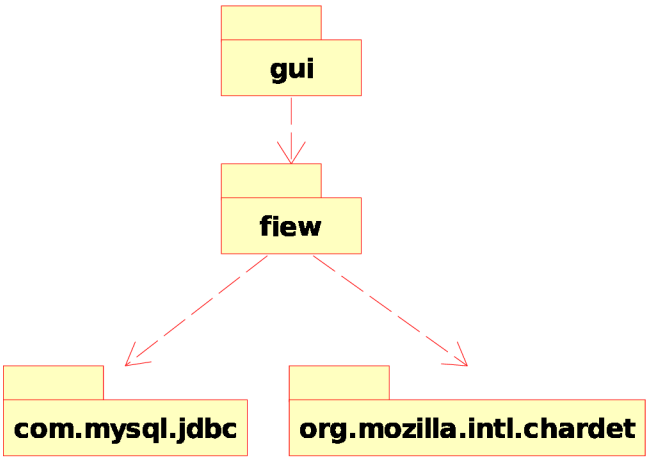
6.1 Diagram balíčků

Diagram balíčků uvedený na obrázku 6.1 zobrazuje základní strukturu aplikace. Obsahuje balíčky, které aplikaci tvoří (gui, fiew) a balíčky aplikací používané (com.mysql.jdbc, org.mozilla.intl.chardet), které nejsou dostupné ve standardních knihovnách jazyka java edice Java SE 6. Balíček gui obsahuje třídy tvořící grafické uživatelské rozhraní a třídý, které využívají nabízených metod z balíčku fiew. Ten je tvořen třídami pro načítání vstupních dokumentů, pro dynamické načítání modulů a pro práci s těmito moduly a dokumenty. Dále obsahuje třídy, které se starají o dávkové zpracování. Tento balíček představuje knihovnu, se kterou se pracuje prostřednictvím třídy Controller, která je popsaná v části 6.4.7. Balíček com.mysql.jdbc je touto knihovnou využíván pro práci s databází mysql. Knihovna dále využívá balíček org.mozilla.intl.chardet, který slouží pro odhad použitého kódování na základě porovnávání znaků s využitím statistik a heuristik. Tento přístup je popsán v části 5.2.1. Popis důležitých tříd z balíčku fiew je uveden v následujících častech.

6.2 Základní blokové schéma aplikace

Základní blokové schéma aplikace je uvedeno na obrázku 6.2. Zobrazuje především strukturu balíčku fiew, který je uveden v diagramu balíčků v předešlé části. Navázání na baliček gui je znázorněno ve spodní části schématu.

Prvním blokem je získání a zpracování dokumentu. Úkolem tohoto bloku je získání požadovaného dokumentu v textové podobě. Na základě příkazu z bloku řízení, je načten dokument z lokálního disku nebo z WWW a vrácen do řídícího bloku jako textový řetězec. V obou případech bude provedena detekce kódování postupem popsaným v části 5.2.1.



Obrázek 6.1: Diagram balíčků

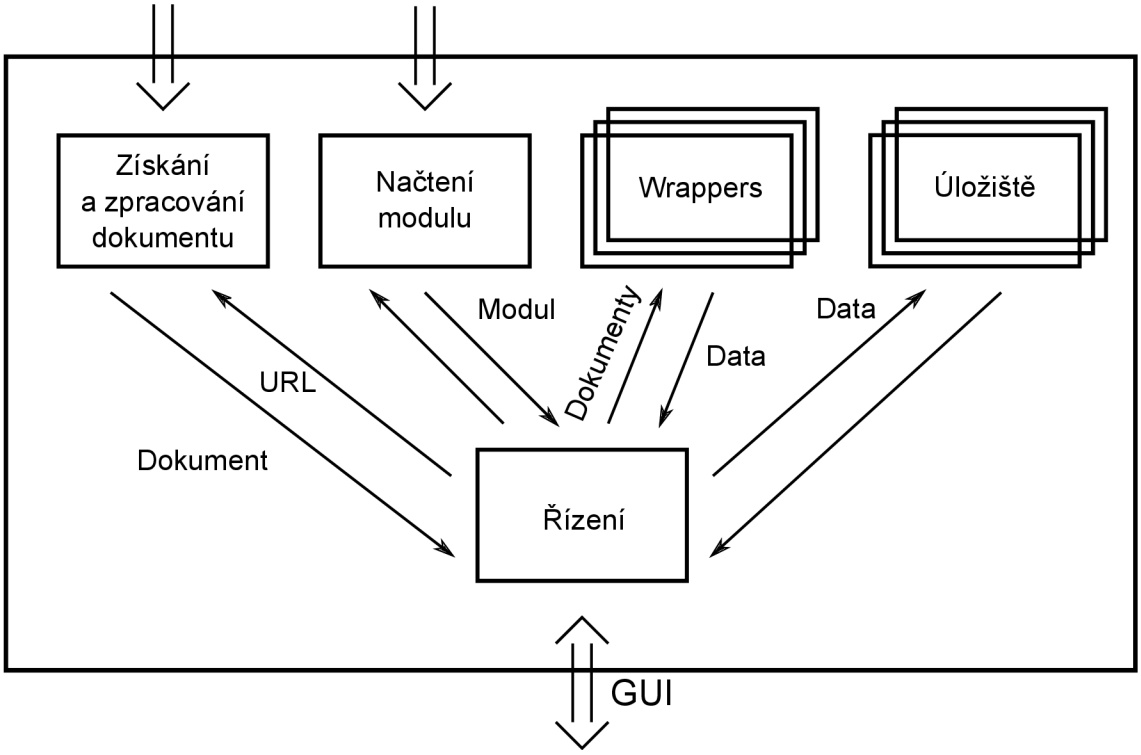
Dokument z WWW bude získáván na základě jeho URL. Předpokládá se využití protokolu HTTP. Dokumenty z lokálního disku budou načítány na základě řetězce s absolutní cestou k tomuto dokumentu.

Blok načtení modulu, se stará o dynamické načítání modulů. V systému budou celkem tři typy modulů. Prvním typem budou jednotlivé extrakční metody, druhým typem budou moduly pro ukládání výstupních dat do souboru XML a posledním typem budou moduly pro ukládání výstupních dat do databáze. Pro každý typ modulu bude navrženo rozhraní, které se budu používat pro komunikaci. Na základě volání z řídícího bloku bude z lokálního disku načten jeden z modulů a bude vrácen jako odpovídající rozhraní. Načtené moduly nebudou knihovnou udržovány, toto je ponecháno na nadřazeném systému, který ji bude používat (v případě této aplikace to je grafické uživatelské rozhraní).

Blok wrappers představuje jednotlivé načtené moduly pro extrakci informací. Jak bylo řečeno výše, knihovna neslouží k udržování modulů, řídící blok pouze spouští moduly předané z obslužného systému. Spolu s modulem je nutné předat vstupní textový řetězec, se kterým bude modul pracovat a moduly, které se mají použít pro uložení výstupu.

Blok úložiště představuje jednotlivé načtené moduly pro uložení výstupu do souboru XML nebo do databáze. Udržování těchto modulů je opět ponecháno na nadřazeném systému. Tyto moduly jsou z nadřazeného systému předávány řídícímu bloku spolu se vstupním textovým řetězcem a s modulem pro extrakci informací, pro který jsou určeny.

Řídící blok slouží jako rozhraní pro nadřazený systém, který tuto knihovnu využívá. V případě této aplikace jde o grafické uživatelské rozhraní. Toto rozhraní bude podrobně popsáno v další části.



Obrázek 6.2: Blokové schema systému

6.3 Vnitřní reprezentace získaných dat

Aby bylo možné předávat data získaná modulem pro extrakci informací do modulů, které se postarají o jejich uložení do XML souboru nebo do databáze, je nutné definovat v jaké podobě tyto data budou. Tím že je možné implementovat vlastní moduly pro ukládání dat, bude vhodné použít co nejobecnější formát, aby bylo možné pracovat i se složitými strukturami. Ideální pro předávání dat bude objekt typu *java.util.Vector*. Ten představuje dynamické pole objektů. Objekty z tohoto pole jsou přístupné indexováním, nebo je možné projít celé pole za využití iterátoru. Jednotlivé objekty v tomto poli pak budou představovat záznamy získané modulem pro extrakci informací. Ten tedy bude moci použít libovolnou reprezentaci záznamů, což umožní i přenášení složitých struktur. V takovém případě bude muset být implementován modul pro uložení těchto dat, který bude umět s použitou strukturou pracovat. Takovýto přístup dává naprostou volnost. Modul pro uložení dat tedy dostane vytvořený vektor a způsob jakým ho zpracuje už je čistě v jeho režii. Tento přístup velice zjednoduší práci se získanými daty. Pokud bychom chtěli implementovat obecný modul pro vytváření XML souborů, byl by to netriviální úkol. Bylo by nutné nějakým způsobem popsat jak má výstupní soubor vypadat (názvy elementů, atributů, struktura, atd.) a jak do něj uložit získaná data (která data zobrazit v elementech, která v atributech, atd.). V případě ukládání získaných dat do databáze nastává problém ještě větší. Není možné im-

plementovat modul, který by umožňoval ukládání dat do všech existujících typů databází. Z těchto důvodů bude v knihovně implementován základní modul pro tvorbu XML souborů pouze s jednoduchou strukturou. Implementace modulů pro tvorbu složitějších XML dokumentů bude ponechána na uživateli systému. To stejné platí pro ukládání dat do databáze. V knihovně bude implementován pouze základní modul pro ukládání dat s jednoduchou strukturou do databáze typu mysql. Implementace modulů pro ukládání do jiných typů databází je ponechána na uživateli systému.

Jak bylo popsáno výše, aby nebylo nutné implementovat vlastní moduly pro ukládání nalezených dat, které mají jednoduchou strukturu, knihovna obsahuje připravené moduly, které lze pro jejich uložení využít. Jednoduchou strukturou se myslí struktura popsaná v části 4.1. Je to tedy pole asociativních polí. Jako asociativní pole bude využit objekt typu *java.util.LinkedHashMap*. Implementované metody pro ukládání získaných dat tedy očekávají objekt typu *java.util.Vector*, který bude obsahovat objekty typu *java.util.LinkedHashMap*. Podrobnější popis těchto základních modulů je uveden v části 6.4.6.

6.4 Diagram tříd

V této části jsou uvedeny a popsány diagramy tříd z balíčku *fiew*. Nejprve je uvedena použitá notace, poté přehledový diagram tříd, který obsahuje všechny třídy z balíčku *fiew*. V dalších částech jsou uvedeny podrobnější diagramy, které zobrazují vždy nějaký logický celek.

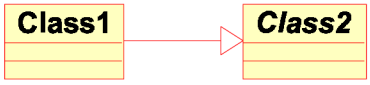
6.4.1 Použitá notace

U diagramů tříd v následujících částech je použita tato notace:

Obrázek 6.3: Asociace

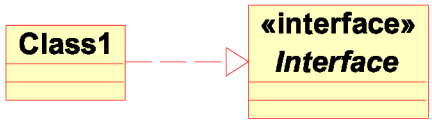
Jednoduchá šipka mezi třídami na obrázku 6.3 znamená, že třída *Class1* používá třídu *Class2*. Kardinalita není v návrhu řešena.

Dědičnost je znázorněna na obrázku 6.4. V tomto případě třída *Class1* dědí třídu *Class2*. Pokud se jedná o abstraktní třídu, je její název zadán kurzívou.



Obrázek 6.4: Dědičnost

Pokud některá třída implementuje rozhraní, je to znázorněno diagramem na obrázku 6.5. V tomto případě třída *Class1* implementuje rozhraní *Interface*.



Obrázek 6.5: Rozhraní

6.4.2 Přehledový diagram tříd

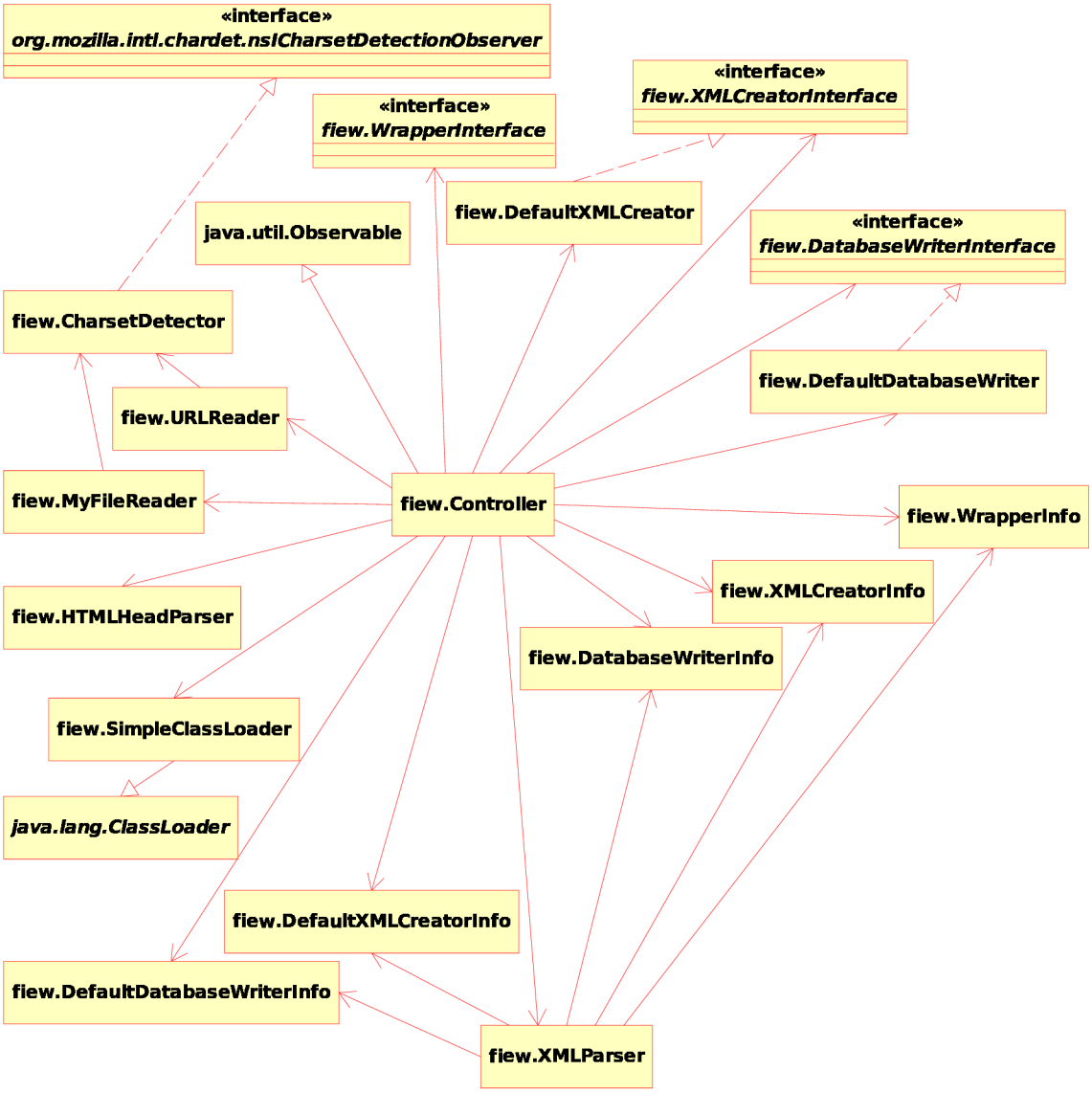
Diagram tříd na obrázku 6.6 ukazuje spolupráci tříd v balíčku fiew. Slouží pouze jako přehled, důležité třídy jsou zobrazené v podrobnějších diagramech, které jsou uvedeny v následujících částech.

Třída Controller odpovídá řídícímu bloku ze schématu uvedenému v předešlé části. Tato třída představuje rozhraní tohoto balíčku. Pro načtení dokumentu z WWW slouží třída URLReader, která využívá pro detekci kódování třídy CharsetDetector. Třída MyFileReader také využívá CharsetDetector pro zjištění použitého kódování a slouží pro načtení souboru z lokálního disku. Tato část je podrobněji popsaná v 6.4.3. Třída HTMLHeadParser slouží pro zjištění dostupných informací z hlavičky HTML dokumentu. Tato třída parsuje hlavičku pomocí regulárních výrazů. Extrahuje informace z elementu *title* a z elementů *meta*. V těchto elementech jsou zjišťovány klíčová slova, popis, autor a použité kódování. Třída SimpleClassLoader slouží k dynamickému načítaní tříd. Podrobnější popis této třídy je uveden v části 6.4.4. Jednotlivá rozhraní uvedená v diagramu jsou popsaná v části 6.4.5. Třída XMLParser slouží k načtení XML souboru pro dávkové zpracování a jeho parsování. Pro udržení informací načtených z tohoto souboru o jednotlivých modulech, využívá pomocné třídy WrapperInfo, XMLCreatorInfo, DatabaseWriterInfo, DefaultXMLCreatorInfo a DefaultDatabaseWriterInfo. Třídy DefaultXMLCreator a DefaultDatabaseWriter představují defaultní moduly pro uložení výstupních dat do XML souboru nebo do databáze. Tyto třídy jsou podrobněji popsány v části 6.4.6. Třída Controller je potomkem třídy java.utril.Observable, takže umožňuje zaregistrovat posluchače, které informuje o probíhajících operacích. Toho lze využít pro zasílání zpráv například grafickému rozhraní.

6.4.3 Třídy pro načítání dokumentů

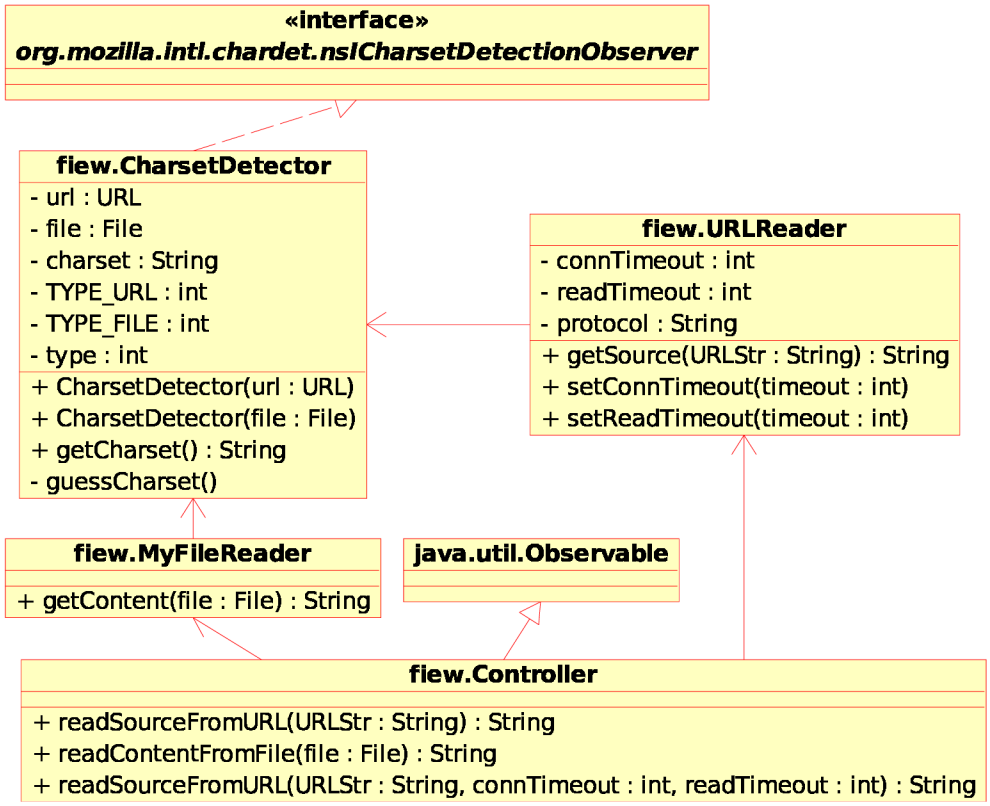
Na obrázku 6.7 jsou zobrazeny třídy, které se starají o načítání vstupních dokumentů.

O načítání dokumentů z lokálního disku se stará třída *MyFileReader*. Tato třída obsahuje metodu pro získání dokumentu, která vrací načtený dokument jako objekt typu *java.lang.String*. Pokud nastala chyba a dokument není načten, metoda vyvolává výjimku. Požadovaný dokument jí je předáván jako instance třídy *java.io.File*. Pro detekci kódování



Obrázek 6.6: Diagram tříd balíčku fiew

je využívána třída *CharsetDetector*. Ta se dá využít jak pro detekci kódování u dokumentů načítaných z disku, tak pro dokumenty načítané z WWW. Možné způsoby jak detekovat kódování jsou popsány v části 5.2.1. Metoda *getCharset* vrací nejpravděpodobnější kódování jako objekt typu *java.lang.String* nebo *null*, pokud došlo k chybě. Tato třída se nejprve pokusí získat kódování parsováním HTML hlavičky, pokud neuspěje, použije se knihovna *chardet.jar*, která slouží k odhadu použitého kódování za použití statistik. Z tohoto důvodu tato třída implementuje rozhraní *org.mozilla.intl.chardet.nsICharsetDetectionObserver*. Třída *URLReader* je určena k načítání dokumentů z WWW s využitím protokolu HTTP. Metoda *getSource* na základě zadaného URL vrací načtený obsah jako objekt typu *java.lang.String*. Pokud došlo k chybě, metoda vyvolává výjimku. Metody *setConnTimeout* a *setReadTime-*



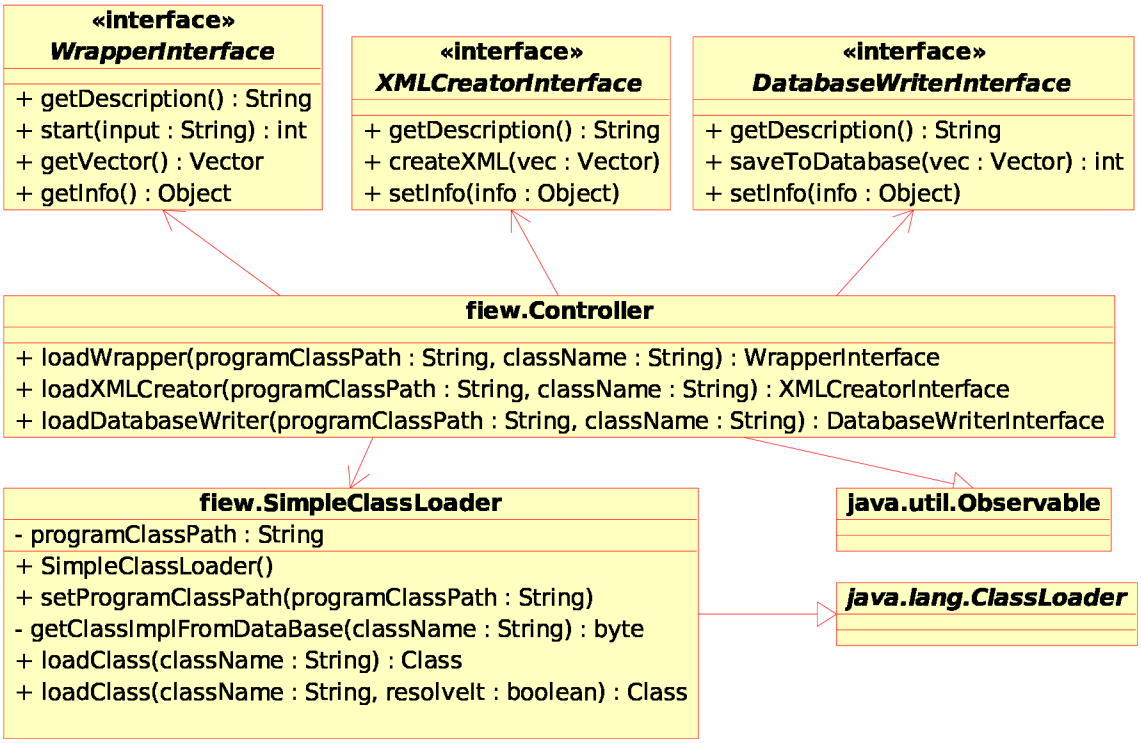
Obrázek 6.7: Diagram tříd z pohledu načítání dokumentů

out slouží k nastavení timeoutu při načítání dokumentu. V prvním případě jde o čas, po který se čeká při pokusu o připojení k serveru, v druhém případě o čas, po který se čeká při načítání dat. Defaultní hodnoty jsou nastaveny na 5000 ms. Poslední třída v diagramu je *Controller*. Tato třída je využívaná jako rozhraní pro komunikaci s nadřazeným systémem, který využívá jejích metod. Na tomto diagramu jsou uvedeny pouze metody, které jsou využívány pro načítání vstupních dokumentů.

6.4.4 Třídy pro načítání modulů

Diagram tříd na obrázku 6.8 zobrazuje třídy využívané pro dynamické načítání modulů.

Třída *SimpleClassLoader* slouží pro načtení třídy z lokálního disku. Je potomkem abstraktní třídy *java.util.ClassLoader*. Název požadované třídy je předáván jako objekt typu *java.lang.String*. Je očekáván celý název včetně balíčků (například mujBalik.mojeTrida). K načtení třídy slouží metoda *loadClass*. Ta se nejprve pokusí nalézt požadovanou třídu ve své lokální paměti dříve načtených tříd, pokud se jedná o ještě nenačtenou třídu, předá se požadavek na systémový class loader (přesný postup, jakým systémový class loader hledá požadovanou třídu je popsán v části 5.2.2), pokud třídu nenalezne, je volána metoda *getClassImplFromDataBase*. Tato metoda se pokusí načíst požadovanou třídu z lokálního



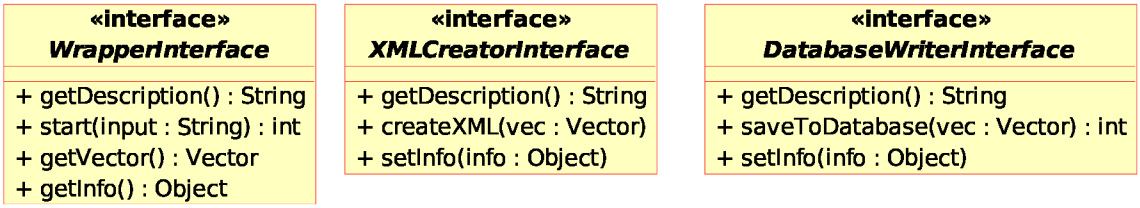
Obrázek 6.8: Diagram tříd z pohledu načítání modulů

disku, pokud ji nalezne, vrátí ji jako pole bajtů, pokud ne, vrací null. Prohledávání probíhá na základě názvu třídy a zadané programové class path, která se nastavuje metodou *setProgramClassPath*. Programová class path je objekt typu *java.lang.String*, který obsahuje cesty k adresářům, ve kterých se bude požadovaná třída hledat. Jakmile je třída v některém adresáři nalezena, prohledávání končí. K oddělení jednotlivých cest musí být použit systémově závislý oddělovač *java.io.File.pathSeparator* (operační systém windows používá středník, UNIX používá dvojtečku). Stejně tak zápis cest musí odpovídat stylu použitého operačního systému. Vrácené pole bajtů je převedeno metodou *defineClass* na objekt typu *java.lang.Class*. Dále je použita metoda *resolveClass*, která zajistí, že budou načteny i třídy, na které se načítaná třída odkazuje. Při chybě, obě tyto metody vyvolávají výjimku. Podrobný popis implementace vlastního class loaderu je popsán v 5.2.2. Načtená třída je vrácena jako obecný typ *java.lang.Class*. Jednotlivé metody pro načítání modulů z třídy *Controller* provedou přetypování vrácené třídy na odpovídající rozhraní a vytvoří instanci, která je metodou vrácena.

V tomto diagramu jsou u třídy *Controller* opět zobrazeny pouze metody využívané pro načítání modulů.

6.4.5 Rozhraní modulů

Na obrázku 6.9, jsou uvedeny rozhraní jednotlivých modulů. Pro zajištění správné funkčnosti, musí jednotlivé moduly tato rozhraní dodržovat.



Obrázek 6.9: Rozhraní modulů

Prvním z nich je *WrapperInterface*. Toto rozhraní slouží pro komunikaci s modulem pro extrakci informací. Metoda *getDescription* slouží pro získání popisu modulu. Toho lze využít například v grafickém uživatelském rozhraní, pro zlepšení přehlednosti při práci s více moduly. Další metodou je *start*. Této metodě se předá objekt typu *java.lang.String*, který představuje všechny vstupní dokumenty, ve kterých se bude vyhledávat. Vrácená hodnota odpovídá počtu nalezených záznamů. Tyto záznamy se získají pomoci metody *getVector*, která vrací objekt typu *java.util.Vector*. Způsob, jakým je tento vektor vytvořen a co za objekty obsahuje, je ponecháno na modulu. Pro správnou funkčnost je potom nutné vytvořit modul pro uložení výstupních dat, který bude umět s vytvořeným vektorem pracovat. Pokud se pro uložení dat použije některý z defaultních modulů, je nutné, aby struktura tohoto vektoru odpovídala struktuře popsané v části 6.3. Metoda *getInfo* vrací objekt, který obsahuje informace pro modul, který se použije pro uložení výstupních dat. Předáním tohoto objektu bude možné, aby více různých modulů pro extrakci informace používalo pro uložení získaných dat jeden modul. Například pokud se použije modul pro vytváření XML souboru, může se mu tímto objektem předat informace o tom, jak má výstupní soubor vypadat a kam se má uložit.

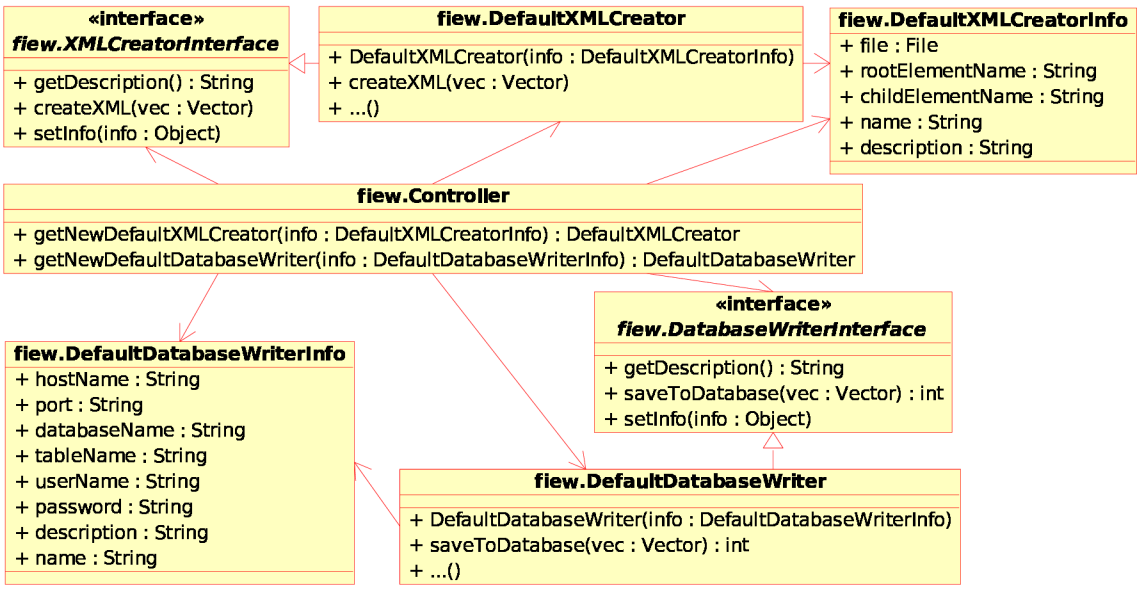
Rozhraní *XMLCreatorInterface* slouží pro komunikaci s modulem, který zapisuje získaná data do XML souboru. Opět obsahuje metodu pro získání popisu daného modulu. Metodě *createXML* je předán objekt typu *java.lang.Vector*, který obsahuje záznamy nalezené modulem pro extrakci informaci. Tyto záznamy jsou potom uloženy do souboru XML. Způsob jakým je tento soubor vytvořen je ponechán na modulu. Pokud při vytváření souboru nastala chyba, je generována výjimka. Metoda *setInfo* slouží pro předání objektu, jehož využití bylo popsáno v předešlé části.

Posledním rozhraním je *DatabaseWriterInterfase*, které slouží pro komunikaci s modulem pro uložení získaných dat do databáze. Stejně jako u předešlých rozhraní obsahuje metodu pro získání popisu modulu. Metoda *saveToDatabase* uloží data z předaného vektoru do databáze. Jakým způsobem data uloží je opět ponecháno na modulu. Pokud vše proběhlo v pořádku, je vrácen počet uložených záznamů. Pokud nastala při ukládání chyba,

je generována výjimka. Metoda *setInfo* má stejné využití jako u předešlého rozhraní.

6.4.6 Defaultní moduly pro uložení výstupních dat

Aby nebylo nutné ke každému modulu pro extrakci informací implementovat speciální modul pro uložení získaných dat, jsou implementovány dva defaultní. Diagram tříd je uveden na obrázku 6.10. Jak bylo uvedeno v části 6.3, tyto moduly slouží pro uložení dat s jednoduchou strukturou. Tato struktura odpovídá objektu typu *java.util.Vector*, který obsahuje objekty typu *java.util.LinkedHashMap*, které odpovídají jednotlivým nalezeným záznamům. Každý záznam může mít několik prvků. Pokud budeme například získávat data z tabulky, každý řádek odpovídá jednomu objektu *java.util.LinkedHashMap*, ve kterém je každý název sloupce namapován na odpovídající hodnotu sloupce. Dodržování této struktury je nutné pro správnou funkčnost těchto defaultních modulů.



Obrázek 6.10: Diagram tříd defaultních modulů

Prvním z nich je *DefaultXMLCreator*, který slouží pro vytvoření XML souboru. Pomocná třída *DefaultXMLCreatorInfo* obsahuje všechny parametry modulu. Parametry *name* a *description* mají význam při práci s modulem v grafickém rozhraní pro lepší orientaci. Pokud nejsou zadány, obsahují defaultní hodnoty. U dávkového zpracování se nepoužívají. Parametry *rootElementName* a *childElementName* odpovídají názvu kořenového elementu a prvního vnořeného elementu ve struktuře vytvořeného XML souboru tak, jak to je zobrazeno v ukázce níže. Pokud nejsou tyto parametry vyplněny, použijí se defaultní hodnoty. Parametr *file* určuje název a umístění generovaného XML souboru. Ke všem těmto parametrům jsou v třídě *DefaultXMLCreator* gettery a settery, což je v diagramu naznačeno třemi tečkami. Samotné generování XML souboru je spuštěno metodou *createXML*, které se

předá objekt typu *java.util.Vector*. Pokud nastane chyba, je vygenerována výjimka. Struktura vytvořeného XML bude vypadat následujícím způsobem:

```
<?xml version="1.0" encoding="utf-8"?>
<rootElement>
    <childElement>
        <key1>value</key1>
        <key2>value</key2>
        .
        .
    </childElement>
    .
    .
</rootElement>
```

Verze vytvořeného XML souboru je 1.0, kódování utf-8. Každý childElement odpovídá jednomu objektu typu *java.util.LinkedHashMap*. Názvy vnořených elementů key1 až keyn odpovídají názvům jednotlivých klíčů použitých pro mapování. Hodnoty těchto elementů odpovídají hodnotám odpovídajících klíčů.

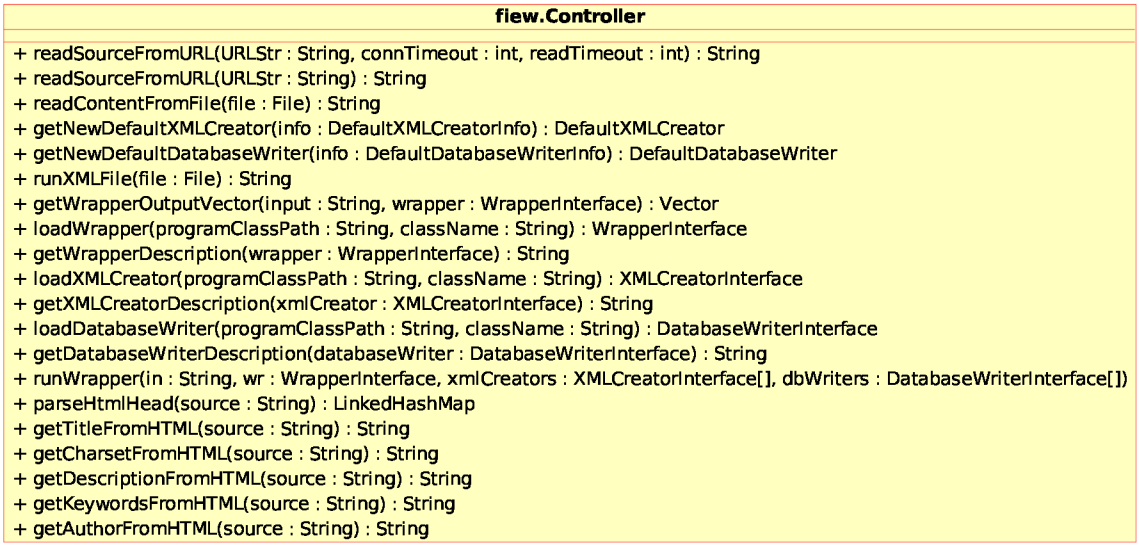
Dalším modulem je *DefaultDatabaseWriter*. Ten slouží pro uložení získaných hodnot do databáze mysql. Stejně jako u předešlého modulu, je i zde použita pomocná třída pro snadné přenášení potřebných parametrů. Parametry *name* a *description* mají stejný význam jako u předešlého modulu. Použití ostatních parametrů je zřejmé z jejích názvu. Ke všem parametrům jsou opět v třídě *DefaultDatabaseWriter* gettery a settery. Pro uložení dat do databáze slouží metoda *saveToDatabase*, které se předá objekt typu *java.util.Vector*. Pokud při ukládání nastala chyba, je generována výjimka, pokud proběhlo vše v pořádku, vrací počet uložených řádků do tabulky. Při ukládání se předpokládá, že je v databázi vytvořena odpovídající tabulka, kde názvy sloupců odpovídají použitým klíčům pro mapování v objektu *java.util.LinkedHashMap*. Jednotlivé hodnoty jsou ukládány do tabulky jako textové řetězce.

I v tomto diagramu jsou u třídy *Controller* opět zobrazeny pouze metody využívané pro tuto část.

6.4.7 Controller

Jak už bylo uvedeno v předešlých částech, tato třída představuje rozhraní balíčku fiew. Pokud je tedy tento balíček používán, měly by se využívat především metody, které nabízí tato třída. Diagram je uvedený na obrázku 6.11.

Metoda *readSourceFromURL* slouží pro načtení dokumentu z WWW prostřednictvím protokolu HTTP. Umístění dokumentu je dáno objektem typu *java.lang.String*, který obsahuje URL. Načtený dokument je vrácen jako objekt typu *java.lang.String*. Metoda *readContentFromFile* slouží pro načtení dokumentu z lokálního disku. Umístění dokumentu je



Obrázek 6.11: Třída controller

dáno objektem typu *java.io.File*. Načtený dokument je vrácen stejně jako u předšlé metody. Obě metody při chybě vyvolávají výjimku. Podrobnější popis načítání dokumentů je uveden v 6.4.3.

Metody *getNewDefaultXMLCreator* a *getNewDefaultDatabaseWriter* vrací nové instance uvedených tříd. Podrobný popis těchto tříd je v 6.4.6.

Metoda *runXMLFile* zajišťuje dávkové zpracování zadané souborem, jehož umístění je dáno objektem typu *java.io.File*. Tato metoda informuje zaregistrované posluchače o průběhu jednotlivých operaci. Více o dávkovém zpracování v 6.5.

Metoda *getWrapperOutputVector* vrací objekt typu *java.util.Vector*. Tento vektor představuje data, která nalezl zadaný modul pro extrakci informace v zadaném vstupním souboru. Tento modul je předáván jako objekt typu *fiew.WrapperInterface*. Více o formátu získávaných dat v 6.3.

Metody *loadWrapper*, *loadXMLCreator* a *loadDatabaseWriter* slouží pro načtení odpovídajících modulů z lokálního disku na základě zadané class path a názvu třídy. Načítání modulů je podrobně popsáno v části 6.4.4.

Metoda *runWrapper* zajistí spuštění modulu pro extrakci informací a uložení nalezených dat. Parametr *in* představuje dokumenty, ve kterých bude probíhat extrakce informací. Parametr *wr* představuje modul pro extrakci informací, který se má použít. Další dva parametry obsahují moduly, které se mají použít pro uložení nalezených dat. Tato metoda informuje zaregistrované posluchače o průběhu jednotlivých operací.

Následující metody v diagramu slouží pro zjišťování informací z HTML hlavičky.

6.5 Dávkové zpracování

Dávkové zpracování slouží pro efektivní praci s vytvořenými moduly. Operace, které mají být provedeny, je možné systému předat prostřednictvím XML souboru. Tímto způsobem je možné například definovat, se kterými dokumenty má modul pro extrakci informací pracovat a které moduly se mají použít pro uložení nalezených dat apod. Pro správnou funkčnost musí struktura toho souboru odpovídat navrženému DTD (Document Type Definition), který vypadá následujícím způsobem:

```
<?xml version="1.0" encoding="utf-8"?>
<!ELEMENT wrappers (wrapper+)>
  <!ELEMENT wrapper (inputs,outputs)>
  <!ATTLIST wrapper
      classpath CDATA #REQUIRED
      classname CDATA #REQUIRED>
    <!ELEMENT inputs (file|page)+>
      <!ELEMENT file EMPTY>
      <!ATTLIST file
          path CDATA #REQUIRED>
      <!ELEMENT page EMPTY>
        <!ATTLIST page
          url CDATA #REQUIRED>
    <!ELEMENT outputs (XMLcreator|databaseWriter|defaultXMLcreator|
      defaultDatabaseWriter)+>
      <!ELEMENT XMLcreator EMPTY>
      <!ATTLIST XMLcreator
        classpath CDATA #REQUIRED
        classname CDATA #REQUIRED>
      <!ELEMENT databaseWriter EMPTY>
      <!ATTLIST databaseWriter
        classpath CDATA #REQUIRED
        classname CDATA #REQUIRED>
      <!ELEMENT defaultXMLcreator EMPTY>
      <!ATTLIST defaultXMLcreator
        filepath CDATA #REQUIRED
        rootElementName CDATA #IMPLIED
        childElementName CDATA #IMPLIED>
      <!ELEMENT defaultDatabaseWriter EMPTY>
      <!ATTLIST defaultDatabaseWriter
        hostName CDATA #REQUIRED
        port CDATA #IMPLIED
        databaseName CDATA #REQUIRED
        tableName CDATA #REQUIRED
        userName CDATA #REQUIRED
        password CDATA #REQUIRED>
```

Kořenový element může obsahovat jeden nebo více elementů *wrapper*. Tyto elementy odpovídají jednotlivým modulům pro extrakci informace a mají dva povinné atributy, které

určují umístění třídy na lokálním disku. Každý tento element musí obsahovat jeden element *inputs* a jeden element *outputs* v uvedeném pořadí. Element *inputs* musí obsahovat minimálně jeden z elementů *file* nebo *page*. Ty slouží pro určení vstupních dokumentů, se kterými bude modul pracovat. Element *file* má jeden povinný atribut *path*, který určuje umístění dokumentu na lokálním disku. Element *page* ma jeden povinný atribut *url*, který obsahuje URL požadovaného dokumentu, který se získá prostřednictvím protokolu HTTP. Oba tyto elementy musí být prázdné. Element *outputs* musí obsahovat minimálně jeden z elementů *XMLcreator*, *databaseWriter*, *defaultXMLcreator* nebo *defaultDatabaseWriter*. Element *XMLcreator* představuje modul pro uložení nalezených dat do souboru XML. Element *databaseWriter* představuje modul pro uložení nalezených dat do databáze. Oba tyto elementy mají dva povinné atributy se stejným významem jako u elementu *wrapper* a musejí být prázdné. Element *defaultXMLcreator* představuje defaultní modul pro vytváření XML souborů. Má jeden povinný atribut *file*, který určuje název a umístění vygenerovaného souboru a dva volitelné atributty. Element *defaultDatabaseWriter* představuje defaultní modul pro uložení nalezených dat do databáze MySQL. Oba tyto elementy musí být prázdné. Podrobný popis těchto defaultních modulů a jejich atributů, které z části odpovídají atributům uvedených elementů, je uveden v části 6.4.6.

Pro lepší pochopení jsou v příloze uvedeny ukázky validních XML dokumentů s tímto DTD. Třída která se stará o parsování XML dokumentu je validující a pokud se nejedná o validní XML dokument, je generována výjimka.

6.6 Uživatelské rozhraní

Jak bylo uvedeno v sekci 6.1, aplikace se skládá ze dvou částí. První z nich tvoří knihovna *fiew*, které se věnovala předešlá část této kapitoly. Druhou část aplikace tvoří grafické uživatelské rozhraní, které tuto knihovnu využívá.

Hlavním cílem je vytvořit intuitivní grafické rozhraní pro snadnou práci s knihovnou a udržování načtených dokumentů a modulů tak, aby bylo možné spolupracující části různě kombinovat.

Hlavní okno aplikace je rozděleno do čtyř částí. V první jsou udržovány načtené moduly a dokumenty ve stromové struktuře. V druhé části je zobrazován obsah načtených dokumentů. Třetí část slouží pro zobrazení dostupných informací o označeném modulu nebo dokumentu. V poslední části jsou vypisovány průběhy operací například při dávkovém zpracování.

Pro udržení načtených dokumentů a modulů jsou využity objekty typu *java.util.HashMap*. U dokumentů načtených z WWW se jako klíč používá URL, u dokumentů z lokálního disku je klíčem absolutní cesta k dokumentu. Tyto klíče jsou namapovány na objekty typu *java.lang.String*, které obsahují odpovídající dokumenty. U modulů pro extrakci informace je namapován název třídy na objekt typu *fiew.WrapperInterface*. Název třídy slouží jako klíč i u modulů pro uložení nalezených dat do XML souboru a u modulů pro uložení nalezených dat do databáze. V prvním případě je to tedy mapování názvu třídy na objekt

typu *fiew.XMLCreatorInterface* a v druhém na objekt typu *fiew.DatabaseWriterInterface*. U defaultních modulů pro uložení nalezených dat jako klíč slouží jedinečný zadaný název těchto modulů. U defaultních modulů pro uložení nalezených dat do XML souboru je to tedy mapování názvu modulu na objekt typu *fiew.DefaultXMLCreator*, u defaultních modulů pro uložení dat do databáze je název mapován na objekt typu *DefaultDatabaseWriter*. Všechny klíče jsou objekty typu *java.lang.String*.

Všechny tyto moduly a dokumenty jsou zobrazeny v odpovídající stromové struktuře. U všech dokumentů, modulů pro extrakci informace a modulů pro uložení nalezených dat je možné načtení aktuální verze. U defaultních modulů pro uložení získaných dat je možné editovat zadané parametry.

Pro snadnou práci s dokumenty a moduly je k dispozici průvodce, který zajistí výběr modulu pro extrakci informace, výběr dokumentů, které budou sloužit jako vstup pro tento modul a výběr modulů, které se použijí pro uložení nalezených dat.

Kapitola 7

Implementace

Aplikace byla vyvíjena v NetBeans IDE 6.5. Jedná se o přenositelnou desktopovou aplikaci, implementovanou s využitím Java Platform, Standard Edition (Java SE).

Jak bylo uvedeno v sekci 6.1 v předešlé kapitole, aplikace se skládá ze dvou balíčků. Balíček *fiew* tvoří samostatnou knihovnu, která je implementována podle návrhu z předešlé části. Balíček *gui* využívá tuto knihovnu a obsahuje třídy grafického uživatelského rozhraní, třídy pro udržení načtených dokumentů a modulů a třídy pro snadnou práci s nimi. Třídy tvořící grafické uživatelské rozhraní byly vytvořeny za pomocí form builderu, který je součástí vývojového prostředí NetBeans 6.5. Využity byly komponenty Swing. Zdrojové kódy všech tříd z obou balíčků a kompletní programová dokumentace jsou na CD přiloženém k této práci. Pro tvorbu programové dokumentace byl využit nástroj JavaDoc.

Pro snadnou práci s knihovnou *fiew* byl vytvořen archiv *fiew.jar*, který obsahuje všechny třídy knihovny. Stejně tak pro snadné spouštění celé aplikace byl vytvořen archiv *FiewGUI.jar*. Pro běh aplikace jsou nutné knihovny *fiew*, *jchardet* a *MySQL JDBC Driver*, které jsou přiloženy ke zdrojovým kódům aplikace jako archivy *fiew.jar*, *chardet.jar* a *mysql-connector-java-5.1.6-bin.jar*. Návod na spuštění aplikace je součástí manuálu, který je uveden v příloze. *Jchardet* je využívána pro detekci kódování a *MySQL JDBC Driver* se používá pro práci s databází MySQL.

Při vývoji byly použity různé návrhové vzory. Observer byl použit pro předávání informací o průběhu jednotlivých operací při dávkovém zpracování. Třída *controller* slouží jako jednoduché rozhraní pro práci s knihovnou *fiew*, což odpovídá návrhovému vzoru Facade. Pokud bylo nutné předávat metodě více parametrů, byl pro ně vytvořen speciální objekt (návrhový vzor Messenger).

Kapitola 8

Závěr

Tato práce se věnovala problematice extrakce informací z WWW. V první části jsou popsány používané technologie v prostředí WWW a různé přístupy pro extrakci informací. Tato část slouží především k seznámení s danou problematikou.

Výsledkem této práce je aplikace sloužící pro extrakci informací z HTML dokumentů a jejich uložení do XML souboru a databáze. Jedná se o přenositelnou aplikaci implementovanou v jazyce Java. Proces vývoje, od analýzy po implementaci, je popsán v druhé části této technické zprávy. Jednoznačnou výhodou této aplikace je její pružnost. Díky modularitě umožňuje použití různých extrakčních metod a různých metod pro uložení získaných dat, což zaručuje použitelnost pro mnoho typů úloh. Jádro aplikace tvoří knihovna, která se stará o načítání vstupních dokumentů, dynamické načítání modulů, dávkové zpracování a několik dalších funkcí. Tuto knihovnu lze snadno využít pro vytváření dalších aplikací.

Tuto aplikaci lze využít pro práci s nejrůznějšími typy wrapperů. Ty mohou například shromažďovat informace z mnoha webových stránek, což se může hodit při hledání nejnižších cen v elektronických obchodech, zobrazování aktuálních kurzovních lístků vybraných bank na jednom místě apod. Aplikace umožňuje dávkové zpracování úloh zadaných prostřednictvím XML souboru, takže provedení mnoha složitých operací lze zajistit spuštěním pouze jednoho souboru. Aplikaci lze dále využít při vytváření wrapperů a při experimentování s nimi. Tím, že umožňuje jejich znovunačítání za běhu, může být upravený wrapper okamžitě otestován.

Při vytváření wrapperů vyšly najevo jejich hlavní nevýhody. Vývoj wrapperu je velmi pracný a pro každou webovou stránku s odlišnou strukturou musí být vytvořen speciální. Pokud se struktura stránky změní, musí se upravit i vytvořený wrapper.

Z těchto důvodů by bylo vhodné aplikaci rozšířit o součást, která by umožňovala automatickou tvorbu wrapperů některým ze způsobů uvedených v části 4.1.2. Dalším možným rozšířením je propracování grafického uživatelského rozhraní a vytvoření sofistikovaných modulů pro automatickou tvorbu XML dokumentů. Zvýšení výkonnosti aplikace by bylo možné při paralelním provádění některých operací ve více vláknech.

Literatura

[1] Ashish, N.; Knoblock, C.: Wrapper Generation for Semi-structured Internet Sources. In Workshop on Management of Semistructured Data. Tucson, Arizona, 1997.

[2] Baumgartner, R.; Flesca, S.; Gottlob, G.: Visual Web Information Extraction with Lixto. [pdf], 2001.
URL http://www.dia.uniroma3.it/~vldbproc/016_119.pdf

[3] Burget, R.: Information Extraction from HTML Documents Based on Logical Document Structure. [pdf], 2001.

[4] Buttler, D.; Liu, L.; Pu, C.: A Fully Automated Object Extraction System for the World Wide Web. In Proc. of IEEE International Conference on Distributed Computing Systems, 2001.

[5] Bártík, V.: Dolování z textu a na webu. [pdf], 2008.
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ZZN-IT/lectures/09_TextWebMining.pdf

[6] Ceri, S.; Comai, S.; Damiani, E.; aj.: XML-GL: a graphical query language for querying and restructuring XML documents. WWW Conference, 1999.

[7] Chamberlin, D.: XQuery: A query language for XML. 2001.
URL www.w3.org

[8] Crescenzi, V.; Mecca, G.; Merialdo, F.: RoadRunner: Towards automatic data extraction from large web sites. Technical Report n. RT-DIA-64-2001, D.I.A. Universita di Roma Tre, 2001.

[9] Firat, A.; Madnick, S.; Yahaya, A.; aj.: Information Aggregation using Cameleon Web Wrapper. 6th ICECWT, 2005.

[10] Florescu, D.; Deutsch, A.; Levy, A.; aj.: A query language for XML. WWW Conference, 1999.

[11] Freitag, D.: Using Grammatical Inference to Improve Precision in Information Extraction. In ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition, 1997.

[12] Freitag, D.; McCallum, A.: Information Extraction with HMMs and Shrinkage. In Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction, 1999.

[13] Gold, E.: Language Identification in the Limit. Information and Control, 10(5):447-474, 1967.

[14] Hong, T.; Clark, K.: Using Grammatical Inference to Automate Information Exraction from the Web. In Principles of Data Mining and Knowledge Discovery, 2001.

[15] Hsu, C.; Dung, M.: Generating Finite State Transducers for Semi-Structured Data Extraction form the Web. Information systéme 23, 1998.

[16] IANA: IANA — MIME Media Types. [online], [cit. 2009-05-10].
URL http://www.iana.org/assignments/media-types/

[17] ICU User Guide: Character Set Detection. [online], Poslední modifikace: 2008 [cit. 2009-05-10].
URL http://userguide.icu-project.org/conversion/detection

[18] java.sun.com: Understanding Extension Class Loading. [online], [cit. 2009-05-10].
URL http://java.sun.com/docs/books/tutorial/ext/basics/load.html

[19] Kosala, R.; Van den Bussche, J.; Bruynooghe, M.; aj.: Information Ex-traction in Structured Documents using Tree Automata Induction. In Principles of Data Mining and Knowledge Discovery, Proceedings of the 6th International Conference (PKDD-2002), 2002.

[20] Kushmerick, N.; Weld, D.; Doorenbos, R.: Wrapper Induction for Information Extraction. In International Joint Conference on Artificial Intelligence, 1997.

[21] Matcha, K.: A Look At The Java Class Loader. [online], [cit. 2009-05-10].
URL http://www.javalobby.org/java/forums/t18345.html

[22] Mcmanis, C.: The basics of Java class loaders. [online], [cit. 2009-05-10].
URL http://www.javaworld.com/jw-10-1996/jw-10-indepth.html?page=1

[23] Muslea, I.; Minton, S.; Knoblock, C.: Hierarchical wrapper induction for semistructured information sources. AAMA 4, 2001.

[24] Sahuguet, A.; Azavant, F.: Building Intelligent Web Applications using Lightweighted wrappers. Data and Knowledge Eng. 36, 2001.

[25] Toman, M.: Srovnání přístupů extrakce užitečné informace z webu. [pdf], 2008.
URL http://znalosti2008.fiit.stuba.sk/download/articles/znalosti2008-Toman.pdf

[26] Vítek, M.: Dolování z textu. [pdf].
URL
http://www.fit.vutbr.cz/study/courses/ZZD/public/seminar0405/vitek.pdf

[27] W3C: The global structure of an HTML document. [online], [cit. 2009-05-10].
URL http://www.w3.org/TR/html4/struct/global.html#edef-META

[28] W3C: HTML Document Representation. [online], [cit. 2009-05-10].
URL http://www.w3.org/TR/html4/charset.html#encodings

[29] W3C: XHTML 1.0 - DTDs. [online], [cit. 2009-05-10].
URL
http://www.w3.org/TR/xhtml1/dtds.html#dtdentry_xhtml1-strict.dtd_meta

[30] Wikipedie - otevřená encyklopedie: Extensible HyperText Markup Language. [online], Poslední modifikace: 04. 11. 2008. [cit. 2008-12-28].
URL http://cs.wikipedia.org/wiki/Extensible_HyperText_Markup_Language

[31] Wikipedie - otevřená encyklopedie: Hypertext Transfer Protocol. [online], Poslední modifikace: 06. 11. 2008. [cit. 2008-12-28].
URL http://cs.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[32] Wikipedie - otevřená encyklopedie: Sémantický web. [online], Poslední modifikace: 27. 10. 2008. [cit. 2008-12-28].
URL http://cs.wikipedia.org/wiki/S%C3%A9mantick%C3%BD_web

[33] Wikipedie - otevřená encyklopedie: HyperText Markup Language. [online], Poslední modifikace: 28. 12. 2008. [cit. 2008-12-28].
URL http://cs.wikipedia.org/wiki/HyperText_Markup_Language

[34] Wikipedie - otevřená encyklopedie: World Wide Web. [online], Poslední modifikace: 28. 12. 2008. [cit. 2008-12-28].
URL http://cs.wikipedia.org/wiki/World_Wide_Web

[35] Zendulka, J.; Bártík, V.; Lukáš, R.; aj.: Získávání znalostí z databází - studijní opora. [pdf], 2006.
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/ZZN-IT/texts/ZZN.pdf

Seznam příloh

Příloha A

Manuál

A.1 Uživatelská příručka

A.1.1 Spuštění aplikace

Aplikaci je možné spustit z příkazového řádku následujícím způsobem:

java -jar FiewGUI.jar

Aplikace využívá tříd z archivů *fiew.jar*, *chardet.jar* a *mysql-connector-java-5.1.6-bin.jar*. Cesty k těmto archivům musí být tedy obsaženy v systémové class path. Pokud budou všechny tyto archivy obsaženy v adresáři *lib*, který se bude nacházet na stejné úrovni jako *FiewGUI.jar*, je možné aplikaci spustit příkazem uvedeným výše. Pokud budou knihovny umístěny jinde, je nutné k nim nastavit cestu pomocí přepínače *-cp* nebo *-classpath*, kde se jednotlivé cesty se oddělují středníkem.

A.1.2 Načítání dokumentů a modulů

Dokumenty z WWW jsou načítány na základě zadané URL. Ta se zadává prostřednictvím dialogu, který je dostupný v menu `File->Open URL`. Dokumenty z lokálního disku se načítají prostřednictvím dialogu v menu `File->Open Files`, který umožňuje načtení více dokumentů zároveň.

Moduly pro extrakci informací se načítají prostřednictvím dialogu v menu `Wrapper->Load`, kam se zadá úplný název požadované třídy včetně balíčků a jmenných prostorů bez koncovky *.class*. Stejným způsobem probíhá načítání modulů pro uložení získaných dat prostřednictvím dialogů v menu `Output->XML->Load Class` a `Output->Database->Load Class`. Zadané třídy jsou nejprve hledány v systémové class path, pokud nejsou nalezeny tam, tak se prohledává programová class path. Nastavení programové class path je popsáno v následující části. Podrobný popis, jakým pracuje class loader, je uveden v sekci 5.2.2. Defaultní moduly pro uložení nalezených dat jsou načítány prostřednictvím dialogů v menu `Output->XML->Load Default XML Creator` a `Output->Database->Load Default Database Writer`. Způsob,

jakým tyto defaultní moduly pracují, je uveden v části 6.4.6. Po úspěšném načtení je každý dokument nebo modul zobrazen v odpovídající části stromové struktury.

A.1.3 Programová class path

Programová class path obsahuje cesty k adresářům, ve kterých se budou hledat načítané třídy, pokud nebudou nejprve nalezeny v systémové class path. Programovou class path je možné zobrazit v menu `Classpath->Show Program Classpath`. Editovat jí je možné přímo v tomto dialogu. Jednotlivé cesty musí být odděleny systémově závislým oddělovačem. Operační systém windows používá středník, unixové systémy používají dvojtečku. Vyhledávání tříd probíhá v adresářích v pořadí, v jakém jsou uvedeny v programové class path. Po nalezení požadované třídy prohledávání končí. Pro snadné přidávání cest do programové class path slouží dialog v menu `Classpath->Add to Program Classpath`.

A.1.4 Spuštění wrapperu

Pro spuštění wrapperu slouží průvodce. V prvním kroku se vyberou ze všech načtených dokumentů ty, ve kterých se bude vyhledávat. V druhém kroku se vybere wrapper, který se má spustit. V dalších krocích jsou vybrány moduly, které se postarají o uložení nalezených dat. Tento průvodce je v menu `Wrapper->Start` a `Run->Wrapper`.

A.1.5 Dávkové zpracování

Dávkový XML soubor je spuštěn prostřednictvím dialogu v menu `Run->XML File`. Popis tohoto souboru je uveden v části 6.5. Pro lepší pochopení jsou v následující kapitole uvedeny ukázky několika těchto souborů.

A.2 Použití knihovny fiew

Knihovnu *fiew* je možné snadno využít pro vytvoření své vlastní aplikace. Ve vývojovém prostředí NetBeans lze snadno přidat archiv *fiew.jar* jako knihovnu. Poté už jen stačí importovat potřebné třídy. Spolu s tímto archivem je nutné přidat archivy *chardet.jar* a *mysql-connector-java-5.1.6-bin.jar*, které jsou využívané knihovnou *fiew*. Potřebná programová dokumentace této knihovny je na přiloženém CD.

Příloha B

Ukázka použití systému

B.1 Extrakce nejlépe hodnocených filmů

První ukázkou je použití wrapperu pro získání dat z tabulky obsahující 200 nejlépe hodnocených filmů na serveru csfd.cz a jejich uložení do XML souboru a do databáze MySQL. Ukázka také obsahuje XML soubor pro dávkové zpracování této úlohy.

Pro tento úkol postačí vytvořit jednoduchý wrapper typu HLRT, který bude získávat data pomocí regulárních výrazů. Protože se jedná o extrahování dat z tabulky s jednoduchou strukturou, je možné pro uložení získaných dat použít defaultní moduly.

Prvním krokemje vytvoření wrapperu, který implementuje rozhraní *WrapperInterface*. Metoda *start* bude obsahovat kód pro extrakci informace a naplnění výstupního vektoru, což je objekt typu *java.util.Vector*. Tato metoda pracuje se vstupním dokumentem jako z objektem typu *java.lang.String*. Nejprve pomocí vhodných řetězců označí začátek a konec tabulky, ze které se budou data získávat, aby mohl být ostatní text přeskočen. Poté je potřeba vytvořit regulární výraz, který bude hledat požadovaná data v tabulce. Sloupce tabulky, které chceme extrahovat obsahují název filmu, rok uvedení, hodnocení uživatelů a počet hodnotících. Nalezenými daty je nutné naplnit výstupní vektor, což zajistí následující kód:

```
LinkedHashMap <String, String> m;
vec = new Vector();
int i=0;
while(matcher.find()){
  i++;
  m = new LinkedHashMap<String, String>();
  m.put("nazev", matcher.group(1));
  m.put("rok",matcher.group(2).replace('(',' ').replace(')',' ').trim());
  m.put("hodnoceni", matcher.group(3));
  m.put("pocet", matcher.group(4).replaceAll("[^0-9]", ""));
  vec.add(m);
```

```
}
```

Aby bylo možné použít defaultní moduly pro uložení nalezených dat, musí vektor odpovídat struktuře popsané v části 6.4.6. Výstupní vektor tedy musí být naplněn objekty typu *java.util.LinkedHashMap*. Každý takovýto objekt představuje jeden řádek tabulky a mapuje názvy sloupců na jejich hodnoty.

Dalším krokem je vytvoření XML souboru pro dávkové zpracování. Ten může vypadat následujícím způsobem:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wrappers SYSTEM "D:\Skola\Diplomka\program\DTD\schema.dtd">
<wrappers>
  <wrapper classpath="D:\Skola\Diplomka\program\MyWrappers\build\classes\"
  classname="myWrappers.Nej200FilmuNaCSFDWr">
    <inputs>
      <file path="d:\ČSFD  NEJLEPŠÍCH 200 FILMŮ.htm"/>
    </inputs>
    <outputs>
      <defaultXMLcreator filepath="d:\200NejFilmu.xml"
      rootElementName="filmy" childElementName="film"/>
      <defaultDatabaseWriter hostName="localhost" databaseName="test"
      tableName="filmy" userName="test" password="test"/>
      </outputs>
  </wrapper>
</wrappers>
```

Po spuštění tohoto souboru je načten wrapper, poté HTML dokument, umístěný na lokálním disku, ze kterého se budou informace extrahovat. Wrapper je spuštěn a je získán výstupní vektor. Ten se předá defaultním modulům pro vytvoření XML souboru a pro uložení dat do databáze. Vytvořený soubor *200NejFilmu.xml* vypadá následujícím způsobem:

```
<?xml version="1.0" encoding="utf-8"?>
<filmy>
    <film>
        <nazev>Vykoupení z˜věznice Shawshank</nazev>
        <rok>1994</rok>
        <hodnoceni>95.4%</hodnoceni>
        <pocet>17469</pocet>
    </film>
    <film>
        <nazev>Forrest Gump</nazev>
        <rok>1994</rok>
```

```
        <hodnoceni>95.4%</hodnoceni>
        <pocet>20105</pocet>
    </film>
    <film>
        <nazev>Přelet nad kukaččím hnízdem</nazev>
        <rok>1975</rok>
        <hodnoceni>94.7%</hodnoceni>
        <pocet>13653</pocet>
    </film>
    .
    .
```

Modul pro uložení dat předpokládá, že je v databázi vytvořena tabulku s názvem *filmy*, kde názvy sloupců odpovídají použitým klíčům v objektu *java.util.LinkedHashMap* a jejich typ umožňuje ukládat textový řetězec.

B.2 Extrakce kurzovních lístků

V této ukázce získává wrapper aktuální kurzovní lístky různých bank. V tomto případě už mají získávaná data složitější strukturu, takže je nutné implementovat vlastní modul pro uložení těchto získaných dat.

Prvním krokem je opět implementace wrapperu. Ten bude pracovat podobným způsobem jako v předešlém případě s tím rozdílem, že bude pracovat s více různými vstupními dokumenty. To znamená, že pro každý dokument bude muset být vytvořen regulární výraz. Oproti předešlému příkladu bude muset být navíc udržována informace o kterou banku se jedná. Tuto informaci je možné předat zároveň s daty ve výstupním vektoru. Z tohoto důvodu je nutné implementovat vlastní modul pro uložení dat, který bude umět pracovat s daným výstupním vektorem.

Dávkový soubor XML může vypadat následujícím způsobem:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wrappers SYSTEM "D:\Skola\Diplomka\program\DTD\schema.dtd">
<wrappers>
  <wrapper classpath="D:\Skola\Diplomka\program\MyWrappers\build\classes\"
  classname="myWrappers.KurzovniListkyWr">
    <inputs>
      <page url ="http://www.kb.cz/NASA/rateList/SRVRate" />
      <page url ="http://www.csas.cz/banka/appmanager/portal/banka?_nfpb=
      true&amp;_pageLabel=exchangerates" />
      <page url ="http://www.unicreditbank.cz/cz/kurzovni-listek-men.html"/>
    </inputs>
```

```
    <outputs>
      <XMLcreator classpath="D:\Skola\Diplomka\program\MyWrappers\
      build\classes\" classname="myXMLCreators.KurzovniListky"/>
    </outputs>
  </wrapper>
</wrappers>
```

Vytvořený wrapper získává aktuální kurzy z dokumentů s uvedeným URL. Výsledný XML soubor *kurzy.xml* vytvořený třídou *myXMLCreators.KurzovniListky* vypadá následujícím způsobem:

```
<?xml version="1.0" encoding="utf-8"?>
<banky>
    <ČeskáSpořitelna>
        <EUR>
            <Změna>0,45</Změna>
            <DevizyNákup>26,239</DevizyNákup>
            <DevizyProdej>27,201</DevizyProdej>
        </EUR>
        <USD>
            <Změna>0,77</Změna>
            <DevizyNákup>19,331</DevizyNákup>
            <DevizyProdej>20,040</DevizyProdej>
        </USD>
    </ČeskáSpořitelna>
    <UnicreditBank>
        <EUR>
            <Změna>Nezadáno</Změna>
            <DevizyNákup>26.232750</DevizyNákup>
            <DevizyProdej>27.167250</DevizyProdej>
        </EUR>
        <USD>
            <Změna>Nezadáno</Změna>
            <DevizyNákup>19.324793</DevizyNákup>
            <DevizyProdej>20.013208</DevizyProdej>
        </USD>
    </UnicreditBank>
    <KomerečníBanka>
        <EUR>
            <Změna>Nezadáno</Změna>
            <DevizyNákup>26,0410</DevizyNákup>
```

```
            <DevizyProdej>27,1590</DevizyProdej>
        </EUR>
        <USD>
            <Změna>Nezadáno</Změna>
            <DevizyNákup>18,8470</DevizyNákup>
            <DevizyProdej>19,7550</DevizyProdej>
        </USD>
    </KomerečníBanka>
</banky>
```