VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

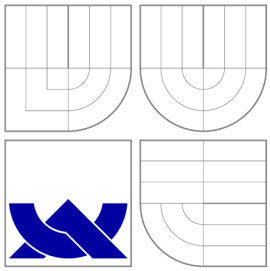
EXTRAKCE DAT Z DYNAMICKÝCH WWW STRÁNEK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

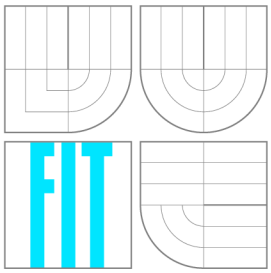
AUTOR PRÁCE
AUTHOR

Bc. PETR PUNA

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

EXTRAKCE DAT Z DYNAMICKÝCH WWW STRÁNEK

DATA EXTRACTION FROM DYNAMIC WWW PAGES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR PUNA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK BURGET, Ph.D.

BRNO 2009

Abstrakt

Tato práce obsahuje stručný přehled technologií používaných pro prezentaci a získání dat na WWW a popisuje vybrané nástroje pro extrakci dat z webových stránek. Práce dále navrhuje nový nástroj pro získání stránek generovaných na základě vyplnění webových formulářů který umožňuje uživateli definovat data na takovýchto webových stránkách a dokáže takto definovaná data extrahovat a nabídnout ve formátu XML použitelném pro další strojové zpracování.

Klíčová slova

Extrakce dat z WWW WWW HTML XML HTTP XHTML (X)HTML formuláře Lixto Java JSP HTML Parser JavaScript

Abstract

This work contains a brief overview of technologies for representation and obtaining data on WWW and describes selected web data extraction tools. The work designs a new tool for obtaining pages generated by filling in web forms which allows its user to define data on such web pages and which can extract those data and offer it in a XML format suitable for future machine processing.

Keywords

WWW data extraction WWW HTML XML HTTP XHTML (X)HTML forms Lixto Java JSP HTML Parser JavaScript

Citace

Petr Puna: Extrakce dat z dynamických WWW stránek, diplomová práce, Brno, FIT VUT v Brně, 2009

Extrakce dat z dynamických WWW stránek

Prohlášení

Prohlašuji, že jsem tuto technickou zprávu vypracoval samostatně pod vedením Ing. Radka Burgeta Ph. D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Puna
27. ledna 2009

Obsah

Kapitola 1

Úvod

Na internetu je dnes k dispozici velké množství informací a jejich prohlížení je velmi zdlouhavé. Proto by bylo vhodné nahradit ruční prohlížení webových stránek strojovým zpracováním.

Rozhodneme-li se data z webových stránek dále vyhodnocovat a zpracovávat, pak nám současná podoba webových stránek příliš vyhovovat nebude – užitečná data jsou zabalena mezi prostředky pro jejich prezentaci a není možné je snadno strojově najít.

Můj projekt si proto klade za cíl vytvořit aplikaci, která by umožnila uživateli definovat na webových stránkách data, která má zájem zpracovávat, tato data následně vyhledávat a získat jejich nové hodnoty. Aplikace také umožní získat tato data ve formátu, který je vhodný pro další strojové zpracování.

Tato práce nabízí nejprve přehled technologií používaných k přístupu k datům na webových stránkách (2.1) a k jejich prezentaci (2.2). V semestrálním projektu, který této práci předcházel, jsem také popsal dva existující nástroje pro extrakci dat z webových stránek – prvním je moje bakalářská práce, na kterou tento projekt navazuje (3.1) a druhým je komerční systém pro extrakci dat Lixto (3.2).

Na základě poznatků z existujících nástrojů a úvah načrtnutých v semestrálním projektu pak v další kapitole upřesňuji cíl práce, podrobněji popisuji její součásti a představuji konkrétní řešení (4).

V kapitole, která následuje (5) se dále věnuji vybraným problémům více do hloubky.

Kapitola 2

Protokoly a jazyky na WWW

Moje práce zpracovává webové stránky. Proto je vhodné stručně popsat technologie, které se v této oblasti používají.

2.1 HTTP

Protokol HTTP (*Hypertext Transfer Protocol*) v aktuální verzi 1.1 byl navržen organizacemi W3C (*World Wide Web Consortium*) a IETF (*Internet Engineering Task Force*) a publikován jako RFC dokument v roce 1999 [3].

HTTP je protokol aplikační vrstvy, který se používá pro komunikaci mezi klientem a WWW serverem pro přenos (nejen) webových stránek. Klient posílá požadavek a server odpověď. HTTP je nejčastěji implementován nad protokolem TCP, ale není to podmínkou. Je však nutné, aby protokoly pod HTTP zaručovaly spolehlivý přenos. Implicitní port pro komunikaci HTTP přes TCP protokol je port 80.

Protokol HTTP je bezestavový. To znamená, že každý požadavek je zpracován nezávisle na předchozích požadavcích klienta. Výhodou je, že server nemusí udržovat informace o klientech. V mnoha situacích je však tato vlastnost nežádoucí a proto se často používají různá řešení pro přenos stavové informace (parametr v dotazové části URI, skrytá pole ve formuláři, cookies)

Od verze HTTP/1.1 umožňuje protokol navazovat perzistentní spojení. Odpadá tak nutnost vytvářet pro každý požadavek nové spojení, což snižuje výsledné zpoždění.

2.1.1 URI

Striktně podle definice [15] je URI (*Uniform Resource Identifier*) obecnější pojem, než URL (*Uniform Resource Locator*) a URN (*Uniform Resource Name*). URL jsou URI, které kromě identifikace zdroje obsahují také informace o jeho umístění. URN naproti tomu poskytuje unikátní název, který není závislý na aktuálním umístění.

V běžné řeči se však ujal termín URL ve významu, který odpovídá URI. I tato práce bude v dalších kapitolách používat termín URL, protože pro můj projekt bude vždy podstatné zejména umístění konkrétního zdroje.

Formát URI

Úplný popis syntaxe URI lze najít v [15]. Zjednodušeně lze URI popsat takto:

```
<schéma>:<cesta>[?<dotaz>]
```

Schéma typicky označuje způsob přístupu k danému zdroji – tedy protokol.

Cesta může být absolutní nebo relativní (v obou případech musí začínat znakem "/"), případně může začínat adresou serveru – pak musí být před adresou znaky "//".

Adresa serveru může být zapsána kanonickým jménem (např. *fit.vutbr.cz*), nebo IP adresou. Za adresou může být uvedeno číslo portu, oddělené dvojtečkou.

Dotaz za znakem "?" tvoří data, která mohou být použita k další identifikaci zdroje. Ve webových aplikacích obvykle obsahuje parametry předávané serveru, stavové informace, případně data z formulářů.

Kódování znaků v URI

URI může ve všech částech obsahovat znaky anglické abecedy a další znaky, jako jsou "–", ".", "_". Některé další znaky však slouží jako oddělovače a nelze je v některých částech URI použít mimo jejich obvyklý význam. Patří sem například ":", "/", "?". Dále se v URI nesmí vyskytovat mezera. Konkrétní seznam rezervovaných znaků pro jednotlivé části URI naleznete opět v [15].

Pokud je třeba vložit do URI některý z rezervovaných znaků, je nutné jej zakódovat. K tomu se používá následující formát:

`%HH`

Kde `H` je číslice v hexadecimální soustavě. Toto číslo pak znamená kód znaku v ASCII tabulce. Mezera se tedy zakóduje jako `%20`, znak "/" jako `%2F` atd.

2.1.2 Požadavek klienta

Požadavek klienta (HTTP request) má následující formát:

```
<metoda> <URI> <verze HTTP>
<prázdný řádek>
```

Volitelně může následovat tělo zprávy.

Příklad požadavku

```
GET http://wis.fit.vutbr.cz/FIT/st/study-v.php HTTP/1.1
```

Metody

Metoda je vždy uvedena na začátku požadavku, protože určuje jeho význam. Největší význam pro účely této práce mají metody GET a POST.

- **Metoda GET**

  Metoda GET se používá pro získání zdroje na zadaném URI. Protože součástí URI můžou být i parametry (za znakem "?"), je možné takto odesílat serveru data. Typickým použitím je předávání stavové informace v takových parametrech – je totiž možné vložit při generování stránky na serveru data jako parametry do odkazů na stránce.

  Metodu GET je také možné použít při odesílání dat formulářem.

- **Metoda POST**

  Metoda POST umožňuje vložit do požadavku data (v těle zprávy), která má server zpracovat. Typickým použitím je odesílání formuláře.

2.1.3 Odpověď serveru

Na požadavek klienta server odpovídá zprávou (HTTP response) v tomto formátu:

```
<verze HTTP> <stavový kód> <zdůvodnění>
<volitelné hlavičky>
<prázdný řádek>
```

Volitelně může následovat tělo zprávy.

Stavové kódy

Každá odpověď serveru musí obsahovat stavový kód. Stavový kód je třímístné číslo. Ke každému kódu je dále přiřazeno zdůvodnění, což je krátký text, popisující případ, kdy je daný kód použit. všechny stavové kódy jsou definovány v [15]. HTTP protokol definuje těchto pět skupin stavových kódů (uvádím dále nejpoužívanější kódy a jejich zdůvodnění):

- **1xx: Informational**

  2xx: Success

  200: OK

- **3xx: Redirection**

  301: Moved Permanently

  307: Temporary Redirect

- **4xx: Client Error**

  400: Bad Request

  403: Forbidden

  404: Not Found

  405: Method Not Allowed

  408: Request Time-out

- **5xx: Server Error**

  500: Internal Server Error

  501: Not Implemented

  503: Service Unavailable

  505: HTTP Version not supported

Hlavičky

Server může vrátit několik typů hlaviček.

Location Hlavička obsahuje URI pro přesměrování u zpráv se stavovým kódem 3xx.

Content-Encoding Určuje kódování přenášené zprávy a tedy i mechanismus dekódování.

Content-Type Tato hlavička určuje typ obsahu zprávy. Použít lze registrované MIME typy. Tyto typy registruje organizace IANA (*The Internet Corporation for Assigned Names and Numbers*) a na jejich stránkách je možné najít i seznam registrovaných typů ([4]). Mezi typy používané na WWW patří zejména:

text/plain

text/html

text/xml

text/css

application/xhtml+xml

image/jpeg

video/mpeg

application/octet-stream – spustitelné soubory, implicitní typ (pokud není typ rozpoznán, použije se tento)

Verze HTTP

0.9 První verze protokolu. V současnosti je již zastaralá. Používala pouze metodu GET.

HTTP/1.0 Zveřejněna v roce 1996. Přidává další metody. Zavádí povinné uvádění verze protokolu. Pro svou jednoduchost je stále používána, zejména proxy servery.

HTTP/1.1 Aktuální verze, která byla zveřejněna v roce 1999. Zavádí optimalizace spojení, jako jsou perzistentní spojení a pipelining (klient může serveru poslat více požadavků, aniž by mezi nimi obdržel odpověď; poté co server požadavky zpracuje, odešle zpět odpovědi).

HTTP/1.2 Tato verze měla obsahovat další rozšíření. Od jejího vydání se ale později upustilo a její část byla v roce 2000 vydána jako experimentální RFC dokument.

2.1.4 Cookies

Cookies jsou relativně krátká data (řádově kilobajty), která může odeslat www server klientovi. S cookie se vždy váže i adresa serveru, který ji zaslal a doba expirace. Klient si následně cookie uloží (pokud není nastaven, aby cookies nepřijímal) a při každém požadavku na tento server odešle s požadavkem také všechny cookies, jejichž doba expirace ještě neuplynula.

Účel cookies spočívá v umožnění předání stavové informace.

2.1.5 HTTPS

HTTPS (*secure HTTP*) označuje použití HTTP protokolu nad připojením SSL nebo TLS protokolem. HTTPS používá URI schéma `https:` a implicitní port 443. SSL (TLS) vytváří šifrované spojení mezi dvěma komunikujícími stranami – zde mezi klientem a serverem. Používají se zde principy asymetrické kryptografie, kdy klient získá veřejný klíč serveru z jeho volně přístupného certifikátu.

HTTPS zajišťuje ochranu proti odposlechnutí přenášených informací. Je tedy vhodné jej použít, je-li třeba přenášet citlivé údaje jako jsou čísla kreditních karet, rodná čísla apod.

2.2 Značkovací jazyky založené na SGML

Ve své práci potřebuji analyzovat zdrojový kód webových stránek. Proto je třeba podrobněji prozkoumat jazyky, které se při psaní webových stránek používají.

2.3 SGML

SGML (*Standard Generalized Markup Language*) je značkovací jazyk, standardizovaný organizací ISO (*ISO 8879:1986 Information processing-Text and office systems-Standard Generalized Markup Language (SGML)*), který vznikl v šedesátých letech minulého století za účelem sdílení strojově zpracovatelných dokumentů (viz [13]).

Standard SGML je velmi komplexní a proto je náročnější jeho podporu implementovat. Z toho důvodu se v současnosti používají zejména jazyky z SGML odvozené, které podporují jen některé z vlastností SGML, jsou však díky tomu jednodušší a snadněji se implementují. Patří mezi ně mj. jazyky HTML, XML a z něj odvozený jazyk XHTML.

2.4 HTML

HTML (*HyperText Markup Language*) je na SGML založený jazyk, navržený pro tvorbu WWW stránek se schopností vytvářet hypertextové vazby (odkazy). HTML je mezinárodním standardem (*ISO/IEC 15445:2000*). Specifikace jazyka spravuje organizace *World Wide Web Consortium* (W3C). Aktuální verze je HTML 4.01.

Tělo HTML dokumentu se skládá z tagů uzavřených do úhlových závorek. Rozlišujeme tagy otevírací (např. `<h1>`) a zavírací (např. `</h1>`), mezi kterými mohou být další tagy, nebo prostý text. Otevírací tagy mohou mít atributy, zapsané ve tvaru:

`<p align=center color=red>`.

U tagu, který neobsahuje žádné vnořené prvky je také možné použít zkráceného zápisu: `<br />`. HTML dokument může obsahovat i komentáře uzavřené do značek `<!--` a `-->`.

Protože tagy mohou být do sebe vnořeny, vytváří se tím stromová struktura dokumentu.

Specifikace definuje sémantiku jednotlivých tagů i jejich atributů, nedefinuje však přesnou podobu stránky ve webovém prohlížeči.

Více informací o HTML je možné najít na [10], případně v příslušné specifikaci [18].

2.5 XML

Jazyk XML (*Extensible Markup Language*) Je značkovací jazyk, který podobně jako HTML vznikl zjednodušením normy SGML (pořád ale platí, že XML dokument je zároveň SGML

dokumentem). XML klade na dokumenty přísnější požadavky. Mezi nová pravidla (proti HTML) patří například:

Všechny tagy musí být uzavřeny V HTML bylo možné ponechat některé tagy neuzavřené. XML toto zakazuje, což usnadňuje zpracování dokumentů. Každý tag je nutné uzavřít zavíracím tagem, nebo použít zkrácené formy `<tag />`.

Tagy se nesmí překrývat Tag, otevřený uvnitř nadřazeného tagu, musí být v tomto nadřazeném tagu i uzavřen. Není možné jej uzavřít až později.
Příklad (špatně): `<strong> <i>text</strong> </i>`.
Příklad (správně): `<strong> <i>text</i> </strong>`.

Atributy musí být uzavřeny v uvozovkách Na rozdíl od HTML, všechny hodnoty atributů musí být uzavřeny v uvozovkách.

XML nedefinuje konkrétní dovolené tagy ani jejich význam. To umožnilo vznik různých specializovaných jazyků založených na XML, např. XHTML.

Více o XML je k nalezení v [14] a v příslušném dokumentu W3C ([17]).

2.6 XHTML

Jazyk XHTML (*Extensible HyperText Markup Language*) je postaven na XML. Jedná se vlastně o XML doplněné o definici tagů, které se vyskytují v jazyce HTML. Tím vzniká jazyk použitelný jako náhrada HTML, který se ovšem řídí přísnějšími pravidly XML (platí pro něj omezení uvedená v části o XML).

V současné době je organizací W3C doporučovaným jazykem pro tvorbu webových stránek aktuální verze XHTML 1.1 (viz [19]).

2.7 Formuláře

Webové formuláře představují způsob, jakým může uživatel webového klienta odeslat data serveru.

2.7.1 Pole v (X)HTML formulářích

Ve formulářích je možné použít různé druhy vstupních polí, které usnadňují vložení určitých typů dat.

- **text**

  Jednoduché textové políčko pro zadání krátkých textů (jeden řádek).

- **textarea**

  Slouží k vložení delších textů (může mít více řádků).

- **checkbox**

  Zaškrtávací tlačítko – slouží k vložení pravdivostní hodnoty.

- **radio**

  Stejně jako chceckbox má dvě polohy a označuje pravdivostní hodnotu (vybráno / nevybráno). Je však možné definovat skupinu těchto prvků, ve které bude vybrán nejvýše jeden z prvků.

- **file**

  Používá se pro nahrání souboru na server.

- **select**

  Umožňuje výběr jedné, nebo několika položek z předem daného seznamu.

- **hidden**

  Zvláštní pole, které zůstává uživateli skryto, ale při odeslání formuláře se jeho hodnota odešle také – toto je užitečné například pro předávání stavové informace na dynamicky generovaných stránkách.

- **reset**

  Tlačítko, které obnoví původní stav formuláře.

- **submit**

  Tlačítko pro odeslání formuláře.

2.7.2 Metody POST/GET

Formulář je možné odeslat na server jednou ze dvou HTTP metod:

GET

Při odeslání dat metodou GET se vytvoří URI HTTP požadavku takto:

`<adresa>?<pole1>=<hodnota1>&<pole2>=<hodnota2>&...&<poleN>&<hodnotaN>`

Kde `adresa` je hodnotou atributu `action` příslušného formuláře, `pole1` až `poleN` jsou názvy formulářových polí (atribut `name`) a `hodnota1` až `hodnotaN` jsou hodnoty těchto polí.

Tato metoda je vhodná jen pro menší množství odesílaných dat, protože se data odesílají v URI a některá zařízení (např. proxy servery) mohou délku URI omezovat. Metodu GET naopak musíme použít, pokud je třeba předávat formulářové hodnoty i v odkazech – příklad: `http://server/web?hledej=WiFi`.

POST

Pokud formulář používá HTTP metodu POST, pak se po jeho odeslání vytvoří HTTP požadavek s touto metodou, kde URI bude opět hodnota atributu `action` formuláře, ale data jsou vložena do těla HTTP požadavku.

Výhodou odesílání dat metodou POST mimo jiné je, že takto odesílaná data nejsou součástí URI a tak nezůstávají v historii prohlížeče.

2.8 DOM

DOM (*Document Object Model*) je objektový model reprezentace HTML a XML dokumentů vytvořený konsorciem W3C [16]. DOM reprezentuje dokument jako strom a nabízí tak snadnou a přehlednou navigaci v dokumentu. Specifikace DOM rozlišuje tři úrovně podpory DOM. Webové prohlížeče se nicméně v implementaci DOM liší.

2.9 JavaScript, JScript

JavaScript je skriptovací jazyk určený ke vkládání do webových stránek. Je však interpretován na klientské straně – ve webovém prohlížeči. Microsoft svou implementaci JavaScriptu v Internet Exploreru nazývá JScript. V tomto textu budu však používat název JavaScript bez ohledu na prohlížeč, ve kterém je spuštěn. JavaScript má syntaxi podobnou jazyku C a používá se nejčastěji pro tyto účely:

- Manipulace s okny prohlížeče (zobrazení, změna rozměrů, skrytí nástrojových lišt,...).
- Validování formulářů před jejich odesláním.
- Úprava grafické podoby stránky (např. různé efekty, menu).

Pro veškerou manipulaci se stránkou používá JavaScript DOM.

Vždy bychom však měli mít na paměti, že klient nemusí JavaScript podporovat, nebo může být jeho provádění zakázáno.

2.9.1 Kompatibilita

Snad největším problémem při programování v JavaScriptu je vzájemná nekompatibilita jeho implementací v prohlížečích. Jde zejména o odlišnosti v objektovém modelu dokumentu (DOM). Programátor je pak nucen psát skripty nebo jejich části několikrát pro různé prohlížeče.

Více se můžete o JavaScriptu dozvědět na [11].

Kapitola 3

Existující nástroje

3.1 Bakalářská práce – Osobní internetový vyhledávač

Ve své bakalářské práci ([9]) jsem implementoval jednoduchý nástroj, který umožňuje uživateli definovat data na konkrétních webových stránkách, tato data sledovat, ukládat jejich nové hodnoty, zobrazit historii jejich změn a vyhledávat v nich.

Hlavním zamýšleným použitím aplikace je sledování ceny zboží v internetových obchodech. Sledovaná proměnná je přesně určena stromem tagů od začátku stránky (bude podrobněji popsáno) a předpokládá se, že půjde o krátkou textovou nebo číselnou informaci (obsah jednoho tagu, nebo jeho část). Z toho plyne, že pokud chce uživatel sledovat více dat na jedné stránce, musí definovat odpovídající počet proměnných.

Projekt byl implementován v jazyce PHP5 – pracuje tedy jako webová aplikace.

V následujících kapitolách popíšu princip činosti aplikace, její možnosti a slabiny.

3.1.1 Popis použitého řešení

Použité řešení lze rozdělit na tři hlavní části:

- Nejprve je třeba definovat sledovaná data a tyto definice uložit ve vhodné formě tak, aby bylo možné pravidelně kontrolovat jak se data změnila.
- Dále je nutné zajistit samotnou kontrolu aktuální podoby dat. Tento proces by měl být co nejméně ovlivněn změnou ostatních částí stránky.
- Konečně poslední částí práce je vyhledávání ve sledovaných datech a vhodná reprezentace historie změn.

3.1.2 Definice sledovaných dat

Úprava zdrojového kódu stránky

Proces vytváření nové sledované proměnné by měl být uživatelsky příjemný a intuitivní. Také jsem chtěl, aby byla celá aplikace přístupná přes webové stránky, jako služba, kterou by bylo možné využívat více uživateli a bez nutnosti spouštět aplikaci na klientském počítači.

Uživatel má možnost vybrat sledovaná data na stránce pomocí jejich označení myší ve webovém prohlížeči. Původní stránku je proto třeba nejprve zpracovat a provést následující úpravy ve zdrojovém textu stránky:

Přidělení identifikátoru všem tagům na stránce

Každému tagu na stránce (mimo těch, u kterých to nemá smysl, např. `<head>`, `<br>`, aj.) je přidělen unikátní identifikátor prostřednictvím atributu `ID`. Současně s identifikátorem je prvkům nastavena událost myši (konkrétně `onMouseUp`) na připravený JavaScriptový kód.

Přidání kódu v jazyce JavaScript

Do stránky je vložen JavaScriptový kód – funkce, která přesměruje prohlížeč a tím předá jako parametry uživatelem označený text a identifikátor elementu, ve kterém se označený text nachází. Tento jednoduchý kód je jedinou součástí projektu, která se spouští na klientském počítači.

Změna relativních odkazů

Zpracovaná stránka je zobrazena serverem vyhledávače. Proto by se veškeré adresy prvků vložených na stránku pomocí relativní cesty staly neplatnými. Pro naše účely jde zejména o vložené obrázky, kaskádové styly a JavaScriptový kód, které by v tomto případě mohly způsobit výrazně odlišný vzhled zpracované stránky. Je tedy nutné takovéto relativně zadané cesty převést na absolutní (konkrétně jde o tagy `<img>`, `<a>` a `<link>`).

Časová náročnost

Takovéto zpracování zdrojového textu může být časově náročné. Zejména u dlouhých stránek proto hrozí překročení časového limitu pro běh PHP skriptu. Aplikace tento problém řeší průběžnou kontrolou doby běhu a před vypršením limitu zpracování stránky ukončí.

Vzhledem k zaměření aplikace není tento problém příliš závažný – předpokládá se totiž, že většina relevantních údajů se nachází na začátku stránky.

Uložení

Takto zpracovaný zdrojový kód je před odesláním klientovi uložen, kvůli pozdějšímu sestavení cesty stromem tagů a nalezení textu obklopujícího proměnnou – to umožňuje sledovat i část textu uvnitř vybraného tagu. Více viz část 3.1.2.

Výběr sledovaných dat

Po zobrazení takto ošetřené webové stránky stačí uživateli běžným způsobem (myší) označit text, který si přeje přidat mezi sledované proměnné. Ukončením výběru se vyvolá událost `onMouseUp` a tím také výše zmíněná JavaScriptová funkce.

Tento přístup má však jednu nevýhodu – sledovaná proměnná musí být celá v jednom tagu, protože v opačném případě by byla událost vyvolána prvkem, který neobsahuje celý vybraný text. Není tedy například možné sledovat hodnotu rozdělenou koncem odstavce, ani řezem písma.

Ani tato nevýhoda není zásadně omezující vzhledem k zaměření projektu na sledování krátkých textových nebo číselných hodnot.

Zpracování výběru

Když je aplikaci prostřednictvím JavaScriptu předán vybraný text a identifikátor tagu ze kterého pochází, je možné sestavit cestu stromem tagů od kořenového k vybranému. Tento strom je klíčový pro nalezení nové hodnoty proměnné při aktualizaci. Sám o sobě však nestačí – výběr totiž může být jen částí textu z tagu který událost vyvolal. Proto je nutné najít a uložit také text, který budoucí sledovanou proměnnou uvnitř tagu obklopuje. K vytvoření cesty stromem a nalezení zmíněných textů použijeme zdrojový kód stránky, zpracovaný a uložený dříve (viz 3.1.2).

HTML entity

HTML entity jsou kódy ve zdrojovém textu webové stránky, které jsou prohlížečem zpracovány a zobrazeny jako různé speciální znaky, které:

- není možné napsat na běžné klávesnici (např. ©),
- mají ve zdrojovém kódu stránky zvláštní význam (např. `<`, `>`)
- mohou být běžné znaky, zapsané pomocí svého kódu zápisem `&#65;` (`65` je kód znaku – v tomto případě odpovídá písmenu `A`)

Nejpoužívanější HTML entity jsou v tabulce 3.1, více o HTML a XML entitách je v [12].

znak	kód	znak	kód
(mezera)	` `	°	`&deg;`
©	`&copy;`	&	`&`
<	`<`	§	`&sect;`
>	`>`		

Tabulka 3.1: některé znakové HTML entity

Vše se komplikuje tím, že některé z těchto znaků se mohou ve zdrojovém textu stránky objevit i přímo – tedy v nezakódované podobě. Prohlížeč je pak zobrazí stejně, jako kdyby zakódované byly, ale při jejich označení a předání pomocí JavaScriptové funkce (viz 3.1.2) předává entity vždy nezakódované – bez ohledu na to, v jaké podobě jsou ve zdrojovém textu. To způsobuje problémy s vyhledáváním řetězců, obsahujících tyto entity.

Situaci se aplikace snaží zmírnit tím, že pokud tento případ nastane, převede některé znaky, které se často vyskytují v zakódované podobě, na entity a pokusí se řetězec najít znovu.

Tento problém se však vyskytuje jen při definici proměnné a tak pokud ani tentokrát není vybraný text v tagu nalezen, je uživatel vyzván k výběru správné části řetězce.

3.1.3 Cesta stromem tagů

Při ukládání definice proměnné je třeba uložit cestu stromem tagů od kořene k vybranému tagu. Přitom se snažíme zachovat možnost nalezení proměnné i při méně zásadních změnách zbytku zdrojové stránky. Proto aplikace neukládá do cesty všechny otvírací a zavírací tagy, ale jen ty, které jsou nutné k opětovnému nalezení cílového tagu.

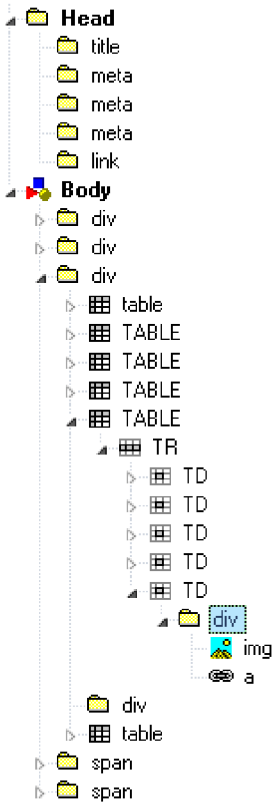
Na obrázku 3.1 je příklad stromu tagů v typické HTML stránce. Cílový tag (obsahující uživatelem vybranou hodnotu) je označen.

Aplikace sestavuje cestu k cílovému tagu tak, že obsah tagu, který je při průchodu stránkou uzavřen ještě před nalezením cílového tagu se neukládá. Ukládá se jen informace o jeho přítomnosti (ve formě `</head>`). Ukládají se však všechny otevírací tagy, jejichž zavírací tag ještě načten nebyl. Takto se postupuje až do nalezení cílového tagu (podle ID, předaného dříve Javascriptovým kódem).

Odpovídající cesta k označenému tagu `<div>` podle popsaného postupu tedy je:

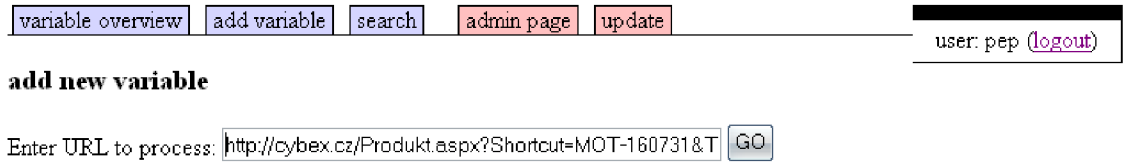
`<head /> <body> <div /> <div /> <div> <table /> <table /> <table /> <table /> <table> <tr> <td /> <td /> <td /> <td /> <td> <div>`.

Výhodou je, že obsah tagů uvedených ve tvaru `</head>` se může libovolně měnit a nalezení proměnné to neovlivní. Nevýhodou je naopak náchylnost na změnu struktury stránky před cíovým tagem (definovanou proměnnou), pokud se změněný (případně přidaný nebo odstraněný) tag nachází na cestě k cílovému tagu.



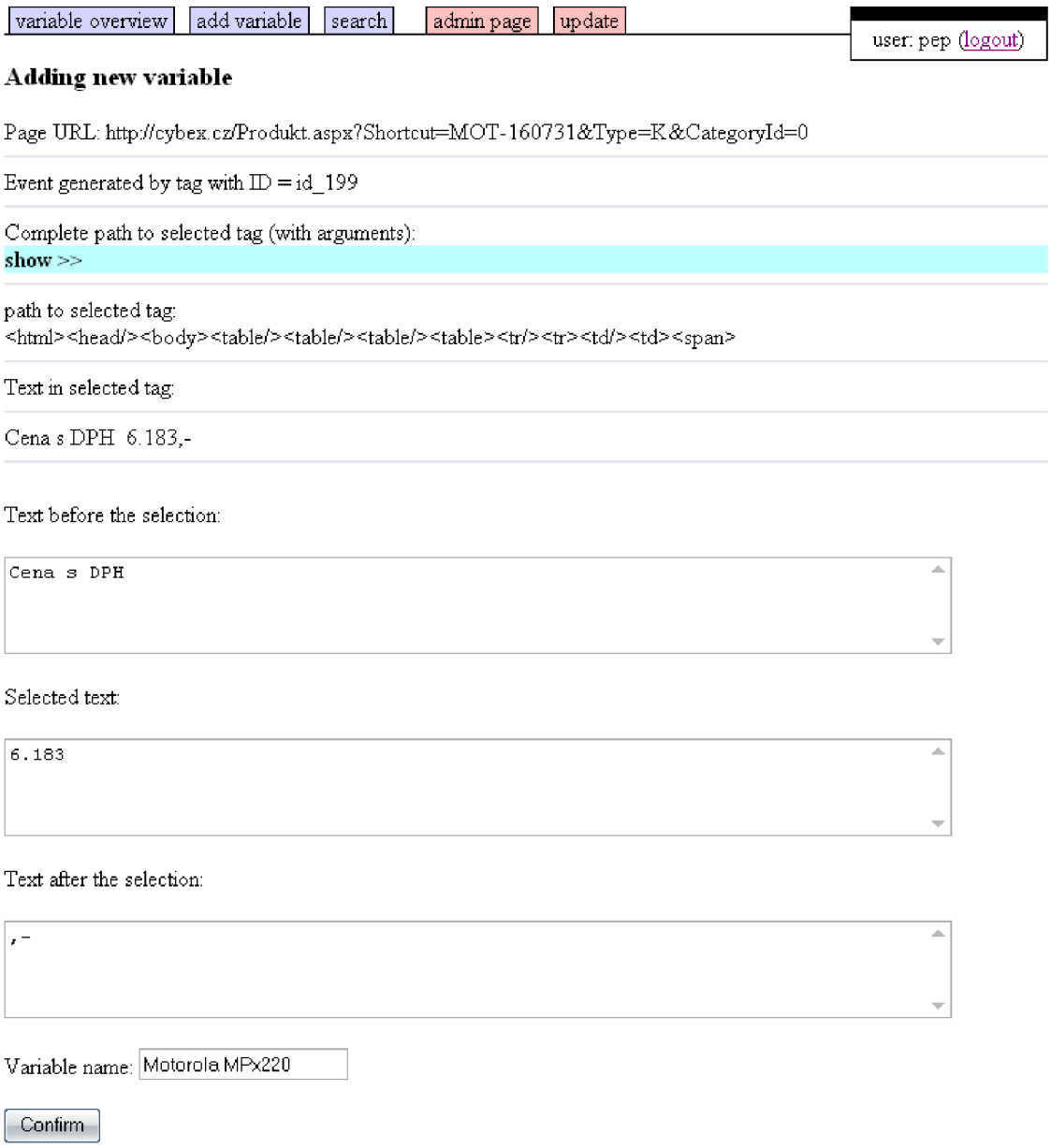
Obrázek 3.1: Typický strom tagů v HTML stránce

Ukázka definice proměnné



Obrázek 3.2: Přidání nové proměnné – zadání URL stránky s daty

Obrázek 3.3: Přidání nové proměnné – výběr sledovaných dat na stránce (cena s DPH)



Obrázek 3.4: Přidání nové proměnné – potvrzení cesty, výběru a okolních textů

3.1.4 Aktualizace dat

Aktualizace je spouštěna periodicky a zajišťuje načtení nových hodnot sledovaných proměnných a jejich uložení do databáze.

Aktualizace proměnné

Aplikace při aktualizaci proměnné načte aktuální obsah příslušné stránky a podle uložené cesty se snaží nalézt novou hodnotu proměnné. Prochází přitom všechny tagy na stránce obdobně, jako při vytváření cesty k proměnné (viz. 3.1.3). S tím, že obsah uzavřených tagů nebere v úvahu.

Načtené tagy si musí navzájem odpovídat, jinak aktualizace selže a proměnná již nebude v budoucnu aktualizována (do zásahu uživatele).

Pokud je nalezen i poslední tag uloženeé cesty, jedná se o hledaný cílový tag. Pokud je proměnná definována jen jako část obsahu cílového tagu, je tato část nalezena a uložena, jinak se jako nová hodnota proměnné uloží celý obsah tagu.

Plánování aktualizací

Aktualizace proměnných probíhá automaticky v definovaných intervalech. Vzhledem k omezení maximální doby běhu PHP skriptu však nemusí být možné tuto aktualizaci provést najednou pro všechny proměnné. Proto je třeba pravidelně spouštěný skript spouštět častěji než je zvolený interval aktualizací.

Pokud aktualizace proměnné proběhne úspěšně, nastaví se jí příslušný příznak. Pokud ale není aktualizace dokončena (může to mít více příčin – velká zátěž serveru projektu, nedostupnost cílového serveru,...), tak se při příštím spuštění aktualizačního skriptu provede další pokus. Pokud i ten selže, proměnná se v této sérii aktualizací přeskočí.

Pokud aktualizace proběhla, ale proměnná nebyla na stránce nalezena (nastává v případě změny adresy, nebo struktury stránky), pak se proměnná již aktualizovat nebude a je nutný zásah uživatele. Ten ji může buď znovu definovat, nebo změnit příslušné URL.

Tento postup se opakuje dokud ještě nebyly aktualizovány všechny proměnné. Pak už aplikace jen čeká na konec definovaného intervalu mezi aktualizacemi.

3.1.5 Přehled proměnných, historie změn a vyhledávání

Přehled proměnných

Aplikace na své hlavní stránce zobrazuje přehled všech definovaných proměnných s jejich aktuálními hodnotami a časem jejich získání. Také jsou zobrazeny definice, jejichž aktualizace selhala, spolu s možností změny (viz. 3.1.5).

variable overview | add variable | search | admin page | update | user: pep (logout)

monitored variables

	url	name	last update	last value	history
✕	http://www.alfacomp.cz/php/zbozi.php?akc=2&mnu=2&zob=2&pol=50&zbo=18022	DIMM DDR 1GB	2006-04-14 10:05:19	2 080	view history
✕	http://shm-km.wz.cz/hlavni.php	pocet navstev	2006-04-14 10:05:12	3126	view history
✕	http://seznam.cz	kdo ma dnes svatek	2006-04-14 10:05:12	Vincenc	view history
✕	http://www.czechcomputer.cz/product.jsp?artno=35367	Sapphire Atlantis ATI Radeon X1600 Pro 256MB	2006-04-14 10:05:16	4 070	view history
✕	http://www.czechcomputer.cz/product.jsp?artno=31625	athlon 64 3000+	2006-04-14 10:35:07	3 672	view history
✕	http://cybex.cz/Produkt.aspx?Type=K&CategoryId=609&Shortcut=MSI-CAX51121	Microstar Star Key v2.0, USB Bluetooth	2006-04-14 10:35:09	612	view history
✕	http://cybex.cz/Produkt.aspx?Shortcut=NOK-N90&Type=K&CategoryId=0	Nokia N90	2006-04-14 10:35:11	17.456	view history
✕	http://kasa.cz/audio-video-technika/koupit/90259/lg/lg-42px3rva/http://kasa.cz/audio-video-technika/koupit/90259/lg/lg-42px3rva/	LG 42PX3RVA	2006-04-14 10:35:11	39 591.00	view history
✕	http://electrohall.cz/store/goodsdetail.asp?sCGoodsID=SE01378959	Sluchátka bezdrátová Thomson WHP 360D	2006-04-14 10:35:12	1999	view history
✕	http://seznam.cz	kurz eura	2006-04-14 10:35:13	28,55	view history
✕	http://seznam.cz	diakritika v jiném kódování (UTF-8)	2006-04-14 10:35:13	či	view history
✕	http://novinky.cz	kurz dolaru	2006-04-14 10:35:13	23.615	view history

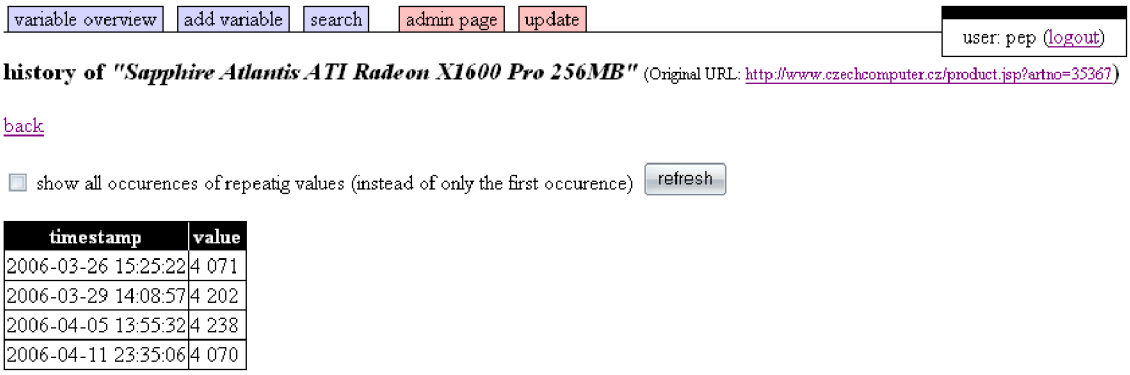
Obrázek 3.5: Přehled sledovaných proměnných

Historie

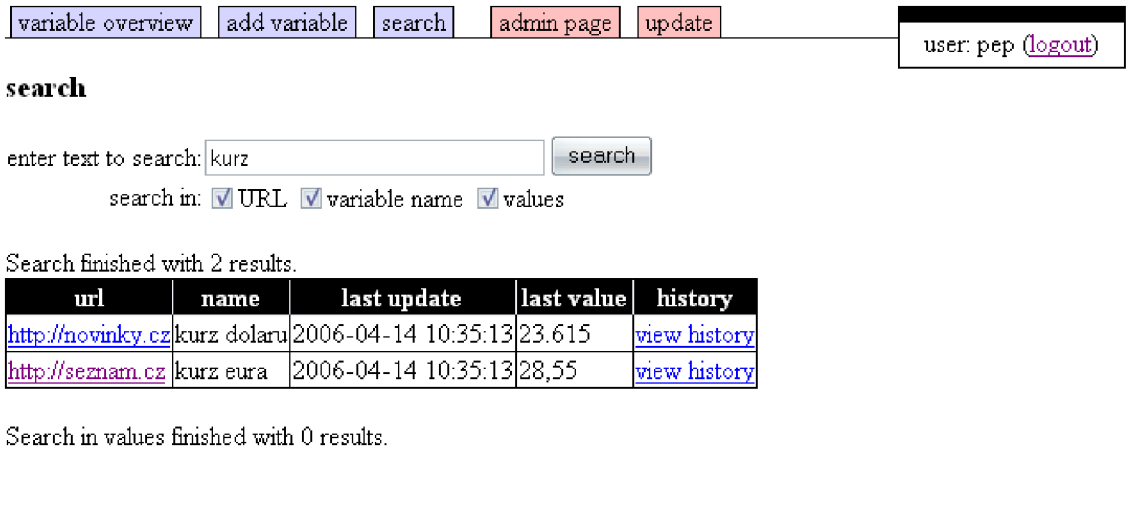
U všech sledovaných proměnných je možné jednoduše prohlížet historii změn (viz. 3.1.5).

Vyhledávání

Aplikace umožňuje vyhledávat v názvech proměnných, jejich URL a v historii hodnot všech proměnných (viz. 3.1.5).



Obrázek 3.6: Detail historie jedné proměnné



Obrázek 3.7: Vyhledávání ve sledovaných proměnných

3.2 Lixto

Lixto je vyspělý komerční nástroj, postavený na práci prezentované v příspěvku [2]. Dnes je součástí skupiny produktů zaměřených na získávání dat (nejen) z webových stránek.

3.2.1 Získání webové stránky

Aktuální verze Lixto Visual Developer ([7]) využívá prostředí Eclipse a integruje prohlížeč Mozilla. Prostřednictvím prohlížeče může uživatel zobrazit webovou stránku běžným způsobem. Lixto Visual Developer dovoluje uživateli vytvořit posloupnost akcí na webové stránce (psaní textu, akce myší,...), které vedou k získání stránky určené k extrakci dat. Tímto způsobem je možné extrahovat data ze stánek používajících moderní webové technologie, jako je AJAX.

Lixto také podporuje HTTP autentizaci a protokol HTTPS.

3.2.2 Definice dat na stránce

Definice dat na webové stránce se nazývá "pattern", tedy jakýsi vzor. Tyto vzory jsou interně reprezentovány programem ve speciálně vyvynutém deklarativním jazyce Elog. Uživatel však s ním nemusí přijít do styku. Každý vzor je definován jedním nebo více filtry. Více filtrů umožňuje rozšířit definici vzoru. Filtr se skládá z podmínek. Podmínky naopak definici omezují. Podporováno je několik druhů podmínek:

before/after – Požadavek na přítomnost určitého elementu před/za výskytem vzoru.

notbefore/notafter – Požadavek na nepřítomnost určitého elementu před/za výskytem vzoru.

vnitřní podmínky – Požadavek na přítomnost/nepřítomnost určitého elementu uvnitř výskytu vzoru (jako podelement).

rozsah – Podmínka, která omezuje výběr jen některých instancí vzoru (např. první, nebo poslední výskyt).

Elementy můžou být definovány také hodnotami atributů (typ písma, velikost,...), případně obsahem.

Mezi vzory může být hierarchie – je možné například definovat vzor "kniha", uvnitř kterého budou vzory "název" a "autor". Instance takových vzorů se pak mohou vyskytovat jen uvnitř instance vzoru nadřazeného.

3.2.3 Vytvoření definice

Lixto Visual Developer poskytuje prostředky pro intuitivní vytváření definic dat – vzorů. Jak je vidět na webcastu na stránkách výrobce ([8]), uživatel může vytvořit nový vzor jednoduše kliknutím načást stránky, která ho zajímá. Vzor pak může upravit přidáním filtrů a podmínek. Aplikace pak sama vygeneruje odpovídající kód vjazyce Elog a vzor okamžitě aplikuje na zobrazenou stránku, takže uživatel vidí, jaké instance vzoru byly na stránce nalezeny.

3.2.4 Extrakce dat

Při definici dat je pro každý vzor vytvořen program v jazyce Elog. Elog je jazyk podobný datalogu a má tedy syntaxi podobnou prologu. K popisu vzoru tedy používá predikáty. Některé popisují umístění instancí vzoru v modelu dokumentu, případně umístění ve vztahu k jiným elementům, další predikáty se vztahují k hodnotám atributů. Predikáty jsou voleny tak, aby umožňovaly při definici jednoduše popsat filtry a jejich podmínky.

Model dokumentu

Model dokumentu v Lixto je strom, který odpovídá tagům ve zdrojovém kódu webové stránky. Každý uzel je reprezentován množinou, kterou tvoří název odpovídajícího elementu, všechny jeho atributy a jejich hodnoty a také imaginární atribut "elementtext", který obsahuje čistý text uvnitř daného elementu (včetně textu elementů pod ním ve stromové struktuře)

Mechanismy extrakce

Lixto nabízí dva mechanismy extrakce dat – stromovou a řetězcovou. Oba lze použít v jazyce Elog.

V extrakci založené na stromech jsou elementy identifikovány cestou ve stromu tagů dokumentu (s případnými hodnotami atributů). Přitom je možné použít neúplnou definici cesty pomocí symbolu "*". Příkladem takové cesty může být `*.p.*table.*.td`.

Řetězcová extrakce se používá pokud je třeba identifikovat část řetězce. K tomu se používá regulární výraz. Obě metody je možné kombinovat – lze tak definovat cestu stromem k elementu s textovým obsahem a v tomto obsahu najít požadovaná data pomocí regulárního výrazu.

3.2.5 Uložení dat

Lixto umožňuje uložení získaných dat v XML formátu, který je vhodný pro další zpracování. Implicitně jsou jako XML elementy volena jména vzorů. Je také zachována hierarchie vzorů. Podobu výstupu je navíc možné ovlivnit - lze například zvolit které atributy se uloží do XML.

Další možnosti

Lixto dokáže při extrakci dat procházet více webových stránek prostřednictvím nalezených odkazů. Může tak hledat zadané vzory na celých webech.

Aktuální verze obsahuje i konvertory známých kancelářských formátů (PDF, DOC, XLS, CSV,...). Takové soubory dokáže stáhnout z webu a jejich obsah zpracovat podobným způsobem jako webovou stránku.

V neposlední řadě je vhodné zmínit i produkt Lixto Transformation Server [6], který mimo jiné slouží jako platforma pro automatické spouštění pravidel pro extrakci dat vytvořených ve Visual Developeru.

Kapitola 4

Návrh řešení

4.1 Cíl práce

Mým cílem bylo navrhnout aplikaci pro extrakci dat z webových stránek. Dále jsem chtěl, aby šlo o webovou aplikaci. Tedy aby ji nebylo nutné instalovat na klientský počítač. Data ve formátu XML jsou tak přístupná na www serveru (tedy i pro další aplikace přes HTTP protokol).

Shrnutí výhod a nevýhod tohoto přístupu:

+ aplikace je přístupná přes internetový prohlížeč bez nutnosti instalovat další SW
+ získaná je možné použít v dalších aplikacích (ve formátu XML)
+ webové stránky s daty jsou zobrazeny přímo prohlížečem uživatele
– k definici dat na stránce je nutná úprava jejího kódu a její zobrazení serverem aplikace
– menší kontrola nad zobrazenou stránkou, ve srovnání se samostatnou aplikací a integrovaným jádrem prohlížeče (viz. Lixto 3.2)
– menší možnosti uživatelského rozhraní ve srovnání se samostatnou aplikací

4.1.1 Názvosloví

V této části textu budu používat následující termíny v uvedeném významu. Uvádím i odpovídající anglický výraz, který nemusí být vždy přesným překladem – pro účely tohoto projektu a výsledné aplikace (která bude v anglickém jazyce) budou však takto definované dvojice výrazů ekvivalentní:

strom tagů, strom tagů stránky, strom dokumentu (document tree, page tree) – stromová reprezentace zdrojové stránky

proměnná (variable) – nejmenší entita při definici dat; je vždy součástí vyhledávacího vzoru

vzor, vyhledávací vzor (pattern) – definice místa v dokumentu (webové stránce), kde se nachází sledovaná data ve formě proměnných; je reprezentován vyhledávacím stromem

vyhledávací strom (search tree) – reprezentace vyhledávacího vzoru

instance vzoru (pattern instance) – úsek na webové stránce, který odpovídá některému vzoru

navigační krok (navigation step) – interakce s webovým serverem jako při běžném prohlížení (zadání adresy, odeslání formuláře, následování odkazu)

navigace (navigation) – posloupnost navigačních kroků

šablona (wrapper) – nejmenší (a jediná) entita pro kterou lze spustit extrakci dat; obsahuje navigaci a vyhledávací vzory; začíná vždy navigačním krokem (URL), dále již se může navigace se vzory libovolně střídat

extrakce dat – získání aktuálních hodnot proměnných, definovaných ve vyhledávacím vzoru

Uvádím ještě další termíny, které mají svůj obecný význam, ale v této práci je používám ve zcela konkrétním významu, který by nemusel být na první pohled zřejmý:

server aplikace – protože moje práce je webovou aplikací, budu tímto termínem označovat (webový) server, na kterém bude tato aplikace spuštěna

aplikace – aplikace jako celek, bez rozlišení jednotlivých částí – viz. diagram architektury 4.1.3

webová stránka, nebo jen stránka – pokud nebude z kontextu zřejmý jiný význam, pak se jedná o konkrétní zpracovávanou stránku, která je v popisovaný okamžik zobrazena v okně prohlížeče (v době definice šablony – v tom případě se všechny definice vzorů a jeden následující navigační krok vztahují k této stránce), nebo kterou aplikace načetla a právě zpracovává (v případě extrakce dat)

okno prohlížeče – součást uživatelského rozhraní při úpravách šablony; zobrazuje načtenou webovou stránku (s přidaným JS kódem a kódem appletu)

okno appletu – druhá část uživatelského rozhraní pro úpravu šablony; umožňuje celý proces ovládat

uzel – uzel ve stromu tagů stránky – může jít o tag, nebo textový uzel

textový uzel – prostý text v HTML stránce; jde např. o řetězece "klikněte" a "zde" v kódu `<span>klikněte<a href="#">zde</a>`

tag – HTML tag uzavřený v úhlových závorkách; může jít o párový i nepárový tag, s potomky i bez

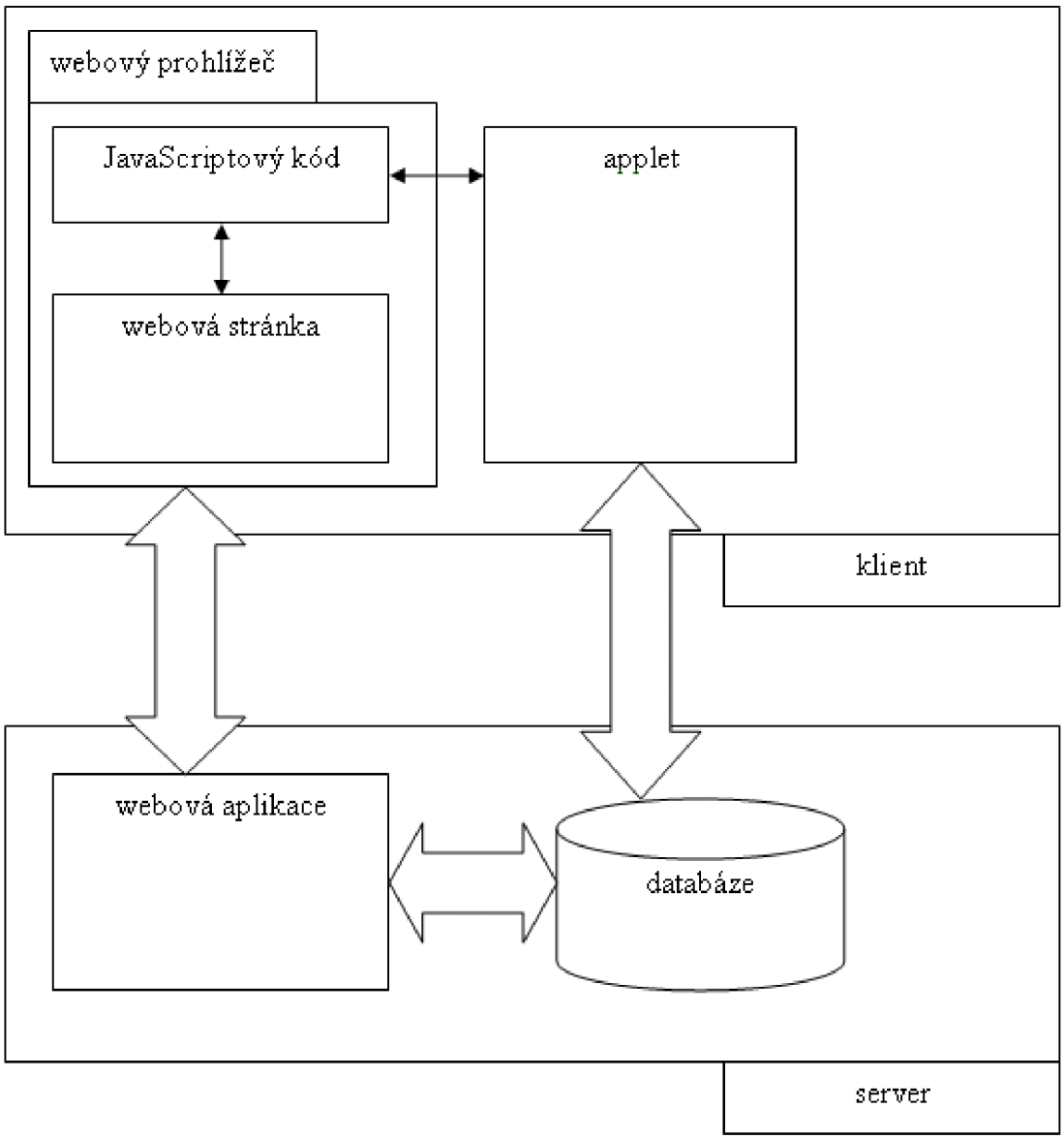
4.1.2 Části problému

Problém získávání dat z webových stránek je možné rozdělit na tyto části:

- Získání webové stránky
- Definice sledovaných dat
- Extrakce dat

Výstupem prvních dvou částí je šablona, obsahující definici navigace a vzorů pro vyhledávání proměnných. Ve třetí části se pak podle této šablony získají aktuální data, která se připraví pro další zpracování.

4.1.3 Architektura aplikace



Obrázek 4.1: Architektura aplikace

4.2 Získání webové stránky s daty

Nejjednodušší způsob jak získat webovou stránku pro extrakci dat je zadáním pevné adresy (URL). Takto pracuje například aplikace v mé bakalářské práci. V praxi však tento způsob často nestačí. Užitečnější by byla možnost automaticky získat data z většího počtu stránek – například zboží v celém obchodě. Někdy také pro přístup k požadovaným údajům musíme vyplnit webový formulář – příkladem mohou být výsledky vyhledávání konkrétního výrazu,

nebo hledání spojení v jízdních řádech. Tyto případy se budu snažit postihnout.

Posloupnost kroků, které by za běžných okolností prováděl uživatel prohlížeče, aby si mohl informace přečíst, nazývám "navigace". Aplikace musí poskytovat jak způsob zadání těchto kroků, tak jejich simulaci před samotnou extrakcí. Proto jsem navrhl následující postup:

4.2.1 Zpracování původní stránky

K tomu, abychom mohli zaznamenávat aktivitu uživatele na stránce pro pozdější použití při extrakci musíme zdrojový text stránky upravit. Úpravy jsou potřeba i kvůli možnosti definice dat. Po zadání adresy stránky tedy aplikace stáhne její zdrojový kód a provádí tyto úpravy:

Změna relativních URL

Zpracovaná stránka bude zobrazena serverem aplikace. Proto by se veškeré adresy prvků vložených na stránku pomocí relativní cesty staly neplatnými. Pro naše účely jde zejména o vložené obrázky, kaskádové styly a JavaScriptový kód, které by v tomto případě mohly způsobit výrazně odlišný vzhled zpracované stránky. Je tedy nutné takovéto relativně zadané cesty převést na absolutní (konkrétně jde o tagy `<img>`, `<a>` a `<link>`).

Změna odkazů a formulářů

Pro účely záznamu navigace je třeba upravit odkazy na stránce tak, aby vedly zpět do aplikace, která tak zjistí, že uživatel odkaz použil, přidá příslušný navigační krok do šablony a přejde na stránku určenou odkazem. Stejně je třeba ošetřit formuláře – zadaná data se pak odešlou nejprve do aplikace a ta je teprve odešle původnímu serveru. Jeho odpověď poté zpracuje jako další stránku.

Přidělení identifikátorů

Tagům na stránce bude přidělen unikátní identifikátor prostřednictvím atributu `ID`. Spolu s reakcí na událost kliknutí myší, slouží tento identifikátor k usnadnění definice vyhledávacího vzoru.

Přidání kódu v jazyce JavaScript, vložení klientského appletu a ošetření událostí

Do stránky je vložen Javascriptový kód a applet, který bude tvořit uživatelské rozhraní pro vytváření šablony.

4.2.2 Navigace

Vytvoření navigačního kroku

První navigační krok je vytvořen automaticky z adresy, kterou uživatel zadal při vytváření nové šablony. Stránka s touto adresou se zároveň zpracuje a zobrazí. Otevře se také okno appletu v režimu navigace (dalším je režim definice). V okně je také vidět vytvářená šablona, kterou reprezentuje posloupnost navigačních kroků a vyhledávacích vzorů (nyní obsahuje jedinou položku – navigační krok otevření zadaného URL). Uživatel může následovat odkazy na stránce a vyplňovat formuláře. Tyto činnosti jsou přesměrovány přes server aplikace a pokaždé je vytvořen příslušný navigační krok.

Typy navigačního kroku a odpovídající vlastnosti

Používám tři typy navigačních kroků, podle toho jakou akci provádí. Typ určuje také vlastnosti, které lze upravovat.

- **Typ URL**

  Používá se k načtení stránky na konkrétní URL.

- **Typ formulář**

  Krok vzniká vyplněním webového formuláře. Formulář může využívat HTTP metody GET, nebo POST, v obou případech jsou odeslaná data uložena do navigačního kroku.

- **Typ robot**

  Tento typ musí být přidán ručně. Na stránce vyhledá všechny odkazy a použije je pro další vyhodnocování. Lze nastavit hloubku hledání odkazů (hodnota hloubky rovna 0 znamená, že se použijí pouze odkazy z aktuální stránky) a jestli mají být použity i odkazy vedoucí na jiné servery. Více viz. 5.4.3.

4.3 Definice sledovaných dat

K definici dat používám vyhledávací stromy. Vyhledávací strom popisuje umístění hledaných proměnných na stránce. Je to souvislý podstrom stromu tagů. Proto se také při vytvoření nového vyhledávacího stromu použije kopie celého stromu tagů. Pokud je okno appletu v režimu definice dat, pak kliknutím na obsah libovolného tagu v okně prohlížeče se tento tag zvýrazní a zároveň se vybere odpovídající uzel ve vyhledávacím stromu.

V okně appletu je zobrazen detail vybraného uzlu vyhledávacího stromu se všemi jeho vlastnostmi. Dále jsou pak zobrazeny atributy odpovídajícího uzlu ve stromu tagů.

S každým uzlem vyhledávacího stromu lze provést tyto akce:

z uzlu vytvořit kořen vyhledávacího stromu z vyhledávacího stromu se smažou všechny nadřazené uzly a všechny uzly mimo podstrom s tímto kořenem

odstranit syny z vyhledávacího stromu jsou odstraněny všechny poduzly

odstranit uzel z vyhledávacího stromu je odstraněn uzel i všechny jeho poduzly

4.3.1 Vlastnosti uzlu vyhledávacího stromu

Uzlem vyhledávacího stromu může být tag, nebo textový uzel. Oba typy uzlů je možné použít pro omezení vyhledávacího vzoru a/nebo pro definici proměnných.

Atributy

Tagům je možné zadat atributy a jejich hodnoty, které jsou pak podmínkou k nalezení tagu při extrakci.

Textový obsah

Specialitou textových uzlů je naopak možnost zadat text, který mají obsahovat. Uzel je při extrakci nalezen jen v případě že se na daném místě nachází textový uzel, který obsahuje zadaný řetězec.

Proměnné

Tagy mohou obsahovat proměnné, jejichž hodnoty se hledají při extrakci. Jako proměnnou je možné označit:

- textový obsah tagu (včetně vnořených uzlů)
- hodnotu zadaného atributu
- test na přítomnost zadaného atributu

Textový uzel může být také chápán jako proměnná, platí však dvě omezení: je možné použít jen proměnnou typu "textový obsah" – hodnotu proměnné tak tvoří celý textový uzel; platí to přitom jen tehdy, když nastavena podmínka na obsah uzlu – podmínku a proměnnou není možné nastavit současně.

Každá proměnná má povinně jméno.

4.4 Extrakce dat

4.4.1 Zpracování šablony

Šablona je předpis pro získání dat. Obsahuje chronologicky uspořádané navigační kroky a vyhledávací vzory, jak byly popsány výše. Nic přitom uživateli nebrání umístit za vyhledávací vzor(y) další navigační kroky.

Při zpracování šablony tedy aplikace postupuje stejně, jako uživatel při jejím vytváření – postupně prochází navigační kroky a pokud narazí na vyhledávací vzor, pokusí se jej najít na aktuální stránce. Totéž následuje nezávisle i pro případné další vzory definované pro danou stránku a to až do výskytu dalšího navigačního kroku. Pokud algoritmus nalezne navigační krok typu robot, jehož výsledkem je více stránek, pak se provádění rozvětví a další postup se provádí pro každou stránku zvlášť.

4.4.2 Nalezení definovaných dat na stránce

Pro nalezení hodnot proměnných je třeba ve stromu tagů stránky najít instance vyhledávacího vzoru. Berou se přitom v úvahu i všechny nastavené atributy uzlů vyhledávacího stromu.

4.5 Uživatelské rozhraní

Uživatelské rozhraní má dvě části – první tvoří webové stránky aplikace a stránky, které uživatel prochází při definici šablony. Druhou částí je pak okno appletu. Toto okno je zobrazeno v průběhu vytváření šablony a slouží ke kontrole celého procesu tvorby šablony. Applet se stránkou v prohlížeči komunikuje prostřednictvím Javascriptového kódu vloženého na stránku.

Okno appletu může být ve dvou módech:

- mód navigace – slouží k vytváření navigačních kroků; v okně prohlížeče je možné následovat odkazy a odesílat formuláře – tyto akce se zaznamenají do šablony
- mód definice – slouží k vytváření vyhledávacího stromu; odkazy v okně prohlížeče nefungují, namísto toho kliknutí na stránce zvýrazní celý tag a vybere odpovídající uzel v právě vytvářeném vyhledávacím stromě

Poznámka: Ve skutečnosti používám pro každý mód samostatný applet; applet, který je použit při navigaci, tak může mít menší velikost a je tak rychlejší, než kdyby obsahoval kód pro práci s vyhledávacími vzory.

4.5.1 Navigace

Vytvoření šablony Novou šablonu vytvoříte na hlavní stránce aplikace (se seznamem šablon), kliknutím na odkaz "Vytvořit novou šablonu" a vyplněním webové adresy.

Zadaná adresa se zobrazí v okně prohlížeče a otevře se okno appletu v režimu navigace viz. 4.2.

Navigace V režimu navigace můžete procházet webové stránky standartním způsobem. Je možné procházet odkazy i vyplňovat webové formuláře. Každý přechod na další stránku se přitom přidá jako nový řádek v okně appletu. Zde je možné jednotlivé kroky krok odstranit, přidat krok typu robot, zadat název šablony, přepnout se do tvorby vyhledávacích vzorů (tlačítko "přidat vzor") a nebo šablonu uložit.

Krok typu Robot Krok typu robot přidáte tak, že označíte navigační krok, který vede na stránku, na které chcete nechat vyhledat odkazy pro extrakci, zadáte hloubku zanoření, zvolíte, jestli se mají následovat odkazy jen v rámci serveru a tlačítkem "přidat/upravit" vložíte nový krok typu robot. Stejným tlačítkem je později možné vybraný robot upravit.

Obrázek 4.2: Navigační mód

4.5.2 Tvorba vyhledávacího vzoru

Úprava stromu Po přepnutí do režimu definice vyhledávacího vzoru je okno prázdné – až po kliknutí na některý z tagů na stránce se vytvoří strom z tagů a textových uzlů a vybraný uzel se v tomto stromě zvýzarní. Nyní máte možnost upravit strom pomocí třech tlačítek nad stromem. viz. 4.3

Atributy U vybraného uzlu vyhledávacího vzoru můžete nastavit požadavky na přítomnost (a případně hodnotu) libovolných atributů. Automaticky jsou přidány atributy nalezené na stránce. Všechny atributy můžete jednoduše smazat odškrtnutím políčka "atributy". Nový atribut můžete přidat tlačítkem "přidat", následně upravte jméno a hodnotu atributu a tlačítkem "uložit" vše potvrďte.

Proměnné Práce s proměnnými je podobná – po kliknutí na tlačítko "přidat" můžete vyplnit vlastnosti proměnné, které pak tlačítkem "uložit" potvrdíte.

Textové uzly Pokud ve stromě vyberete textový uzel, můžete mu nastavit buď text, který má na daném místě být, nebo můžete z textového obsahu vytvořit proměnnou. Změny je opět třeba uložit tlačítkem "uložit".

4.6 Implementační prostředky

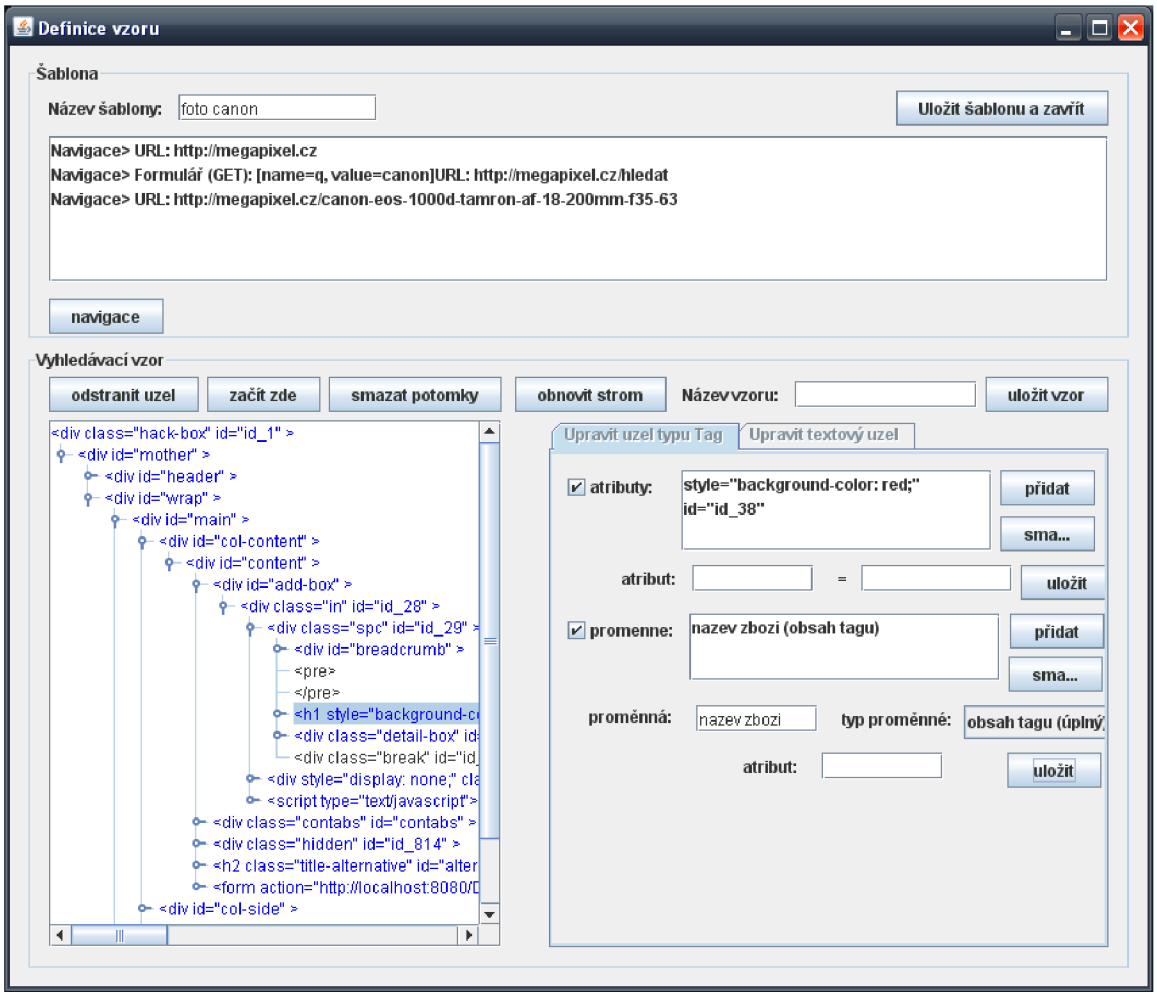
4.6.1 Požadavky na platformu

Při práci na semestrálním projektu jsem zvažoval možné implementační platformy pro svou práci. Z návrhu řešení vyplynuly některé požadavky, které by měly splňovat použité nástroje. Jde zejména o (snadnou a efektivní):

- práci s připojením k www serveru (čtení www stránky), komunikace HTTP protokolem na úrovni odeslání POST požadavku
- možnost zobrazit www stránku v prohlížeči a možnost definice dat
- možnost připojení k databázi
- práci s ADT strom, případně dalšími algoritmy nad stromy
- práci se seznamy

Mezi uvažovanými technologiemi byly:

- PHP
- .NET, C#, ASP.NET
- Java, JSP



Obrázek 4.3: Definice vyhledávacího vzoru

Zhodnocení platforem

Nejprve je třeba zmínit, že pro některé úkoly není jiná možnost, než využít JavaScript. Jde zejména o definici hledaných dat ve webovém prohlížeči.

Dále pak pro následné úpravy definice dat na stránce bylo vhodné (zejména bereme-li v úvahu uživatelskou přívětivost) vytvořit rozhraní, které by tyto úpravy provádělo na straně klienta. Považoval jsem za nejvýhodnější použít pro tyto účely Java applet.

Nakonec jsem se rozhodl použít Javu i na serverové straně v podobě JSP, protože to přináší výhodu v možnosti použít kód serverové části a jejích knihoven i v appletu .

4.7 Použité knihovny

V této sekci popisuji knihovny třetích stran, které jsem ve své práci použil a způsob, jakým je používám.

4.7.1 HTML Parser

Ve svém projektu zpracovávám HTML kód a proto potřebuji parser (X)HTML dokumentů. Namísto vývoje vlastního parseru jsem se rozhodl použít některou knihovnu, dostupnou pod volnou licencí. Nakonec jsem zvolil projekt s prostým názvem *HTML Parser*, který má domovskou stránku na adrese `http://htmlparser.sourceforge.net/` a kromě očekávané možnosti získat HTML dokument ve formě stromu tagů (se snadným přístupem k jejich atributům) nabízí také další užitečné nástroje a vlastnosti. Ty nejdůležitější nyní popíšu.

Třídy a rozhraní pro reprezentaci obsahu dokumentu

Node Rozhraní `Node` představuje obecně uzel v HTML dokumentu. Ať jde o tag, volný text, nebo komentář (komentáře ve svém projektu vynechávám).

Tag Rozhraní `Tag` slouží k reprezentaci tagu; tag může obsahovat atributy, potomky (obecně uzly) a rozhraní `Tag` nabízí metody pro jejich procházení, změnu i vytváření nových.

Text Rozhraní `Text` reprezentuje textový uzel a jeho vlastnosti se proto omezují na práci s jeho textovým obsahem.

Attribute Třída `Attribute` slouží k uložení atributů tagů a jejich hodnot.

NodeList Třída `NodeList` je seznam uzlů. Tvoří základ pro každý strom uzlů (seznam přímo obsahuje kořenové uzly, které mohou mít potomky, čímž vzniká strom). `NodeList` nabízí řadu užitečných metod. Umožňuje s uzly manipulovat, převést seznam uzlů zpět na HTML kód, nebo jen zobrazit textové uzly (podobně, jako to dělá prohlížeč), nabízí iterátor nad seznamem a konečně umožňuje aplikovat na uzly filtr, nebo visitor (oboje viz. dále).

Filtry

Filtry jsou třídy, implementující rozhraní `NodeFilter`. To obsahuje jedinou abstraktní metodu `accept(Node node)`, která vrací logickou hodnotu, podle toho, zda daný uzel vyhovuje podmínkám filtru.

Filtry je možné skládat pomocí existujících filtrů, které implementují logický součin (`AndFilter`), součet (`OrFilter`) a negaci (`NotFilter`) nad jinými filtry. HTML Parser obsahuje několik dalších filtrů, z nichž vybírám ty, které ve svém projektu využívám:

TagNameFilter Je filtr, který vyžaduje tag se zadaným jménem (např. "div").

HasAttributeFilter Vyžaduje přítomnost zadaného atributu, volitelně i se zadanou hodnotou.

HasChildFilter Tomuto filtru vyhoví jen uzel, který má potomka (vyhovujícího zadanému filtru).

StringFilter Filtr na textové uzly; je možné vyžadovat konkrétní textový obsah.

Kombinaci těchto filtrů využívám při sestavování vyhledávacího vzoru - více viz. 5.5.3.

Použití filtru Vytvořený filtr je možné aplikovat jednak hned při parsování stránky, ale také na objekt třídy `NodeList` - při aplikaci filtru se pak projde celý strom uzlů a výsledkem této operace jsou jen ty tagy, které vyhověly filtru.

Na obrázku 4.4 je vidět úsek HTML stránky a výsledek následujícího filtru (zápis je zjednodušený):

```
AndFilter(
TagNameFilter( "LI" ),
HasChildFilter( StringFilter( "fotky" ) ),
HasChildFilter( TagNameFilter( "A" ) )
)
```

Filtr hledá tedy výskyty tagu `<LI>`, které obsahují textový uzel a odkaz; textový uzel přitom musí obsahovat řetězec "fotky" (porovnávání nerozlišuje velikost písmen). Z obrázku je vidět, že filtr nalezl 5 tagů, vyhovujících danému filtru (dva nevyhovující tagy `<LI>` - druhý a poslední - jsou vynačeny - první nebyl nalezen kvůli chybějícímu tagu `<A>` a druhý neobsahuje řetězec "fotky"). Také si můžeme všimnout faktu, že výsledkem filtru jsou všechny synovské uzly nalezeného tagu (i ty, které nebyly ve filtru) - s tímto je třeba počítat při vyhodnocování proměnných.

Třída Visitor

Třída `Visitor` poskytuje možnost snadněji projít všechny uzly stromu (seznamu uzlů). Stejně, jako u filtrů, i zde je možné vytvořit vlastní visitor jako potomek třídy `Visitor`. Činnost, která se má s uzly provádět, je třeba uzavřít do metod `visitTag(Tag tag)` nebo `visitNode(Node node)` - podle toho, jestli chceme pracovat s tagy, nebo uzly. Visitor je aplikován na seznam uzlů voláním `NodeList.visitAllNodesWith()`.

Ve svém projektu využívám hned tři vlastní visitory:

linkVisitor Ke změně adres na stránce z relativních na absolutní.

idVisitor K přidání unikátních identifikátorů a JS událostí na stránku (v režimu definice dat).

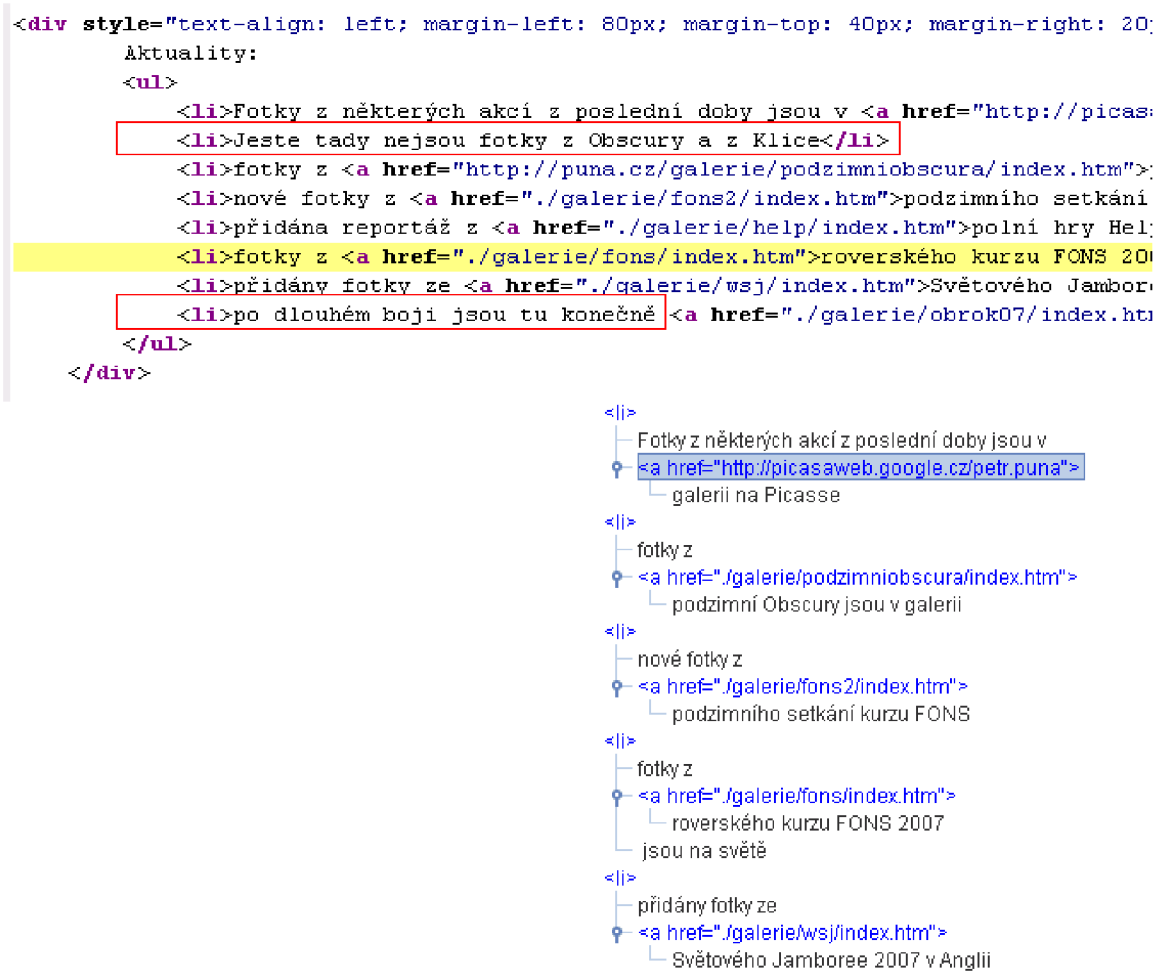
ShowVisitor K odstranění událostí ze stránky pro přehlednější tvorbu vyhledávacího vzoru.

Kódování

HTML Parser se snaží při zpracování stránky přečíst její kódování. Pokud využívá ke stažení stránky standartní třídu `java.net.URLConnection`, dokáže po detekování kódování resetovat čtení stránky a vrátit správně zakódovanou stránku. Já však používám knihovnu HTTPClient (více v následující části) a proto musím detekci kódování věnovat více pozornosti (viz 5.2).

4.7.2 HTTPClient

Ve své práci používám k HTTP komunikaci knihovnu *HttpClient* z balíku *Jakarta Commons*, šířenou pod licencí Apache Source License. Proti standartnímu balíčku `java.net`, nabízí



Obrázek 4.4: Ukázka výsledků filtru

HTTPClient lepší podporu protokolu HTTP, která může být užitečná zejména v případě dalšího rozšíření aplikace. HTTPClient podporuje všechny HTTP metody, umožňuje podporu protokolu HTTPS a dokáže pracovat s cookies. Kompletní přehled podporovaných vlastností lze najít na [5].

Kapitola 5

Řešení vybraných problémů

5.1 Komunikace webové stránky s appletem

Moje aplikace se skládá z části serverové a klientské - ta je pak tvořena oknem prohlížeče a Java appletem. Běžná komunikace klienta se serverem probíhá standartně protokolem HTTP. Zde si však ukážeme jaké jsou možnosti ne příliš obvyklé komunikace mezi html stránkou v prohlížeči (obsahující Javascriptový kód) a Java appletem

5.1.1 Volání metody appletu v JS

Mějme v Java appletu definovanou metodu `setID`:

```
public void setID(String id)
{
this.id = id;
}
```

Applet je vložen na stránku kódem, podobným tomuto:

```
<applet code="applet.class" name="myApplet" archive="applet.jar">
</applet>
```

Nyní se můžeme na applet odkazovat z javascriptu pomocí DOM modelu a to prostřednictvím zadaného jména (`myApplet`). Je tak možné vložit na stránku takovýto tag s JS událostí:

```
<p onclick="document.myApplet.setID('odstavec1');">Vzorový text</p>
```

Po kliknutí na odstavec se text `odstavec1` předá appletu do metody `setID`.

Tento jednoduchý koncept (v rozšířené podobě) využívám ve své aplikaci pro předání vybraného tagu na webové stránce při definici vyhledávacího vzorku. Více o této metodě viz. článek [1].

5.1.2 Volání JS funkce z appletu

Komunikaci opačným směrem není možné realizovat prostým voláním JS funkce, protože applet nemá přístup k DOM modelu stránky. Jistým řešením pro volání JS funkcí je použití protokolu `javascript:` a využití možnosti přesměrování webové stránky. JS funkci `alert()` pro vyvolání dialogového okna tak můžeme z appletu zavolat takto:

```
getAppletContext().showDocument("javascript://alert('zpráva');","_self");
```

Použitím protokolu `javascript:` nedojde k přesměrování stránky, applet zůstane spuštěn a zavolá se příslušná JS funkce.

5.2 Detekce kódování

HTML Parser nabízí ve své třídě `Parser` možnost zjistit detekované kódování stránky – toto kódování je však platné až po parsování stránky. Proto, pokud parser detekuje stránku v jiném kódování, než aplikace předpokládala při vytváření spojení, je třeba připojit se znovu, tentokrát se správně nastaveným kódováním. Abych se tomu ve většině případů vyhnul, snažím se vždy ukládat použité kódování v průběhu procházení a následně extrakce (do session beanu třídy `StateStorage` a do navigačních kroků), předpokládám přitom, že kódování aktuální stránky bude stejné jako u předchozí stránky a také že následující požadavek (request) by měl být ve stejném kódování, jako předchozí odpověď serveru.

5.3 Předzpracování HTML dokumentů

Pro účely aplikace je nutné provést několik úprav v kódu webové stránky. V této sekci je popíšu. Součástí zadání bylo také implementování možnosti dodatečného předzpracování dokumentů, což je zde také popsáno.

Úprava URL Pro zajištění správného zobrazení webové stránky a funkčních odkazů v situaci, kdy je stránka zobrazena serverem aplikace, je nutné změnit všechna relativně zadaná URL na absolutní. Nejde však jen o odkazy a formuláře, ale také o adresy vložených dokumentů (obrázky, CSS stylopisy, JS kód v externích souborech). Moje třída `linkVisitor`, kterou pro tuto úpravu používám proto hledá v tagu atributy `href` (odkazy, `<link>` tagy), `src` (obrázky) a `action` (formuláře) a adresy v nich upraví na absolutní tvar.

V režimu navigace je navíc nutné změnit odkazy tak, aby vedly na server aplikace (původní URL je přitom předáno jako parametr). Změna formulářů je popsána v 5.4.2. Při definici se tato změna neprovádí, aby bylo možné extrahovat původní URL (jen v absolutním tvaru).

Přidání identifikátorů a událostí Pro definici vyhledávacího vzoru je užitečné mít možnost snadno nalézt ve stromu tagů místo na stránce, které nás zajímá (například cenu zboží). Proto v režimu definice zpracovávám webovou stránku i dalším visitorem, nazvaným `idVisitor`. Tento visitor přidává tagům unikátní identifikátor (atribut `id`; identifikátory se generují ve tvaru `id="id_n"`, kde `n` je vzrůstající číslo); pokud tag už identifikátor má, zůstává mu jeho původní hodnota.

Kromě identifikátoru nastavuje `idVisitor` tagům i obsluhu události `onClick` takto:

```
onClick="markit(event); return false;"
```

Funkce `markit()` je definována v souboru `marking.js` a kromě označení daného tagu a předání jeho identifikátoru do appletu předává appletu i celý zdrojový kód stránky, ze kterého pak applet sestaví strom uzlů; příkaz `return false;` způsobuje přerušení kliknutí, pokud je daný tag odkazem.

Vložení JS kódu a appletu Konečně poslední nutnou úpravou je přidání JS kódu (v souboru `marking.js`) a Java appletu na stránku.

5.3.1 Modul pro předzpracování

Aplikaci je možné snadno rozšířit o další předzpracování zdrojového dokumentu, změnou pomocné třídy `Preprocessor`. Tato třída obsahuje dvě statické metody:

- `processForBrowsingAndDefinition()`
- `processForExtraction()`

První jmenovaná je volána při definici šablony (v obou módech) a to ještě před jakoukoli úpravou pomocí visitorů. Druhá metoda je volána při extrakci dat. Úpravou těchto je možné zajistit libovolnou úpravu zdrojového dokumentu.

5.4 Navigace

5.4.1 Odkazy

Upravený odkaz vede do aplikace (na stránku "show.jsp") a jako parametr má původní odkaz, převedený na absolutní adresu. Když tedy při vytváření navigace přijde takovýto požadavek, vytvoří aplikace odpovídající instanci třídy `URLStep` tak, aby s její pomocí bylo možné požadavek zopakovat; takto vytvořený navigační krok aplikace uloží do šablony a dále pokračuje získáním, zpracováním a zobrazením stránky, na kterou požadavek vede.

5.4.2 Formuláře

Aplikace transparentně podporuje webové formuláře tak, aby bylo možné procházet webovými stránkami stejně, jako kdyby spojení probíhalo přímo a nikoli přes server aplikace. To si však nejprve žádá úpravu původního formuláře na webové stránce. Provádím ji společně s úpravou URL ve třídě `linkVisitor` takto:

- do formuláře přidám skryté pole s hodnotou URL původního formuláře (z atributu `action`, pokud je přítomen, jinak se použije adresa stránky)
- do atributu `action` uložím odkaz do aplikace

Zpracování formulářových dat Data vložená při tvorbě navigace do upraveného formuláře se odešlou webovému serveru aplikace. Když pak aplikace přijme formulářová data (GET, nebo POST metodou), uloží je jako seznam (`ArrayList`) objektů třídy `NameValuePair` (tuto třídu používá pro uložení formulářových dat knihovna HTTPClient) do nové instance třídy `FormStep`, která je vytvořena v šabloně. Součástí dalšího požadavku (request) při připojení HTTPClientem jsou pak i tato data. Použije se přitom samozřejmě tatáž HTTP metoda, kterou aplikace data přijala. Odpověď serveru pak tvoří další stránku, která se tak zpracuje a zobrazí.

5.4.3 Navigační kroky

Navigační kroky ukládá aplikace jako instance příslušných potomků třídy `WrapperItem`, tedy položky šablony. Kompletní přehled o třídách použitých při tvorbě šablony získáte z diagramu 5.1.

Třída `Step`

Tato třída je předkem navigačních kroků a obsahuje metody, které musí každý navigační krok implementovat. Jde zejména o metodu `getPage()`, která slouží k získání webové stránky při extrakci.

Třída `URLStep`

Jedná se o základní navigační krok, který vznikl prostým následováním odkazů. Ve skutečnosti však tato třída slouží právě spíše k označení způsobu vzniku kroku, protože obsahem metody `getPage()` je pouze zavolání stejné metody u nové instance třídy `FormStep`, které se nastavilo příslušné URL a prázdné atributy.

Třída `FormStep`

Třída pro uchování požadavku na vyplnění webového formuláře. V metodě `getPage()` používá knihovnu HTTPConnection k odeslání formulářových dat a získání výsledku pro extrakci.

Třída `Robot`

Třída `Robot` není sama o sobě navigačním krokem (není odvozena od třídy `Step`), přesto však patří do části o navigaci. Položku typu Robot je totiž možné vytvořit jen v navigačním módu. Robotu musí předcházet některý z navigačních kroků. Robot označuje místo, kde se při extrakci prohledá aktuální stránka a pro všechny nalezené odkazy se provede zbytek šablony. Je možné nastavit hloubku zanoření a zda se mají následovat odkazy vedoucí na jiné servery.

5.5 Definice dat

Definici dat z pohledu uživatele aplikace jsem popsal v 4.3. V této části se soustředím na to, jak je celá šablona a její struktura navržena. Celý diagram tříd, použitých v šabloně je na obrázku 5.1.

5.5.1 Šablona (třída `Wrapper`)

Šablona (wrapper) je ústřední třída v celé aplikaci – tvoří ji posloupnost instancí potomků třídy `WrapperItem`, což mohou být navigační kroky, robot a vyhledávací vzory. Při procházení stránek v režimu navigace, nebo při definici dat je vždy aktivní právě jedna šablona, do které se úpravy ukládají. Šablona je také jediný celek, pro který lze na uživatelské úrovni spustit extrakci dat.

5.5.2 Položka šablony (třída `WrapperItem`)

Tato abstraktní třída pouze zastřešuje všechny položky, které se mohou vyskytovat v šabloně a sice navigační kroky, roboty a vyhledávací vzory.

5.5.3 Vyhledávací vzor (třída `Pattern`)

Výsledkem tvorby vyhledávacího vzoru v appletu je strom uzlů, který vznikl úpravami původního stromu uzlů celé stránky a označením atributů a proměnných. Z tohoto stromu vytváří třída `Pattern` filtr (viz. 4.7.1), který slouží k nalezení těchto úseků stránky. Jak jsme si ukázali na příkladu 4.4, výsledkem filtru je celý obsah nalezeného tagu. Navíc není možné do filtru uložit definice proměnných. Proto kromě samotného filtru ukládám do třídy `Pattern` také celý tento strom (včetně definic proměnných na příslušných místech) a to v podobě fragmentu HTML kódu. Tento strom nazývám `auxTree`, neboli pomocný strom.

Proměnná (třída `Variable`)

Moje aplikace umožňuje definovat tři typy proměnných (obsah tagu, hodnota atributu, přítomnost atributu). Třída `Variable` slouží k uchování těchto proměnných a při extrakci dat také k uložení nalezených hodnot. Pro definici vyhledávacího vzoru je důležité, že třída `Variable` nabízí metodu `save()`, která vrací definici proměnné v textové podobě v následujícím formátu:

```
n,název proměnné
```

nebo

```
n,název proměnné,jméno atributu
```

Kde `n` je číslo označující typ proměnné. Třída `Variable` nabízí mimo jiné konstruktor, který tento formát převede zpět na instanci proměnné a dále umožňuje uchovat hodnotu proměnné a URL stránky, na které byla nalezena.

Pomocný strom `auxTree`

Pomocný strom je tedy část HTML kódu, který vznikl při vytváření vyhledávacího vzoru. Obsahuje tagy i s atributy, které uživatel zadal, textové uzly se zadaným obsahem a také definice proměnných.

Proměnné jsou v pomocném stromu uloženy ve formě textového zápisu, jak je ukázán v předchozí části. Každý tag, který obsahuje proměnné, má atribut "DIP_variable", ve kterém jsou definice proměnných uloženy v textové podobě, oddělené znakem ";". Definice proměnné v textovém uzlu je uložena přímo jako text tohoto uzlu (což je také důvod, proč není možné mít u textového uzlu současně nastaven text který má obsahovat a definici proměnné).

Ukázka `auxTree`

```
<tr class="our-prize"><td >cena</td>
<td class="prize" DIP\_variable="0,cena;"></td></tr>
```

Vytvoření filtru

Nedílnou součástí vytvoření vyhledávacího vzoru (a instance třídy `Pattern`) je tvorba filtru pro vyhledávání z vytvořeného pomocného stromu. K tomuto účelu slouží metoda `Pattern.createNodeFilter()`.

Celý postup vytvoření filtru je zjednodušeně znázorněn na diagramu 5.2. Metoda `createNodeFilter()` je volána s patrametrem prvního uzlu v pomocném stromu.

V případě, že je uzel tagem, vytvoří se seznam filtrů, do kterého se přidá `TagNameFilter` s parametrem jména tagu a následně také všechny filtry atributů tagu, vytvořené postupným voláním pomocné metody `getAttributesFilter()`, která vytvoří pro atributy filtry `HasAttributeFilter`. Tento seznam se pak spojí filtrem `AndFilter` a uloží jako první část filtru (`first`). Pokud se ale jedná o textový uzel, vytvoří se první část filtru jako `StringFilter` – filtr na přítomnost textového uzlu, pokud uzel v auxTree obsahuje text, je tento text nastaven i filtru.

Druhá část filtru je složena z filtrů potomků (pokud existují). Je vytvořen seznam filtrů, na který se postupně přidávájí filtry, které vrátí rekurzivní volání metody `createNodeFilter()` která je volána pro každého potomka. Získaný filtr potomka je před přidáním do seznamu obalen filtrem `HasChildFilter`. Složením seznamu pomocí filtru `AndFilter` je vytvořena druhá část filtru (`second`).

Nakonec je vrácen filtr složený jako `AndFilter( first, second )`.

5.5.4 Uložení šablony

Vzhledem k tomu, že v průběhu vytváření šablony prochází prohlížeč klienta přes několik stránek, bylo třeba vyřešit průběžné ukládání rozpracované instance šablony. Za tímto účelem jsem vytvořil třídu `StateStorage`, kterou používají jsp stránky aplikace jako session bean. V tomto session beanu je uložen unikátní identifikátor šablony, pod kterým je uložena v databázi na serveru, dále také poslední nalezené kódování a konečně je zde informace o módu aplikace (zda jde o mód navigace, nebo definice dat).

Třída `StateStorage` nabízí dvě statické metody pro ukládání současného a získání předchozího stavu aplikace (včetně rozpracované šablony) a to metody `saveState` a `loadState`. Tyto metody provádí serializaci a deserializaci do/z databáze. Serializace probíhá ve formátu XML pomocí tříd `java.beans.XMLEncoder` a `java.beans.XMLDecoder`. Stejný postup využívám i v appletu pro získání a uložení aktuální šablony.

Třída `StateStorage` přináší i další pomocné metody pro získání všech šablon na serveru a pro odstranění šablony, které využívá hlavní stránka aplikace s přehledem šablon.

5.6 Extrakce dat

Tato část podrobně popisuje průběh extrakce dat z uložené šablony.

5.6.1 Procházení položek šablony

Extrakce dat je zahájena voláním metody `Wrapper.extractData()`. Stejně jako většina metod, které budou následovat, i tato metoda vrací výsedek jako seznam proměnných (`List<Variable>`). Samotné zpracování šablony však začíná až v metodě `extractDataFromWrapperItems()`, které je třeba předat seznam prvků třídy `WrapperItems`. S tímto seznamem metoda pracuje takto:

- Procházíme postupně každou položku seznamu.
- Pokud je položka instancí třídy `Step`, uloží se do dočasné proměnné `lastStep`.
- Pokud je položka vyhledávací vzor (`Pattern`), spustí extrakci proměnných (metoda `Pattern.extract`) viz. 5.6.2.

- Pokud je položka typu `Robot`, extrahuje odkazy z poslední stránky pomocí metody `extractURLs` a pracuje následovně: Pro každý nalezený odkaz vytvoří novou instanci `URLStep`. Pokud je hloubka robota větší, než 0, vytvoří kopii robota s hloubkou zmenšenou o 1. Vytvoří nový seznam `List<WrapperItems>` do kterého přidá nově vytvořené položky URLStep a Robot a pokračuje zkopírováním zbývajících položek původního seznamu.

  Tento seznam předá jako parametr do rekurzivního volání metody `extractDataFromWrapperItems()` a vrácené proměnné přidá k seznamu proměnných.

- Vrátí výsledný seznam proměnných.

5.6.2 Zpracování vzoru `Pattern.extract()`

Metoda `Pattern.extract()` očekává jako parametr navigační krok (třídy `Step`). Volá totiž jeho metodu `getPage()`, výsledek předloží parseru a provede filtr, uložený ve vzoru. Protože, jak jsme si již ukázali na příkladu 4.4, výsledkem filtru může být více instancí vyhledávacího vzoru a tyto obsahují i uzly, které v původním filtru nebyly, je třeba ještě načíst pomocný strom auxTree (parsuje se, stejně jako výsledek filtru) a porovnat jej s každou nalezenou instancí vzoru. Toto porovnání zajišťuje metoda `Pattern.matchAndExtract()`, která očekává dva parametry typu uzel (`Node`) na prvním místě uzel z pomocného stromu, na druhém místě uzel z instance filtru. Použijeme tedy kořenový uzel auxTree a kořenový uzel každé instance vzoru.

Metoda `Pattern.matchAndExtract(auxNode, extractedNode)` Postup extrakce je zjednodušeně znázorněn na diagramu 5.3. Poznámky k algoritmu:

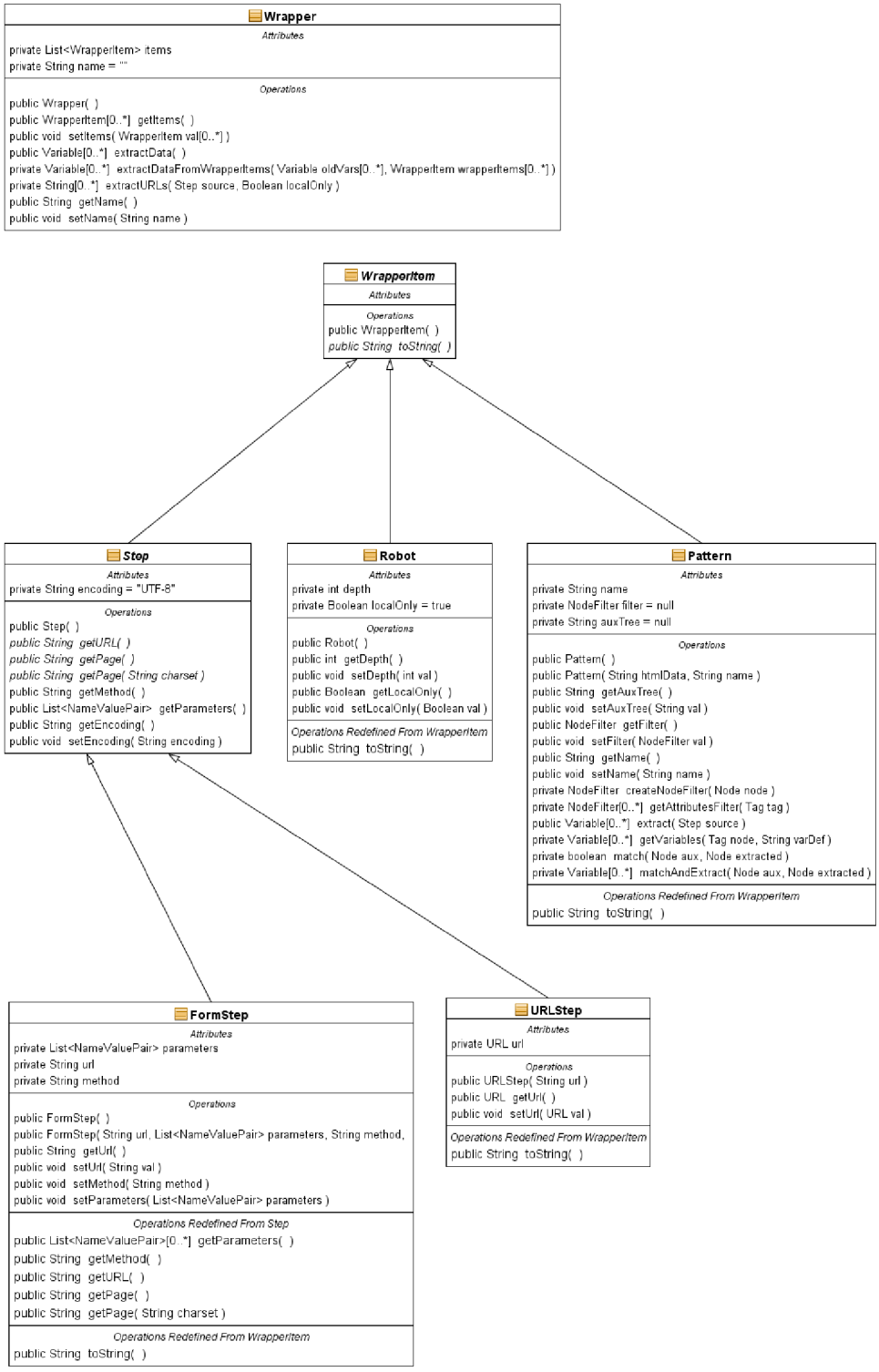
- Na začátku hledám uzly stejného typu (oba tagy, nebo oba textové uzly; u textu to navíc znamená, že `extractedNode` obsahuje jako podřetězec obsah auxNode). Pokud uzly nesouhlasí, přeskakuji na další sourozence uzlu `extractedNode` (protože výsledek filtru může obsahovat sourozence, které do filtru nepatří).

- Metoda `getVariables()`, použitá pro získání hodnot proměnných pracuje tak, že atribut "DIP_variable", který obsahuje textové reprezentace proměnných, rozparsuje na jednotlivé proměnné a ty pak vyhodnotí podle jejich typu.

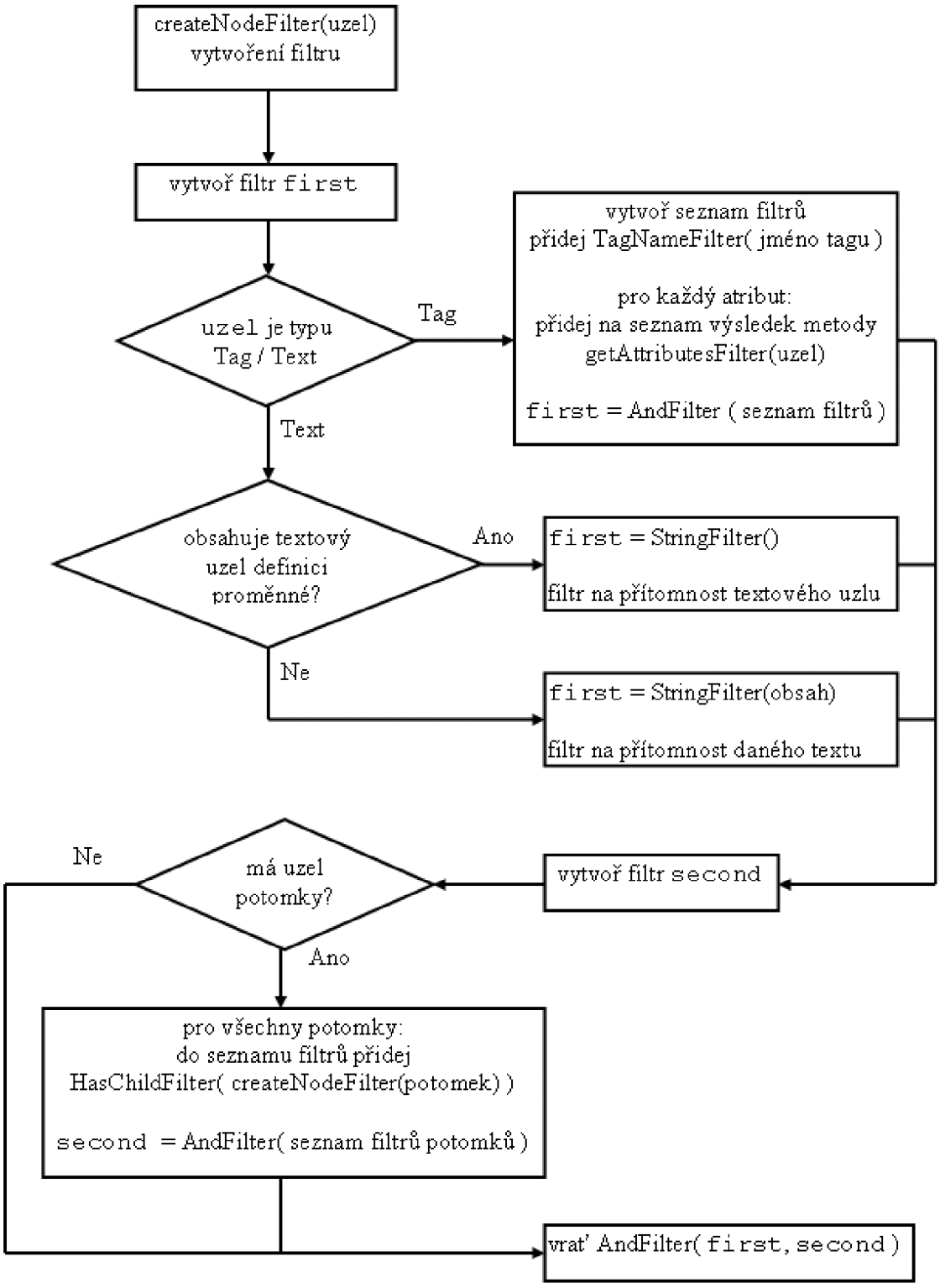
- Při procházení potomků se rekurzivně volá extrakce vždy na dvojici uzlů, která je na stejné pozici v obou stromech.

5.6.3 Výstup proměnných ve formátu XML

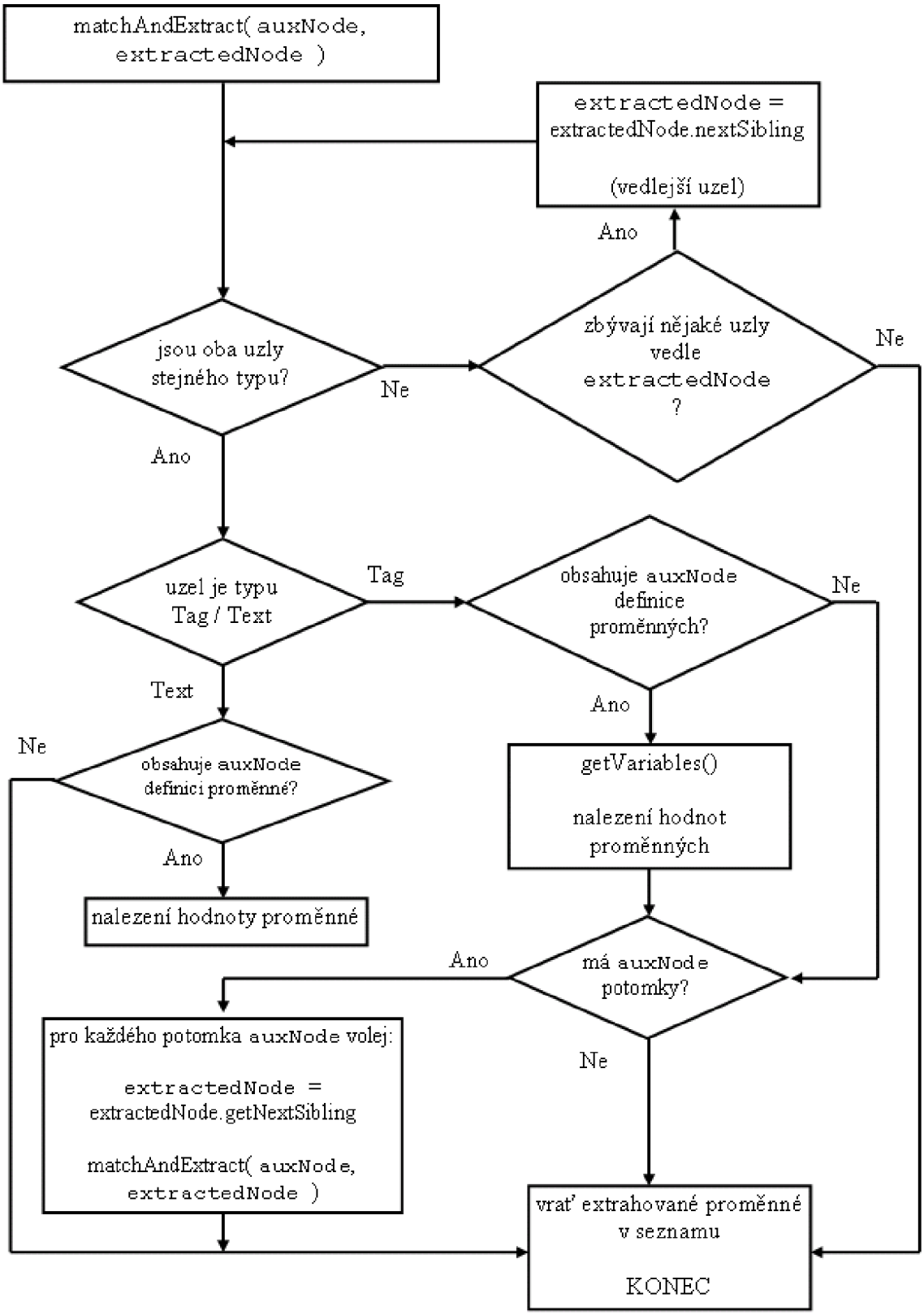
Po dokončení extrakce je výsledkem seznam proměnných, kterým byla přiřazena hodnota, ale také URL zdrojové stránky a název vyhledávacího vzoru, ze kterého proměnná pochází. Stránka `showWrapper.jsp`, která spouští extrakci na uživatelské úrovni (konkrétní šablona lze zvolit parametrem `id` v adrese) seřadí získané proměnné podle URL, názvu vzoru a názvu proměnné a v XML formátu zobrazí – viz. 5.4.



Obrázek 5.1: Diagram tříd pro uložení šablon



Obrázek 5.2: Diagram postupu vytvoření filtru metodou createNodeFilter()



Obrázek 5.3: Diagram postupu extrakce dat v metodě matchAndExtract()

Obrázek 5.4: Ukázka výsledku extrakce dat, zobrazených v XML

Kapitola 6

Pokyny pro překlad a nasazení aplikace

6.1 Prostředí a knihovny

Na přiloženém CD je aplikace ve formě projektu pro NetBeans verze 6.5. Projekt obsahuje všechny soubory a knihovny potřebné pro překlad. Knihovny HTTPClient a HTML Parser jsou přiloženy i ve formě kompletních distribucí.

Databáze Můj projekt využívá databázi MySQL 5 přes JDBC ovladač. SQL skript pro vytvoření použité databáze je na přiloženém CD. Pokud by bylo nutné upravit vlastnosti připojení, je to možné v metodě `getConnection()` třídy StateStorage.

6.2 Podepsání appletů

Oba applety je třeba podepsat – děje se tak automaticky při překladu pomocí ant skriptu. Je však nutné nejprve vytvořit vlastní klíče a hesla – vše je pak možné změnit v souboru `build.xml` u obou appletů.

Kapitola 7

Závěr

Analyzoval jsem možnosti a nedostatky bakalářské práce, na kterou moje diplomová práce volně navazuje. Prostudoval jsem i Lixto – komerční nástroj pro extrakci dat z webových stránek. Výsledkem srovnání byla představa o možnostech mého projektu.

Dále jsem popsal problémy, které bylo třeba při vývoji projektu vyřešit a představil způsob jejich řešení. Konkrétně šlo o získání žádané stránky (včetně možnosti odeslání vyplněného webového formuláře), definici sledovaných údajů na stránce, nalezení těchto údajů (na konkrétní stránce, nebo na větším množství stránek) a jejich zobrazení ve formátu XML.

Výsledkem práce je webová aplikace, která na vyžádání zobrazí extrahovaná data ve formátu XML, který je vhodný pro další strojové zpracování.

Testování na reálných stránkách ukázalo, že při vhodně vytvořených vyhledávacích vzorech je možné dosahovat očekávaných výsledků, nicméně některé stránky po zpracování parserem nefungují tak, jako při běžném prohlížení – jde zejména o ty weby, které velkou měrou spoléhají na JavaScript. V některých případech se také stále objevují problémy s kódováním diakritiky. A konečně testy s automatickým hledáním odkazů (”Robot”) ukázaly na poměrně nízkou rychlost aplikace, která může být částečně způsobena provozem na serveru určeném pro vývoj.

Pokračování vývoje tedy vidím v optimalizaci výkonu, zlepšení práce s kódováním a JavaScriptem na webových stránkách. Vzhledem k současným trendům by také bylo vhodné zaměřit se kromě zpracování běžných webových formulářů také na asynchronní komunikaci s webovým serverem (technologie AJAX).

Literatura

[1] Tomáš Andrýsek. Komunikace java appletu s html stránkou. http://interval.cz/clanky/komunikace-java-appletu-s-html-strankou/. [cit. 2009-01-24].

[2] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual web information extraction with lixto. In *The VLDB Journal*, pages 119–128, 2001.

[3] The Internet Society (1999) R. Fielding. Hypertext transfer protocol – http/1.1. http://tools.ietf.org/html/rfc2616. [cit. 2007-12-19].

[4] The Internet Corporation for Assigned Names and Numbers. Mime media types. http://www.iana.org/assignments/media-types/. [cit. 2008-01-02].

[5] Apache Software Foundation. Http client features. http://hc.apache.org/httpclient-3.x/features.html. [cit. 2009-01-24].

[6] Lixto Software GmbH. Lixto transformation server overview. http://www.lixto.com/lixto_transformation_server. [cit. 2007-12-27].

[7] Lixto Software GmbH. Lixto visual developer overview. http://www.lixto.com/lixto_visual_developer. [cit. 2007-12-27].

[8] Lixto Software GmbH. Lixto visual developer webcast. http://www.lixto.com/flash/vd_en/vd_en.html. [cit. 2007-12-27].

[9] Petr Puna. *Osobní internetový vyhledávač, bakalářská práce*. FIT VUT v Brně, 2006.

[10] Wikipedia the free encyclopedia. Html. http://en.wikipedia.org/wiki/HTML. [cit. 2006-04-11].

[11] Wikipedia the free encyclopedia. Javascript. http://en.wikipedia.org/wiki/JavaScript. [cit. 2006-04-21].

[12] Wikipedia the free encyclopedia. List of xml and html character entity references. http://en.wikipedia.org/wiki/List_of_XML_and_HTML_character_entity_references. [cit. 2006-04-14].

[13] Wikipedia the free encyclopedia. Standard generalized markup language. http://en.wikipedia.org/wiki/Sgml. [cit. 2006-04-11].

[14] Wikipedia the free encyclopedia. Xml. http://en.wikipedia.org/wiki/XML. [cit. 2006-04-13].

[15] L. Masinter The Internet Society (2005) T. Berners-Lee, R. Fielding. Uniform resource identifier (uri): Generic syntax. http://tools.ietf.org/html/rfc3986. [cit. 2007-12-19].

[16] W3C. Document object model (dom). http://www.w3.org/DOM/. [cit. 2008-01-02].

[17] W3C. Extensible markup language (xml) 1.0 (third edition). http://www.w3.org/TR/2004/REC-xml-20040204/. [cit. 2006-04-24].

[18] W3C. Html 4.01 specification. http://www.w3.org/TR/html401/. [cit. 2006-04-11].

[19] W3C. Xhtml 1.1 – module-based xhtml. http://www.w3.org/TR/xhtml11/. [cit. 2006-04-24].