VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

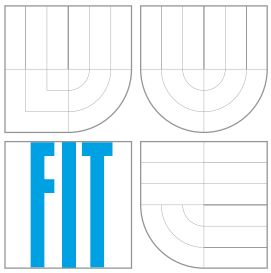
AUTOMATIZOVANÁ ANALÝZA WWW STRÁNEK

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE NIKITA VAŇKŮ
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

AUTOMATIZOVANÁ ANALÝZA WWW STRÁNEK

AUTOMATED WEB PAGE ANALYSIS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — NIKITA VAŇKŮ
AUTHOR

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

BRNO 2015

Abstrakt

Cílem této práce je vytvoření aplikace na analýzu WWW stránek. K implementaci aplikace byl zvolen jazyk Java, konkrétně framework JavaFX s použitím grafové databáze OrientDB. Vytvořená aplikace dokáže analyzovat WWW stránky menšího rozsahu. Aplikace je užitečná jako nástroj k sestavení struktury WWW stránek.

Abstract

Výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Webový pavouk, HTML, analýza webových stránek, Jsoup

Keywords

Web crawling, HTML, web page analysis, Jsoup

Citace

Nikita Vaňků: Automatizovaná analýza WWW stránek, bakalářská práce, Brno, FIT VUT v Brně, 2015

Automatizovaná analýza WWW stránek

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana ...

.....
Nikita Vaňků
13. května 2015

Poděkování

Velice děkuji mému vedoucímu práce Ing. Radku Burgetovi, Ph.D za motivaci, vedení a mou podporu. Děkuji také Mgr. Františu Trosteru za nápad na tento nástroj.

Obsah

Kapitola 1

Úvod

World Wide Web (dále jen WWW) dokumenty jsou stále více využívanějšími sdělovacími prostředky. Jejich počet neustále roste [zdroj]. Využití WWW dokumentů se nachází. WWW dokumenty se využívají pro širokou škálu účelů. Můžeme narazit na dokumenty se spíše statickým obsahem jako jsou blogy, firemní reprezentace atd. Můžeme ale také narazit na dokumenty s kompletně dynamicky generovaným obsahem jako jsou např. internetové vyhledávače. informační systémy, webové aplikace aj.

Množství webových stránek na doméně je téměř neomezený. Na internetu lze najít domény o několika stránek, ale i domény jejichž obsah roste po více jak deset let a obsahují stovky až tisíce. Takové domény pak cyklicky procházejí refactoringem kódu či celého systému. Může se pak stát, že se řada stránek zapomene či vytratí. Či naopak, že systém roste a nalepují se na něj další a další části, mnohdy zbytečně. Stejně tak se staré neodstraní, třeba i z důvodu, že v programátorském týmu "nikdo nemá tušení co to dělá a co se stane, pokud se to dá pryč".

Z tohoto a dalších důvodů využívají týmy mnoho nástrojů, které analyzují zpracování požadavku serveru. Jedná se o měření rychlosti zpracování backendu a frontendu. Velikost stahovaných klientských zdrojů jako jsou například soubory kaskádových stylů, obrázky či Javascriptové knihovny. Jednim z těchto nástrojů jsou Webové crawlery, neboli pavouci.

Pavouci našli největší využití hlavně u internetových vyhledavačů. Jsou navrženi tak taby v krátkém čase dokázali zpracovávat najednou velké množství stránek z různých domén. Při zpracovávání hledá odkazy na další stránky a ty také analyzuje. Vzhledem k velikosti a neústalého růsta pavouci mnohdy svou práci nikdy nekončí, nebo jsou spouštěni pravidelně. Úloha pavouků může být různá.

Mým cílem je vytvořit pavouka, který bude analyzovat pouze jednu doménu. Předpokládá se, tedy že bude zpracovávat pouhé stovky až tisíce stránek. Výstupem analýzy mají být programátorům užitečná data jako grafické zobrazení provázanosti stránek. Měření rychlosti a analýza stahování stránek. Rendering stránek. Součástí analýzy je možno také sledování chyb Javascriptu či validace DOM objektu.

V této práci se budu věnovat několika hlavních částí této problematiky. První část bude věnována analýze problematiky pavouků. V druhé části zpracuji návrh vlastního pavouka. Ve třetí části budu řešit implementaci vyhotoveného návrhu a poslední část je věnována experimentům s pavoukem a výsledky jeho analýz.

Kapitola 2

Teorie

Tato kapitola je věnována teoritickém zpracováním problematiky analýzy webových stránek. Web lze reprezentovat jako graf, ve kterém uzly jsou dokumenty identifikované URL a hrany hypertextové odkazy. Cílem pavouka je projít podgraf tohoto grafu vymezený počátačním uzlem a dalšími omezeníny jako specifikace povolených domén. Robot při procházení může provést jednoduchou analýzu dokumentu jako zaznamenání návratového kodu serveru, extrakce hypertextových odkazů, vyhledání formulářů apod.

2.1 Současní pavouci

Dnes se pavouci využívají převážně k indexování internetu pro internetové vyhledávače. Mezi ty nejvíce známe můžeme zařadit Google či Bing. Detaily implementace těchto pavouků jsou pochopitelně utajeny. Kromě pavouků analyzující celý internet bez omezení existuje i celá řada úzce specializovaných pavouků pro určitou činnost. Například `skipfish`[1] je Open Source pavouk testující zabezpečení domény a odhaluje slabá místa, které by mohli využít útočníci. `HTTrack`[2] je pavouk, který po zadání adresy prochází veškerý její obsah a stahuje ho do uložiště. Stránky lze potom prohlížet Offline.

Pavouci ve své povaze přistupují k webovým serverům v ohromně větším měřítkům než obyčejný uživatel. Toto může znamenat, že server klasifikuje pavouk jako nebezpečného robota či snad pokus o DDos útok a zakáže pavoukův přístup k datům serveru. Tento případ může nastat i v naší aplikaci. Bohužel každý webový server může nastavit tento limit na jiné úrovni a každý se také v případě překročení limitu může zachovat jinak. Některé servery se mohou bránit snížením priority odpovědí, jiné neodpovídáním úplně.

2.2 Analýza URL

Uniform Resource Locator (dále jen URL, neboli též adresa) jsou definovány v `RFC 3986`[1]. Je potřeba rozlišit relativní a absolutní adresu. V případě relativní adresy se zohledňuje aktualní adresa dokumentu. Součástí URL může být též kotva, kotva je definována atributem `id` v elementu `<a>`, kotvy nemají pro pavouka žádný význam a jsou tedy ignorovány. V případě nalezení většího množství odkazů se stejnou adresou v dokumentu, pavouk se zachová jako by pracoval s adresou jedinou.

[1] https://code.google.com/p/skipfish/
[2] https://www.httrack.com/

Pavouk by také měl být připraven čelit generovaným hypertexovým odkazům, jako jsou stránky s dynamickým obsahem, které se mohou jevit z pohledu pavouka jako nekonečné.[2]

Velkým oříškem mohou být též `GET` parametry, které jsou součástí URL. Tyto parametry mají nejednoznačný význam a pavouk nemůže vědět, zda-li jim má věnovat pozornost. GET parametr může i nemusí mít žádnou váhu na vykreslení dokumentu. Mohou existovat i stránky obsahující odkazy s velkým množstvím opakujících se GET parametrů. Pavouk by měl být schopen tyto odkazy normalizovat a tím nevidět rozdíl mezi odkazem

`http://www.domain.com/index.php?param1=1&param2=2`
a odkazem
`htpp://www.domain.com/index.php?param2=2&param1=1`

2.3 Extrakce URL

Extrakce adres je možné analýzou HTML webových dokumentů. Prozkoumáním vybraných elementů, můžeme získat potřebné odkazy. Mezi zmiňované elementy patří[3].

- `<a></a>` atribut `href`
- `<form></form>` atribut `action`
- `<link>` atribut `href`
- `<script></script>` atribut `src`
- `<style></style>` atribut `src`

V závislosti na povaze atributu je možno předpovídat na jaký mime typ webového objektu odkaz ukazuje. Jestliže extrahujeme atributy src z tagu `script` a `style`, můžeme si být jisti že mime type bude text/css či text/javascript. Takto můžeme konkrétizovat zpracování odkazů, jelikož odkazy v těchto souborech hledat nebude.e

2.4 Zaznamenání časových událostí

Pro uživatele je důležitý co nejkratší čas mezi odesláním požadavku na server a zobrazení stránky. Tento čas je možné rozdělit na dvě hlavní části a to:

- čas stráveny zpracováním požadavku na serveru - `Backend`
- čas strávený stažení a zobrazení zdrojových dokumentů - `Frontend`

Webová stránka, která se zobrazí uživateli v prohlížeči je mnohdy složena z více souborů. Jedná se jednak o HTML/XML dokument, ale také o obrázky, Kaskádove styly, Javascript a další soubory. Moderní prohlížeče dnes dokáží stahovat více souborů z jednoho serveru najednou aby čas strávený stahováním co nejvíce urychlily. Množství vláken se však u každého prohlížeče liší a uživatel může s touto hodnotou manipulovat. (zdroj)

Naměřené časy musíme také považovat za nepříliš průkazné. Server může odpovídat rychleji či pomaleji v závislosti na současném zatížení uživatelskými požadavky. Průkazné měření by se dalo dosáhnout pomocí pravidelného a dlouhodobého měření

[3] Je potřeba brát v potaz, že URL mohou být v dokumentu vloženy také jako pouhý text či při zpracování události javascriptem. Tyto odkazy pavouk pochopitelně nenajde.

Kapitola 3

Návrh aplikace

Aplikace je složena z několika částí. Jedná se o algoritmus pavouka, který stahuje a analyzuje dokumenty. Databázovým uložištěm, které uchovává analyzované data a grafická nadstavba (GUI) přes kterou je aplikace řízena. Pro vykreslování dokumentů jsou využity technologie Headless Browseru, v tomto případě byl využit PhantomJS, který pracuje na vykreslovacím jádře WebKit.

3.1 Java

Byla využita platforma Java s platformou JavaFX, která umožňuje tvorbu Rich Internet Applications[5]. JavaFX aplikace je možno využít i jako desktopové aplikace. Aplikace vytvořena součástí této práce byla naprogramována pro Java 8, JavaFX 8.

3.2 Modifikace prohledávaného prostoru

Cílem aplikaci je analyzovat pouze jedinou doménu. Při prohledávání je tedy nutno zamezit zabloudění pavouka na cizí domény a prohledávání zcela jiné části internetu. Toto je dosaženo kontrolou domény nejvyššího soukromého řádu společně s veřejným suffixem. Pokud se doména tohoto řádu nebo suffix liší, adresa se neprohledává.

Je třeba také počítat s tím, že množství dokumentů a odkazů na doméně není předem známo. Doména může obsahovat několik desítek dokumentů, ale i několik stovek. Aplikace musí být na tuto skutečnost připravena a v případě procházení takto velké domény zareagovat. Toto je uskutečněno stanovením maximální hloubky prohledávání. Před prohledáním prvního dokumentu je aktualní hloubka 0 a každým dalším dokumentem se hodnota zvýší o 1. V případě dosažení maximální hloubky prohledávání se z tohoto dokumentu neextraktují další adresy.

Množství webových stránek obsahuje mnohdy část s dynamicky generovaným obsahem, jehož analýza by spotřebovala příliš velké množství prostředků, či nás tato část nezajímá. Může se jednat o blogy, fora, či jiné aplikace kde obsah tvoří převážně uživatelé stránek. V opačném případě mohou být webové stránky přítomny na několika domén najednou. Z tohoto důvodu se prohledávání řídí taky podle uživatelem definovaných pravidel. Pravidlo, které zamezí prohledávání části domény (subdomény) a omezí tak prohledávaný prostor a pravidlo, které naopak povolí prohledávání na dalších doménách a zvětší tím prohledávaný prostor.

3.3 Analýza DOM objektu

Extrakce adres z HTML dokumentů je možné za pomocí prohledání Document Object Modelu (dále jen DOM). V objektu se hledají elementy a atributy zmíněné v 2.3. Vytažené adresy se musí dále zpracovat jak bylo popsáno v 2.2, k adrese se také přídá záznam na kterém dokumentu byl nalezen a následně se vloží do fronty ke zpracování.

Pro práci s DOM objektem a extrakci adres je využit HTML parser `jsoup`. Jedna z vlastností tohoto parseru je umět se vypořádat s nevalidním dokumentem. Chybným použitím tagu apod. [3]

3.4 Uložení dat

Jednou z hlavních otázek bylo kam uložit data vytvořená analýzou. Při přípravě aplikace jsem tuto otázku hluboce podcenila, první zkušební prototypy ukládaly všechny data do RAM paměti. I u velice malých webových stránek se vytvořilo velké množství objektů. Takové velké množství neustále uložených objektů způsobilo naalokování několik GiB paměti a následného pádu aplikace.

Rozhodla jsem se využít databázi pro ukládání většinu dat.

3.4.1 Databáze

Jako databáze byla zvolena `OrientDB 2.0.X Community Eddition`[1], jedná se o hybridní NoSQL databáze umožňujíci pracovat jak v Dokument (`MongoDB`) tak Grafovém (`Neo4j`) modu.

Aby aplikace nemusela skladovat v paměti příliš velké množství dat, data se co nejdříve po zpracování přesunou do databáze. Jelikož zpracované data ve své podstatě vytváří orientovaný graf webových dokumentů, byla pro tuto potřebu zvolena grafová databáze. Způsob uložení dat v databázi tedy co nejvíce připomíná způsobu uložení dat na doméně.

Grafová databáze je sestavena stejně jako graf ze dvou základních stavebních prvků. Vrcholy a hrany mezi vrcholy. Jako vrcholy si můžeme například představi dokumenty. Hrany by poté mohly být odkazy mezi těmito dokumenty. V sekci Implementace databáze 4.2 je věnovaná část návrhu databáze a její propojení s aplikací.

3.4.2 Souborový systém

Pokud se s každým webovým objektem do databáze uloží i identifikátor, je pak možno tento identifikátor využít k sestavení struktury adresářů na souborovém systému. Do těchto adresářů by se ukládala data, které by jednoduše nebylo praktické ukládat do databáze. Jedná se o vykreslené webové snímky a další příliš velká data.

Je nutno také podotknout, že takto uložená data na více místě ztrácí integritu. Databáze může být přítomná na vzdáleném serveru a stačí nám znát pouze adresu a přístupové údaje ke zpřístupnění jejich dat. V případě souborového systému musí být tento systém vždy nějakým způsobem připojen k systému na kterém běží aplikace.

[1] V průběhu vývoje aplikace byla verze několikrát aktualizovaná

3.5 Vykreslení dokumentu

Pro vykreslování webových dokumentu byl využit Headless browser `PhantomJS 2.0`. PhantomJS obsahuje vykreslovací jádro WebKit, lze s ním tedy emulovat vykreslování podobné prohlížečům se stejným jádrem jako je např Google Chrome, Safari či Opera [4].

PhantomJS je plně ovladatelný Javascriptem. Program se spustí s parametrem cesty k javascriptovému souboru, který obsahuje ovládací skript prohlížeče. Prohlížeč vykoná instrukce v tomto skriptu a poté se ukončí. Aplikace spouští prohlížeč s jednoduchým skriptem, který přistoupí na adresu, obsah vykreslí do souboru a poté se ukončí.

3.6 Projekty

Aplikace by měla být znovupoužitelná pro větší počet analýz domén a jejich procházení. Proto jsem se rozhodla analýzy rozdělit na tzv. projekty. Jeden projekt je analýza jedné domény. Projekt je možné vytvořit, spustit analýzu, prohlížet výsledky a smazat.

Projekt je primárně uložen v databázi s většinou dat. Data, nepříliš vhodné uložit do databáze jsou uložena v souborovém systému, projekt by měl být schopen je nalézt.

3.7 Návrh uživatelského rozhraní

Při návrhu jsem se pokusila klást důraz na jednoduché ovládání aby program byl schopen ovládat i mirné pokročilý uživatel internetu.

Uživatel by měl být schopen v aplikaci plně pracovat s projekty. Tedy měl by mít možnost vytvořit nový projekt, nebo otevřít již vytvořený. Spustit analýzu domény či smazat již hotovou analýzu a hlavně procházet výsledky analýzy. Toto se ukázalo jako nejsložitější problém, jelikož uživatel by měl intuitivně procházet grafem. Zvolila jsem způsob zobrazení právě procházeného uzlu a jeho nejbližší okolí v orientovaném grafu. Po poklikání na kterýkoliv vrchol v zobrazení se tento vrchol a jeho nejbližší okolí otevře.

Kapitola 4

Implementace aplikace

Tato kapitola je věnována implementace návrhu v předchozí kapitole a jak se záverečná implementace oproti návrhu změnila.

Aplikace JavaFX podléhají MVC návrhu tedy `Model-View-Controller`. Jádro aplikace, tedy datový model je oddělen od zobrazovací vrstvy. S datovým modelem přímo komunikuje Controller, tedy řídící prvek. Řídící prvek komunikuje taktéž s View, tedy pohledem, přes množinu komponent, která je v pohledu definovaná. Řídící prvek sleduje reaguje na zprávy zaslané pohledem, které zpracovává a přeposílá dále. Pokud se jedná o vlastnosti či parametry modelu, řídící prvek je může pevně obou či jednostraně svázat s komponentou. Např. vlastnost modelu datového typu String může být svázáná s komponentou text v pohledu. Při jakékoliv změny této vlastnosti v modelu, je komponenta aktualizována.

V datovém modelu je obsažena hlavní funkcionalita aplikace. Jádro celé aplikace zastupuje třída `Crawler`. Crawler je spuštěn ve stejném vlákně jako řídící prvky a pohledy. Funguje jako hlavní řídící jednotka. Pokud uživatel zadá příkaz k modifikaci projektu či spuštění analýzy Crawler přepošle příkaz službám v samostatných vláknech, nebo vytvoří dedikované vlákno pro splnění úlohy. Toto se provádí zejména při zpracovávání složitých příkazů, které by činily uživatelské rozhraní neresponzivní.

Mezi služby, které spravuje třída crawler patří sběrač odkazů (sestavení grafu) a post analýza (vykreslování stránek obsahující DOM objekt).

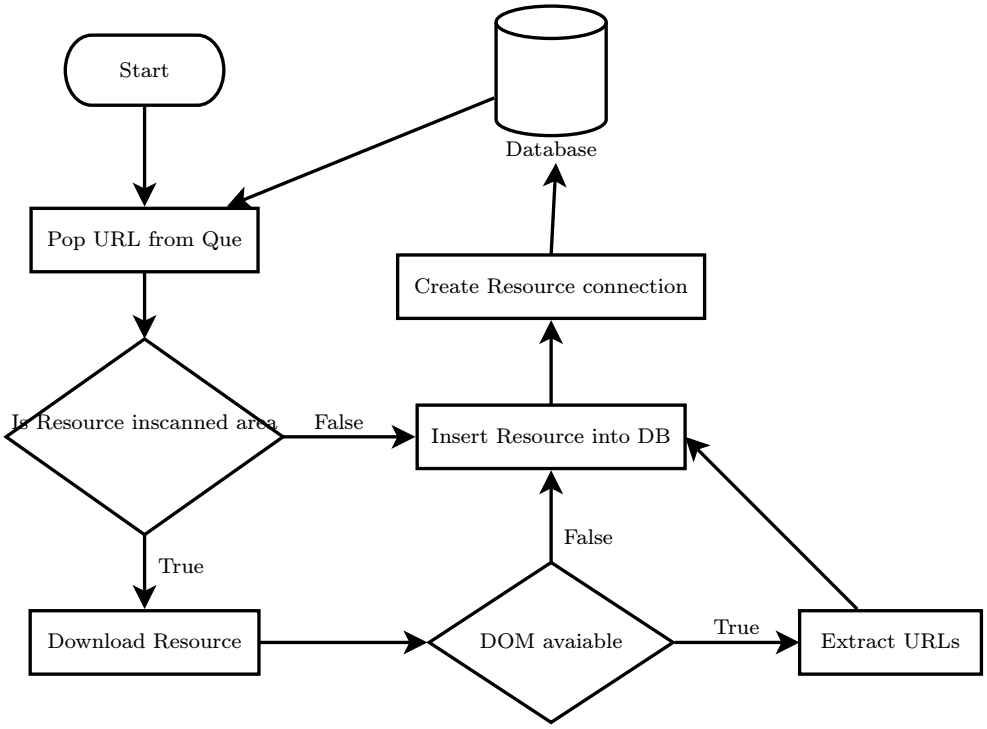
4.1 Analýza domény

Služby, které zprostředkovávají analýzu domény s hlavní třídou komunikují pouze omezenou množinou zpráv oznamující úspěch či neúspěch analýzy. Zpracovaná data jsou předána prostřednictvím databáze či souborového systému.

4.1.1 První část - tvorba grafu

Služba, sestavující graf webových objektu pracuje jako fronta odkazu. Z fronty služba odebere odkaz, u kterého nejdříve zjistí, zda-li se odkaz nachází v prohledávaném prostoru. Pokud je objekt mimo prohledaný prostor do databáze se uloží objekt s minimálním množstvím informací. V opačném případě je objekt stažen a prozkoumám nástrojem `jsoup`. Nástrojem jsou prozkoumány tagy a jejich atributy zmíněne v sekci 2.3, veškeré nalezené odkazy jsou vloženy do zpět do fronty. Jestli-že se jedná o objekt bez DOM části je objekt považován za soubor a do databáze vložen jeho záznam. Po uložení objektu je do databáze vložen také odkaz samotný v podobě hrany. Služba si také udržuje v paměti objekty, které

již jednou zpracovala. Nedochází tak k opakovanému stahování a zpracování již hotových objektů.



Obrázek 4.1: Proces zpracování odkazu

4.1.2 Druhá část - (post processing)

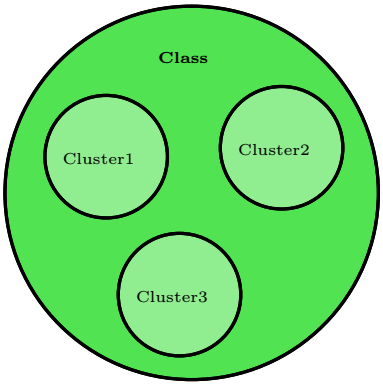
V čase spuštění druhé části analýzy je v databázi přítomen již hotový graf objektu s jejich základními informacemi. Služba z databáze extrahuje objekty, které obsahují DOM (část) a pro každý z těchto objektů nejdříve vytvoří své místo ve stromu souborového systému, kam bude služba později ukládat informace o objektu. Cesta k tomuto adresáři se skládá z názvu projektu a specialního identifáktoru přiřazeného Orient databází. Po dokončení přípravy započne spouštění externího webového prohlížeče `PhantomJS` se skriptem, který webovou stránku vykreslí do obrázku a zároveň zaznamená Javascriptový konzolový výstup při zpracování stránky do soubory. V konzolovém výstupu se často objevují chyby při vykreslování stránek.

4.2 Databáze

Orient Databáze může pracovat s objekty ve dvou modifikacích. Jako dokumenty a jako graf. Každý z těchto dvou modu se hodí pro jinou aplikaci. Jak již bylo zmíněno v sekci 3.4.1, pro ukládání webových objektu je přirozenější použít mod grafu. Naopak pro práci s frontou zmíněnou v sekci 4.1.1 je výhodnější pracovat s dokumenty. Aplikace je propojena s databází přes nativní Java konektor. Konektor je schopen pracovat v těchto dvou modech. Nabízí pro to řadu konstrukcí a metod. Orient databáze umožňuje taky použít SQL jazyk, tento jazyk je narozdíl od např. Oracle SQL jazyku značně omezený, ale obsahuje řadu základních agregačních metod a klauzulí, které jsou pro aplikaci dostačující.

Orient Databáze nezná pojem tabulka, místo ní zavádí pojem třída. Třída mají schopnost dědičnosti. Pokud chce uživatel vytvořit třídu, která bude uzlem nebo hranou v grafu, vytvoří která bude dědit vlastnosti abstraktních tříd V či E. Tyto třídy jsou již v databázi přítomny po jejím vytvoření. V případě, že uživatel vytvoří třídu, která nebude dědit vlastnosti z jedné z těchto tříd, bude tato třída chápana jako dokument.

Orient Databáze ukládá záznamy tříd do tzv. clusteru. Každá třída má alespoň jeden cluster do kterého ukládá své data. Clustery jsou od sebe fyzicky odděleny. Clusteru může být více, vkládání do nich může být prováděno přímo či na základě algoritmu. Clustery jsou od sebe fyzicky odděleny, tato vlastnost se využívá obzvlášť pokud databáze běží v redistribučním modu na několika serverových jednotkách [6]. Spoustu databázi trpí při neustále rostoucím objemu dat, složité dotazy v takovém případě trvají mnohem déle než by trvaly v malých databázích. Přitom postupem času se v databázi začnou objevovat data, ke kterým se bude přistupovat méně či téměř vbůec. Pokud se takové data přesunou do jiných clusterů a databáze pracuje s jednim clusterem, které má omezené množství dat, lze takto rychlost dostazů podstatně navýšit.



Obrázek 4.2: Databázový cluster

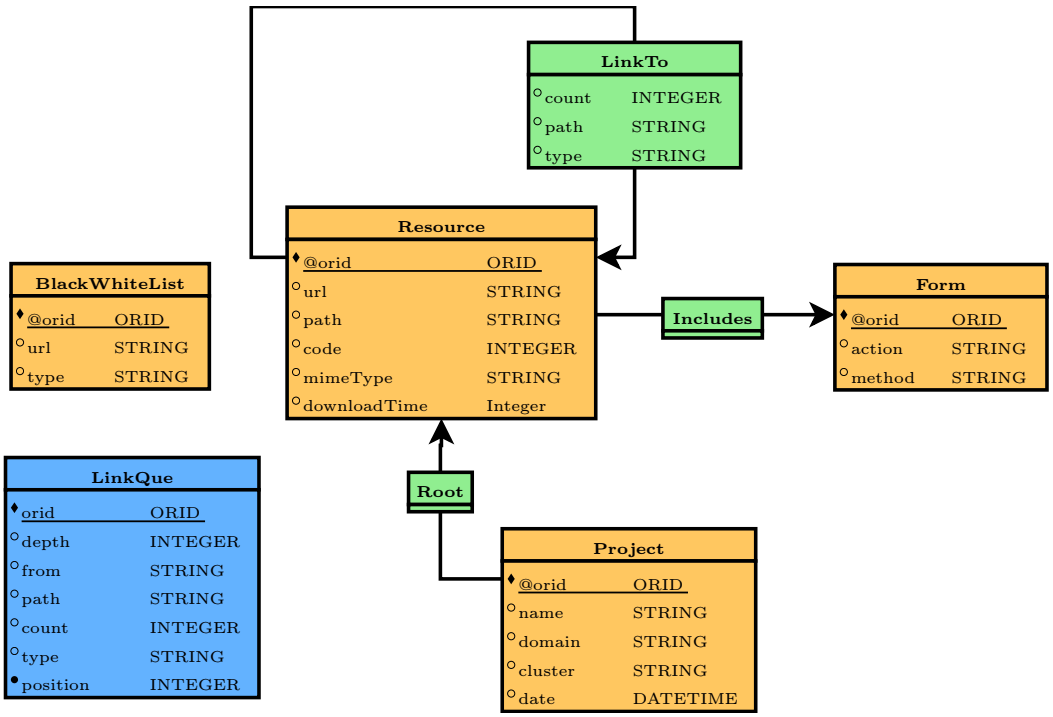
Data, které aplikace ukládá do databáze jsou v podstatě rozdělená do projektů jak již bylo zmíněno v sekci 3.6. Pokud každý projekt bude mít své clustery, které se vytvoří všem třídám se kterými projekt pracuje lze tak spolehlivě oddělit data každé projektu a vyřešit tak problém, který by způsoboval rostoucí dobu vykonávaní dotazů s větším množstvím již analyzovaných projektů.

4.2.1 Schema databáze

V tabulce 5.1 a diagramu 4.3 je znázorňeno jak vypadá finální schema databáze. Projektové clustery se tvoří v třídách `Resource, Form, BlackWhitelist, LinkTo, Includes, LinkQue`. Název těchto clusteru se sestává spojením názvu třídy a unikátnímu identifikátorů, který je uložen jako vlastnost `cluster` ve třídě `Project`.

Název	Typ	Popis
`Resource`	Vrchol	Webový objekt
`Form`	Vrchol	Formulář
`Project`	Vrchol	Projekt
`LinkTo`	Hrana	Odkaz mezi dokumenty
`Includes`	Hrana	Připojení formuláře k dokumentu
`Root`	Hrana	Odkaz na kořenový dokument prohledávané domény
`BlackWhitelist`	Vrchol	Omezení prohledávaného prostoru
`LinkQue`	Dokument	Fronta pro skladování odkazů čekající na zpracování
`LinkSet`	Dokument	Mapa navštívených objektů

Tabulka 4.1: Třídy databáze



Obrázek 4.3: Diagram databáze

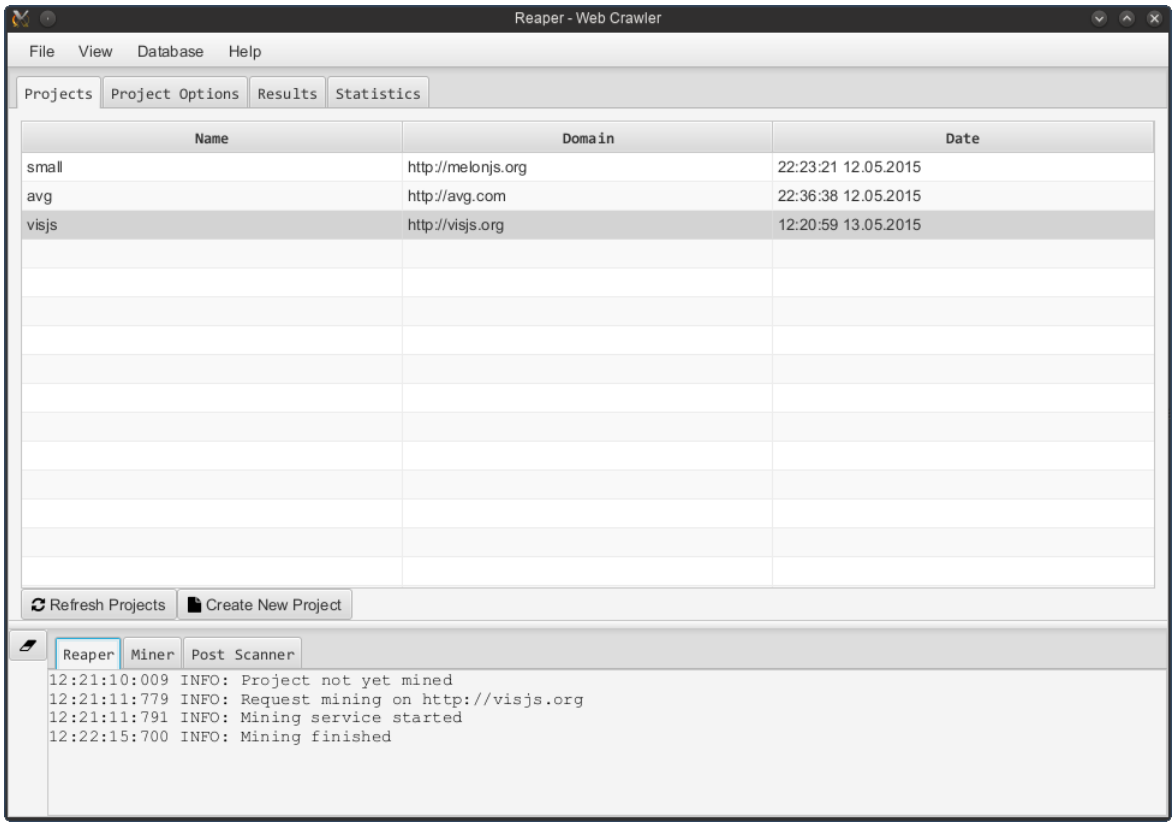
4.3 Grafické rozhraní

JavaFX umožňuje možnost navrhnout design ve specialní XML konstrukci, která se jmenuje FXML. Designové komponenty se zapisují stejně jako XML uzly. Uzly mají přiřazené indetifikátory, které jsou předávány controlleru, tedy ovladači, což mezivrstva mezi modelovací a zobrazovací vrstvou.

V návrhu jsem grafickému rozhraní nevěnovala příliš mnoho času, finální návrh aplikace vznikla množstvím iterací a experimentů s neustálou obměnou designových komponent. Aplikace je rozdělena do několika záložek. Kromě záložek uživatel ovládá aplikaci přes menu panel v horní části aplikace. Ve spoední části aplikace se nachází několik záložek s výpisy konzole programu a služeb. Tyto výpisy slouží k informování uživatele o činnosti aplikace,

jelikož aplikace provádí velké množství operací na pozadí a uživateli by se mohla zdát jako neresponzivní. Pravým tlačítkem s ikonou gumy uživatel smaže obsah právě zvoleného výpisu.

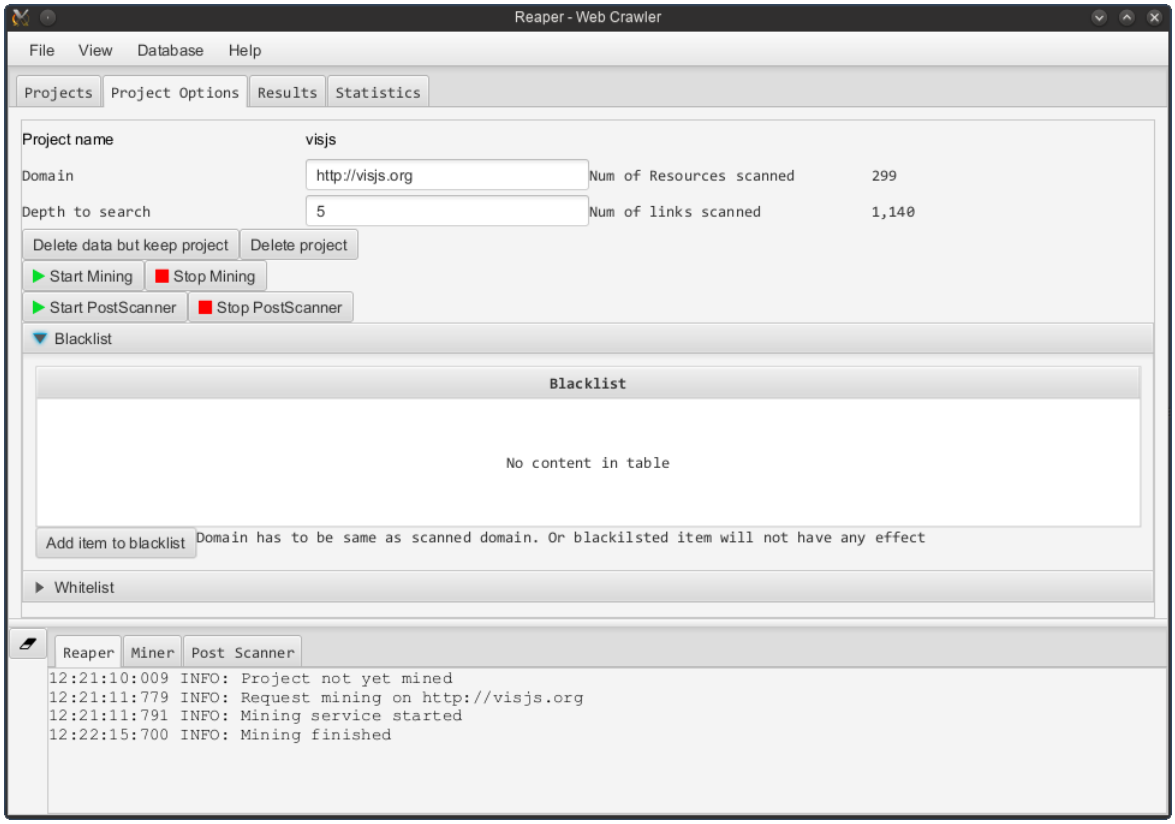
Úvodní okno obsahuje tabulku s projekty. Projekt je možno otevřít, smazat či vytvořit nový.



Obrázek 4.4: Úvodní okno

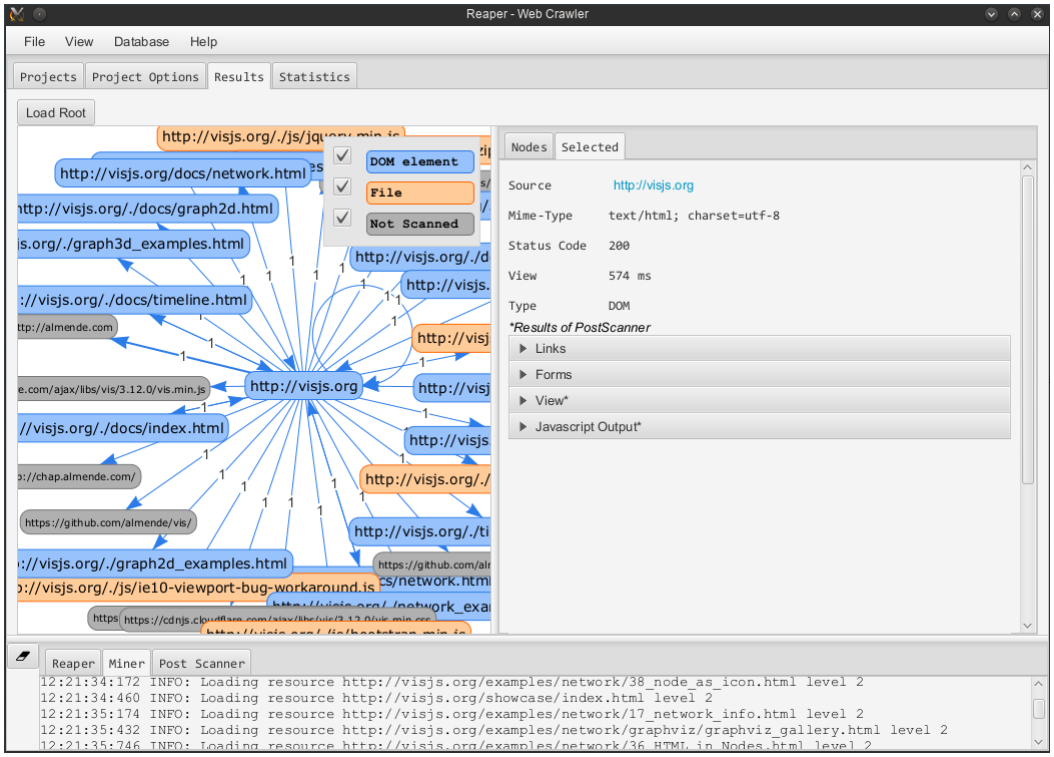
V této záložce je uživatel schopen změnit nastavení prohledávání a to konkrétně cílovou doménu, hloubku, vymezení prohledávaného prostoru. Také je schopen smazat data z již existující analýzy, anebo zahájit analýzu novou. Jak již bylo uvedeno v kapitole (doplnit), analýza probíhá ve dvou fázích. Z důvodu časové náročnosti je nutné spustit obě fáze manuálně. Jelikož druhá fáze zvyšuje lineárně nárok na prostředky a čas k její vyhotovení a informace získane v teto fázi nemusí být pro uživatele důležité.

Po spuštění analýzy aplikace uzamkne velké množství ovládacích prvků dokud analýza není dokončena. Bez tohoto zabezpečovacího mechanismu by uživatel byl schopen vydat příkaz k například smazání dat zatímco jsou stahována a zapříčinil tak nedefinovanému chování aplikace.



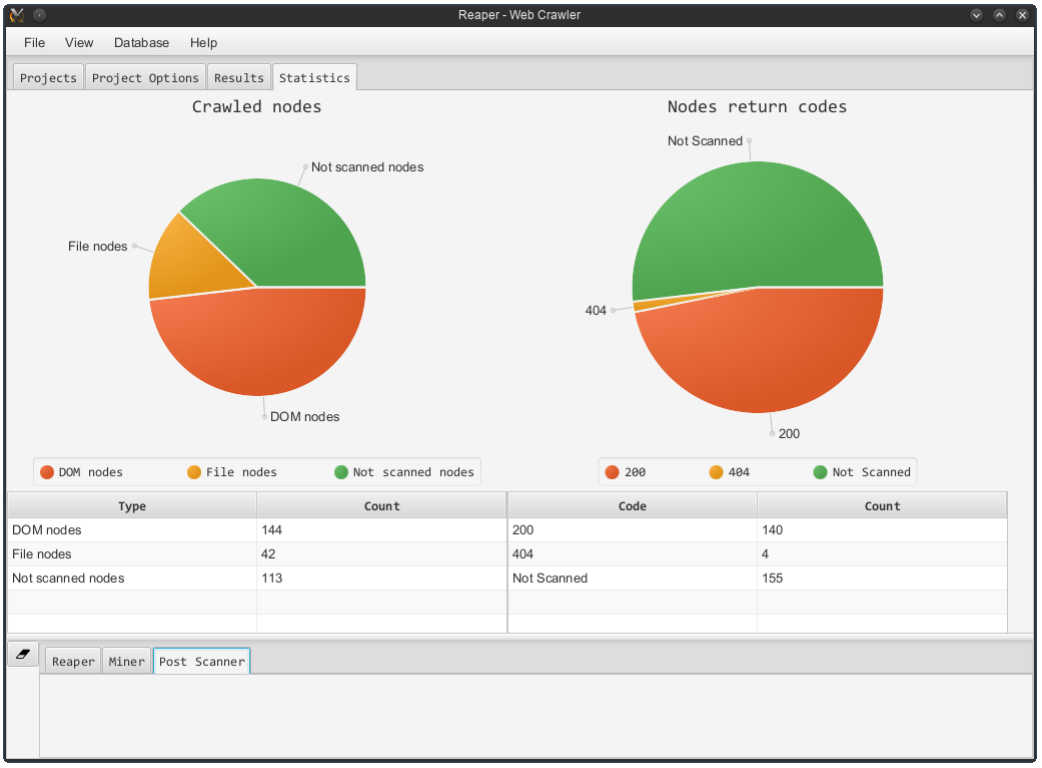
Obrázek 4.5: Ovládání analýzy

Při načtení projektu se v záložce výsledky načte kořenový objekt analýzy. Procházení výsledků je rozděleno do zobrazení grafu vlevo a seznam právě zobrazených objektů včetně detailu procházeného objektu. Graf vsazen do prohlížeče, kde jej vykresluje Javascriptová knihovna `Vis.js` . Tato knihovna nabízí jednoduchou funkcionalitu pro vykreslování časových diagramů a grafů. Graf je vykreslen několika barevně, každá z barev označuje typ webového objektu. U hran mezi uzly je taky zobrazeno číslo, toto číslo reprezentuje kolik odkazů dohromady vede mezi dvěma webovými objekty. V pravém horním rohu jsou přítomny 3 checkboxy, pomoci kterých uživatel ovládá, jaké typy webových objektů se v grafu zobrazí. V pravém panelu je ve dvou záložkách umístěn seznam zobrazených objektů v tabulce a taktéž detail právě procházeného objektu. Množství informací zobraného v detailu závisí na typu webového objektu. Např. u objektu, který nebyl skenován je zobrazeno pouze jeho adresa. Oproti tomu objekt s DOM objektem obsahuje i seznam odkazů, formulářů, které objekt obsahuje. Či např obrázek vykreslené stránky.



Obrázek 4.6: Výsledky analýzy

V záložce statistik jsou zobrazeny dva grafy s tabulkou jejich hodnot. Levý graf reprezentuje z jakých webových objektů v jakém množství je složena prohledaná oblast. V pravém grafu je možno vidět jaké odpovědi serveru byly vráceny. Nutno podotknout že např. chybu 404 mnohé webové stránky nevrací správně, uživateli je zobrazeno hlášení, že hledaný obsah nebyl nalezen, server ale ve skutečnosti odpoví kódem 200.



Obrázek 4.7: Statistiky

4.4 Webové rozhraní

Původní plán této práce bylo také aplikaci v určitém bodě přenést do webového rozhraní. Uživatele by k aplikaci mohli přistoupit prostřednictvím svého prohlížeče. Tato možnost bohužel tohoto projektu již není možná z důvodu rychle rostoucími požadavky na zdroje, které aplikace potřebuje ke svému běhu.

Další z problému, který se v této oblasti objevil je rozhodnutí, společnosti Google stojícím za prohlížečem Google Chrome, o ukončení podpory `NPAPI` pluginu. Toto rozhodnutí bylo stanoveno do konce roku 2015, po vstupu do účinnosti nebude možné spustit Java kod v prohlížeči Google Chrome, aplikace by tak byla podporována pouze v prohlížečích Firefox, Internet Explorer a Safari.

Možné řešení tohoto problému by byl návrh čistě webové aplikace, která by měla přístup k vytvořené databázi a zároveň k souborovému systému s uloženými vykreslenými snímky analýzy. Bohužel toto řešení vyžaduje příliš velké množství času a značně by tím utrpěli ostatní body této práce.

Kapitola 5

Výsledky

Experimenty jsem prováděla na následující konfiguraci.

CPU	FX-6300 3,5GHz
RAM	12GB DDR3
OS	Fedora 20
Java	JDK 1.8.40

Tabulka 5.1: HW/SW konfigurace

Analýzy jsem prováděla na doménách s různým počtem webovým objektů. Domény označené jako menší označuji veškeré domény jejichž analýza objevila méně jak 1000 objektů. Domény jako větší označuji všechny ty, které toto číslo přesáhly.

Spotřeba RAM paměti aplikace po několika hodinách spuštěné analýzy nepřekročila 1 GiB. Toto považuji za veliký úspěch, protože se tím podařilo vyřešit problémy prototypu zmíněně v kapitole návrhu aplikace 3. Na úkor spotřeby zdrojů aplikace se projevil potřebný čas k dokončení analýzy. U domén skládajících se z desítky tisíc odkazů je analýza velice pomalá, např. analýza portálu seznam.cz se dostala do třetí úrovně odkazu až po několika hodinách.

Žádná analýza domén s větším počtem objektů nebyla dokončena během jednoho spuštění. Stávalo se, že během analýzy vypadlo internetové připojení, či dočasně neodpovídala databáze. Z těchto důvodu jsem provedla úpravy v programu aby veškeré data během analýzy se ukládaly do databáze, výsledkem tohoto snažení jsou třídy `LinkQue` a `LinkSet`. V případě opětovné spuštění analýzy, aplikace pokračuje v analýze kde předtím přestala.

I přesto tyto velké domény se mi nepodařilo prozkoumat do větších hloubek, se vzrůstajícím počtem záznamů v databázi se rychlost zpracování dotazu čím dál víc zpomalovala a po několika hodinách zpracovaný počet odkazu se pohyboval kolem několika tisíc a počet odkazů ve frontě se neustále zvyšoval k několika desítkám tisíc.

Oproti tomu analýza obou částí u menších domén trvá téměř vždy pod deset minut.

Kapitola 6

Další možná rozšíření

Některé části této aplikace jsou zpracovány velice jednoduše a existuje mnoho možností jak by dál mohly rozšířit. Zde je uveden seznam věcí, které jsem nestihla dotáhnout do finální podoby.

Jak první tak i druhá část analýzy probíhá synchronně v jednom vlákně. Tato vlastnost nevyužívá dostatečně zdroje počítače a analýza je z tohoto důvodu velmi pomalá. Prvním krokem k napravení by bylo rozšířit první část analýzy k využití několika vláken. Druhá část analýzy při každém spuštění externího programu čeká na jeho ukončení, spouštění několika procesů zároveň by se dosáhlo lepších výsledků. Další podstatnějš náročnější možnost by bylo rozdělení služeb provádějící analýzu a síť klientských programů na více zařízeních. Tyto programy by pracovaly na jedné doméně současně, jako svůj hlavní komunikační prostředek by využily databázi.

Kromě chybějícího webového rozhraní je např. záložka statistik zmíněná v sekci 4.3 chudá na důležité informace. Uživatel se může prostřednictvím grafu dozvědět, že se na doméně nachází objekty, jejichž pokus o zpřístupnění způsobil chybu 500, ale nedoví se, které objekty to byly. V záložce statistik chybí také jakákoliv analýza rychlosti přístupu na objekty a velikosti objektů. Uživatele aplikace by mohlo zajímat, které části webových stránek se ukázaly jako nejpomalejší a podrobit tyto části podrobnějším zkoumáním.

Při vyhledávání a zkoumání formulářů se do databáze ukládá jen velmi malé množství dat, informace o polích formuláře zcela chybí.

Každý webový server je něčím unikátní, tvořit nástroje, které se snaží "vyhovět všem" může být náročně až nemožné. Pokud by aplikace podporovala možnost vložení uživatelských skriptů při práci prohlížeče ve druhé části analýzy, byl by uživatel schopen dodefinovat chování pavouka při specifických případech.

Kapitola 7

Závěr

Podařilo se mi vytvořit aplikaci, která je schopna analyzovat doménou a vytvořit její graf v databázi spolu s údaji o zmapovaných objektech. Aplikace je postavena na frameworku JavaFX a jako úložiště využívá databázi s grafovým modelem OrientDB. Aplikace je také schopná výsledky analýzy prezentovat uživateli. Data jsou uložena na uživateli přístupné místo k dalšímu zpracování/využití.

Aplikace si neklade příliš velké nároky na dostupné prostředky. Namísto toho zpomaluje celkový čas analýzy. Z experimentů je možné vyvodit, že aplikace není vhodná na analýzu dynamických a obsahově rozsáhlých domén. Přesto se může projevit užitečná pro domény s menším a hlavně statickým obsahem. Při opakovaném spuštění můžeme pozorovat změny v struktuře webových objektů i jejich množství.

Webová prezentace dat v aplikaci chybí, ale v sekci 4.4 bylo navrhnuto řešení. Při implementaci aplikace mě napadlo množství dalších rozšíření a směrů, kam by vývoj aplikace mohl dále pokračovat, tyto rozšíření jsou popsána v kapitole 6.

Literatura

[1] Berners-Lee, T.; W3C/MIT; Fielding, R.; aj.: Uniform Resource Identifier (URI). leden 2005.
URL https://www.ietf.org/rfc/rfc3986.txt

[2] Chakrabarti, S.: *Mining the web: Discovering Knowledge from Hypertex Data*. Morgan Kaufmann Publishers, 2003, iSBN 1-55860-754-4.

[3] Hedley, J.: Jsoup: Java HTML Parser. 2015.
URL http://jsoup.org/

[4] Hidayat, A.: PhantomJS. 2015.
URL http://phantomjs.org/

[5] Oracle: Java Platform, Standard Edition (Java SE) 8. 2015.
URL http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

[6] Technologies, O.: OrientDB Manual - version 2.0. 2015.
URL http://orientdb.com/docs/last/

Příloha A

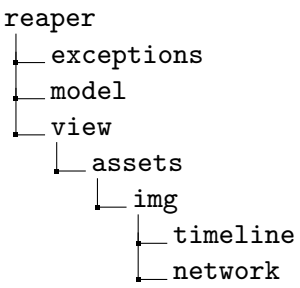
Obsah CD

[Vložit Netbeans projekt????]

- Složka `src`
  - Zdrojové soubory aplikace
- Složka `dist`
  - Přeložený balíček přibalen s potřebnými soubory
- Složka `libs`
  - Knihovny potřebné k přeložení

Příloha B

Struktura programu



Příloha C

Slovníček pojmů

[Odstranit nebo to tu nechat?]

- Frontend - Klientská část zpracování požadavku
- Backend - Serverová část zpracování požadavku
- Crawler - pavouk
- Resource - zdroj/webový objekt
- Headless browser - Webový prohlížeč běžící v pozadí
- Controller - řídící prvek
- View - (doplnit)
- Service - služba