

# 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

AUTHOR

NIKITA VAŇKŮ

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**  
AUTHOR

**NIKITA VAŇKŮ**

**VEDOUCÍ PRÁCE**  
SUPERVISOR

**Ing. RADEK BURGET, Ph.D.**

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 procházet WWW domény menšího rozsahu. Aplikace je užitečná jako nástroj k sestavení struktury WWW stránek.

## **Abstract**

The aim of this thesis is to create an application for World Wide Web page analysis. JavaFX framework with using of graph database OrientDB has been chosen as implementation language. Application is capable of analysis small to medium based web domain and creating their structure.

## **Klíčová slova**

Webový pavouk, HTML, analýza webových stránek, jsoup, PhantomJS, JavaFX, OrientDB

## **Keywords**

Web crawling, HTML, web page analysis, jsoup, PhantomJS, JavaFX, OrientDB

## **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 vypracovala samostatně pod vedením pana Ing. Radka Burgeta Ph.D.

.....  
Nikita Vaňků  
19. května 2015

## Poděkování

Velice děkuji mému vedoucímu práce Ing. Radkovi Burgetovi, Ph.D. za motivaci, vedení a mou podporu.

© Nikita Vaňků, 2015.

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Obsah

<b>1</b>	<b>Úvod</b>	<b>2</b>
<b>2</b>	<b>Teorie</b>	<b>3</b>
2.1	Současní pavouci	3
2.2	Analýza URL	5
2.3	Extrakce URL	6
2.4	Zaznamenání časových událostí	6
<b>3</b>	<b>Návrh aplikace</b>	<b>7</b>
3.1	Java	7
3.2	Specifikace prohledávaného prostoru	7
3.3	Analýza DOM stromu	8
3.4	Uložení dat	8
3.4.1	Databáze	8
3.4.2	Souborový systém	9
3.5	Vykreslení dokumentu	9
3.6	Projekty	9
3.7	Návrh uživatelského rozhraní	9
<b>4</b>	<b>Implementace aplikace</b>	<b>10</b>
4.1	Analýza domény	10
4.1.1	Tvorba grafu	10
4.1.2	Doplnění dodatečných informací	11
4.2	Databáze	11
4.2.1	Schéma databáze	12
4.3	Grafické rozhraní	13
4.4	Webové rozhraní	18
<b>5</b>	<b>Výsledky</b>	<b>19</b>
<b>6</b>	<b>Další možná rozšíření</b>	<b>21</b>
<b>7</b>	<b>Závěr</b>	<b>22</b>
<b>A</b>	<b>Obsah CD</b>	<b>24</b>

# Kapitola 1

## Úvod

World Wide Web (dále jen WWW) dokumenty jsou čím dál více využívanými sdělovacími prostředky. Jejich počet neustále roste[6]. WWW dokumenty se využívají pro širokou škálu účelů. Můžeme narazit na dokumenty se spíše statickým obsahem jako jsou blogy, firemní reprezentace a jiné, ale také na dokumenty s kompletně dynamicky generovaným obsahem jako jsou např. internetové vyhledávače, informační systémy či další webové aplikace.

Množství webových stránek na doméně je téměř neomezené. Na Internetu lze najít domény o několika stránkách, 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 dotazů na server. Jedná se o měření rychlosti zpracování backendu a frontendu, velikost stahovaných klientských zdrojů jako jsou například soubory kaskádových stylů, obrázky či Javascriptové knihovny. Jedním z těchto nástrojů jsou Webové crawlery, neboli pavouci.

Pavouci našli největší využití hlavně u internetových vyhledávačů. Jsou navrženi tak, aby v krátkém čase dokázali zpracovávat najednou velké množství stránek z různých domén. Při zpracovávání hledají odkazy na další stránky a ty následně analyzují. Vzhledem k velikosti a neustalému růstu internetu pavouci mnohdy svou práci nikdy nekončí.

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 stahování stránek a vykreslování stránek. Součástí analýzy může být také sledování chyb Javascriptu či validace DOM stromu.

V této práci se budu věnovat několika hlavním částem této problematiky. Kapitola 2 bude věnována analýze problematiky pavouků. V kapitole 3 zpracuji návrh vlastního pavouka. Poté kapitola 4 řeší implementaci vyhotoveného návrhu. Nakonec rozeberu výsledky implementované aplikace.

# Kapitola 2

## Teorie

Tato kapitola je věnována teoretickému zpracování 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 počátečním uzlem a dalšími omezeními 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. Implementace těchto pavouků je často uzavřená a veřejnosti jsou známy pouze omezené informace o tom, jak tito pavouci přistupují na jejich stránky.

Vyhledávač Bing na svých stránkách uveřejňuje seznam pavouků, které využívá k jakým účelům. Kromě pavouků **Bingbot**, který plní funkci indexování, se můžeme dočíst i o pavouků zpracovávajícím audiovizuální data **MSNBot-Media** či o pavouku vykreslujícím obsah webu **BingPreview**. [7]

Kromě pavouků procházejících a indexujících celý internet existuje i celá řada úzce specializovaných pavouků pro určitou činnost.





**skipfish**<sup>1</sup> je Open Source pavouk testující zabezpečení domény a odhalující slabá místa, která by mohli využít útočníci. Výsledkem testu je interaktivní mapa stránek.

---


<sup>1</sup><https://code.google.com/p/skipfish/>


### Crawl results - click to expand:

---




 **http://www.example.com/** 3 2 171  
Code: 200, length: 438, declared: text/html, detected: text/html, charset: UTF-8 [ show trace + ]




---



 **New 404 signature seen**  
1. Code: 404, length: 285, declared: text/html, charset: iso-8859-1 [ show trace + ]



 **New 'Server' header value seen**  
1. Code: 200, length: 438, declared: text/html, charset: UTF-8 [ show trace + ]  
Memo: Apache/2.2.3 (CentOS)

---

 **error** 3 5  
Code: 403, length: 288, declared: text/html, detected: text/html, charset: iso-8859-1 [ show trace + ]




 **include** 2 3  
Code: 403, length: 296, declared: text/html, detected: text/html, charset: iso-8859-1 [ show trace + ]

 **README** 1  
Code: 200, length: 1979, declared: text/plain, detected: text/plain, charset: UTF-8 [ show trace + ]

 **icons** 164  
Code: 200, length: 30034, declared: text/html, detected: text/html, charset: ISO-8859-1 [ show trace + ]

### Document type overview - click to expand:

---

-  **application/xhtml+xml** (1)
-  **image/gif** (5)
-  **image/png** (9)

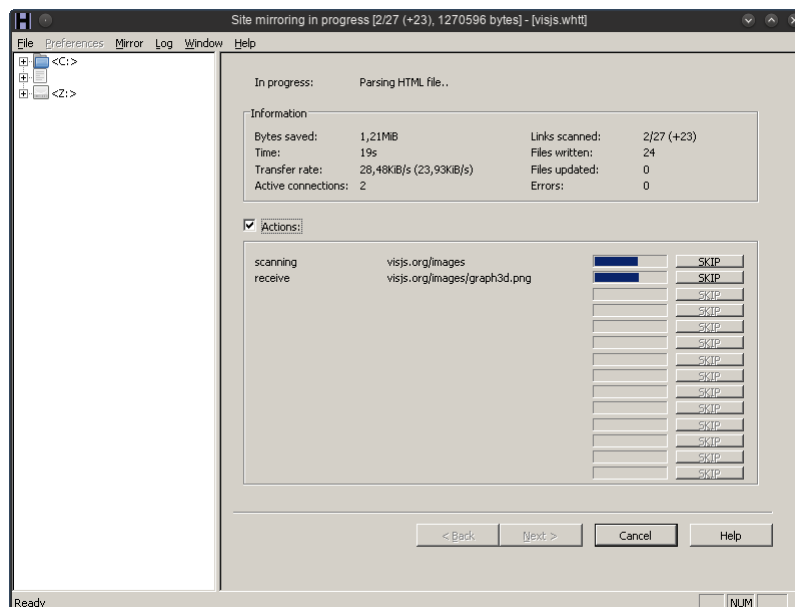
Obrázek 2.1: Skipfish

HTTrack<sup>2</sup> je pavouk, který po zadání adresy prochází webový obsah a u uživatele tvoří jeho offline kopii. Obsah je strukturován do stejné adresářové hierarchie jako na serveru a zkonstruuje relativní hypertextové odkazy. Po vytvoření kopie lze lokálně uložené stránky procházet v prohlížeči. Pavouk umí také aktualizovat již existující kopie webu a lze ho tak využít k vytvoření a udržování zrcadlových webů, tedy webů, které mají stejný obsah. Zrcadlování webů je oblíbená taktika, jak zabezpečit udržení obsahu online při výpadcích či útocích na webové servery.

---

<sup>2</sup><https://www.httrack.com/>





Obrázek 2.2: HTTrack

Pavouci ve své povaze přistupují k webovým serverům v mnohem větším měřítku než obyčejný uživatel. Toto může znamenat, že server klasifikuje pavouka jako nebezpečného robota či snad pokus o DDos útok a zakáže pavoukům 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. Společnost google na svých stránkách uvádí, že GoogleBot nepřistoupí ke stránce častěji než jednou za několik sekund. [4] Takové umělé zpomalení prohlédání v mé aplikaci se může výrazně podepsat na čas potřebném k vyhotovení analýzy a z toho důvodu jsem se rozhodla tuto metodu vyloučit.

## 2.2 Analýza URL

Uniform Resource Locator (dále jen URL, neboli též adresa) je definován v RFC 3986[1]. Je potřeba rozlišit relativní a absolutní adresu. V případě relativní adresy se zohledňuje aktuální adresa dokumentu. Součástí URL může být též kotva, která 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.

Pavouk by také měl být připraven čelit generovaným hypertextovým 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 jim má věnovat pozornost. GET parametr může, ale nemusí význam pro vykreslení dokumentu. Mohou existovat i stránky obsahující odkazy s velkým množstvím opakujících se GET parametrů. GET parametry se v URL adrese uvádějí za cestou. První parametr je označen otazníkem a následuje jméno\_parametru=hodnota\_parametru, další parametry jsou odděleny ampersandem. Na

pořadí uvedení parametru nezáleží, jelikož seznam parametrů je ve své podstatě asociační pole. Pavouk by se měl s touto vlastností počítat a nevidět rozdíl mezi těmito uvedenými odkazy:

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

## 2.3 Extrakce URL

Extrakce adres je provedena analýzou HTML webových dokumentů. Prozkoumáním vybraných elementů je možno získat potřebné odkazy. Mezi zmiňované elementy patří:<sup>3</sup>

- `<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 konkretizovat zpracování odkazů, jelikož odkazy v těchto souborech hledat nebudeme.

## 2.4 Zaznamenání časových událostí

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

- čas strávený zpracováním požadavku na serveru - **Backend**
- čas strávený stažením a zobrazením zdrojových dokumentů - **Frontend**

Webová stránka, která se zobrazí uživateli v prohlížeči, je mnohdy složena z více souborů. Jedná se o HTML/XML dokument, ale také o obrázky, Kaskádové 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. [9]

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

---

<sup>3</sup>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, databázové uložiště, které uchovává analyzovaná data a grafickou nadstavbu, 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[8]. JavaFX aplikace je možno využít i jako desktopové aplikace. Aplikace vytvořena součástí této práce byla naprogramována pro Java 8, JavaFX 8.

### 3.2 Specifikace prohledávaného prostoru

Cílem aplikace je analyzovat pouze jedinou doménu. Při prohledávání je nutno zamezit tomu, aby pavouk pavouk zabloudil na cizí domény a prohledával zcela jiné části internetu. Toho 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. Což se uskuteční stanovením maximální hloubky prohledávání. Před prohledáním prvního dokumentu je aktuální hloubka 0 a každým dalším dokumentem se hodnota zvýší o 1. V případě dosažení maximální hloubky prohledávání se z tohoto dokumentu neextrahují 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 doménách najednou. Z tohoto důvodu se prohledávání řídí také 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 stromu

Extrakce adres z HTML dokumentů je možná za pomoci prohledání Document Object Model (dále jen DOM). DOM prezentuje webový dokument jako strom, který je možno procházet a manipulovat s ním. Ve stromu 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 přidá záznam na kterém dokumentu byl nalezen a následně se vloží do fronty ke zpracování.

Pro práci s DOM stromem a extrakci adres je nejjednodušší využít HTML parser, který převede dokument do struktur jazyka pro přímou manipulaci. Implementací takového parseru je v Javě hned několik.

- **NekoHTML**<sup>1</sup> Jednoduchý HTML parser, dokáže se vypořádat s řadou menších chyb v dokumentech.
- **HTMLCleaner**<sup>2</sup> OpenSource HTML parser navržený tak, aby zpracoval nevalidní dokumenty s větším množstvím závažných chyb. Po zpracování HTML dokumentu vytvoří dobře strukturovaný XML dokument, k elementům je proto možno přistupovat pomocí XPath
- **jsoup**<sup>3</sup> HTML parser navržený na zpracování také nejnovějšího standardu HTML5. Taktéž zpracuje nevalidní dokumenty s větším množstvím chyb. Výsledkem zpracování je DOM strom, se kterým je možno pracovat pomocí klasických javascriptových funkcí jako `getElementById`, `getElementsByClass`, či pomocí CSS a JQuery selectoru. Z tohoto důvodu jsem se rozhodla využít právě tento parser.

### 3.4 Uložení dat

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

Rozhodla jsem se využít databázi pro ukládání většinu dat.

#### 3.4.1 Databáze

Jako databáze byla zvolena **OrientDB 2.0.X Community Edition**<sup>4</sup>, jedná se o hybridní NoSQL databáze umožňující pracovat jak v Dokument (MongoDB<sup>5</sup>), tak Grafovém (Neo4j<sup>6</sup>) modu.

Aby aplikace nemusela skladovat v paměti příliš velké množství dat, data se co nejdříve po zpracování přesunou do databáze. Jelikož zpracovaná 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 způsob uložení dat na doméně.

---

<sup>1</sup><http://nekohtml.sourceforge.net/>

<sup>2</sup><http://htmlcleaner.sourceforge.net/>

<sup>3</sup><http://jsoup.org>

<sup>4</sup>V průběhu vývoje aplikace byla verze několikrát aktualizovaná

<sup>5</sup><https://www.mongodb.org/>

<sup>6</sup><http://neo4j.com/>

Grafová databáze je sestavena stejně jako graf ze dvou základních stavebních prvků. Vrcholy a hrany mezi vrcholy. Jako vrcholy si můžeme například představit 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, která 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 jejích dat. V případě souborového systému musí být tento systém vždy nějakým způsobem připojen k systému, na kterém běží aplikace.

## 3.5 Vykreslení dokumentu

Pro vykreslování webových dokumentů 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 [5].

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 nevhodná k uložení do databáze jsou uložena v souborovém systému a projekt by je měl být schopen nalézt.

## 3.7 Návrh uživatelského rozhraní

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

Uživatel má být schopen v aplikaci plně pracovat s projekty. Má tedy možnost vytvořit nový projekt, nebo otevřít již vytvořený, spustit analýzu domény či smazat již hotovou analýzu a hlavně procházet výsledky analýzy. Pro zobrazení a procházení výsledků jsem se rozhodla výsledek analýzy graficky reprezentovat jako graf. Ovšem graf s větším množstvím objektů se uživateli může zdát nepřehledný. Z tohoto důvodu jsem zvolila 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 tomu, jak se závěrečná 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 řídicí prvek. Řídicí prvek komunikuje taktéž s View, tedy pohledem, přes množinu komponent, která je v pohledu definovaná. Řídicí prvek sleduje a reaguje na zprávy zaslané pohledem, které zpracovává a přeposílá dále. Pokud se jedná o vlastnosti či parametry modelu, řídicí prvek je může pevně obou či jednostranně svázat s komponentou. Např. vlastnost modelu datového typu String může být svázaná s komponentou text v pohledu. Při jakékoliv změně této vlastnosti v modelu je komponenta aktualizována.

V datovém modelu je obsažena hlavní funkcionality aplikace. Jádro celé aplikace zastupuje třída **Crawler**. Crawler je spuštěn ve stejném vlákne jako řídicí prvky a pohledy. Funguje jako hlavní řídicí jednotka. Pokud uživatel zadá příkaz k modifikaci projektu či spuštění analýzy, Crawler přepoše 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 doplnění dodatečných informací (vykreslování stránek obsahující DOM strom).

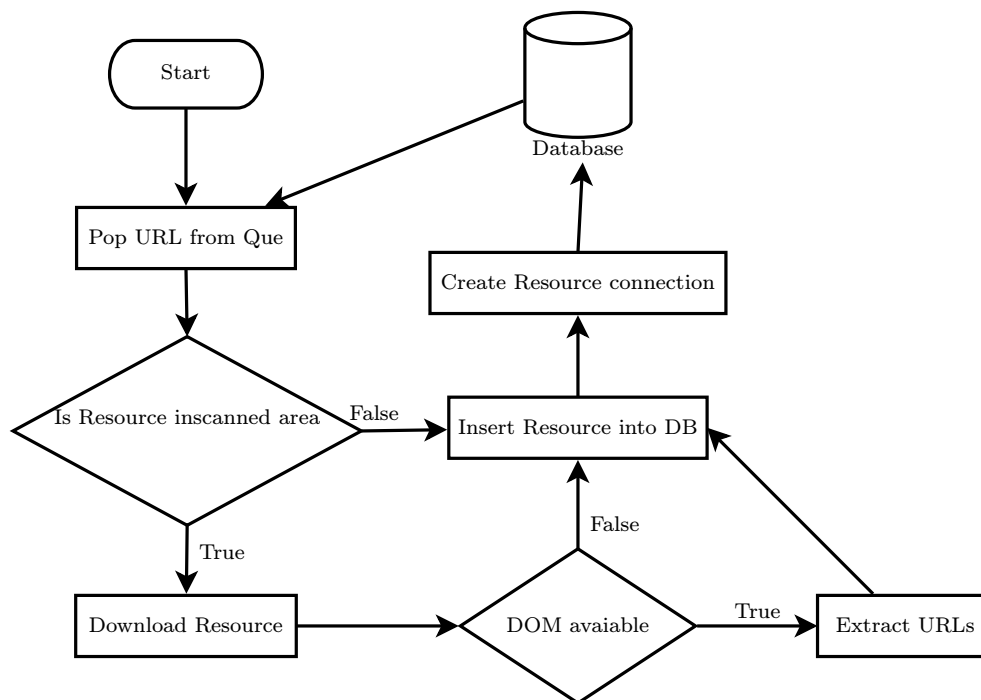
### 4.1 Analýza domény

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 Tvorba grafu

Služba sestavující graf webových objektů 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 prozkoumán nástrojem `jsoup`. Nástrojem jsou prozkoumány tagy a jejich atributy zmíněné v sekci 2.3, veškeré nalezené odkazy jsou vloženy zpět do fronty. Jestliže 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 Doplnění dodatečných informací

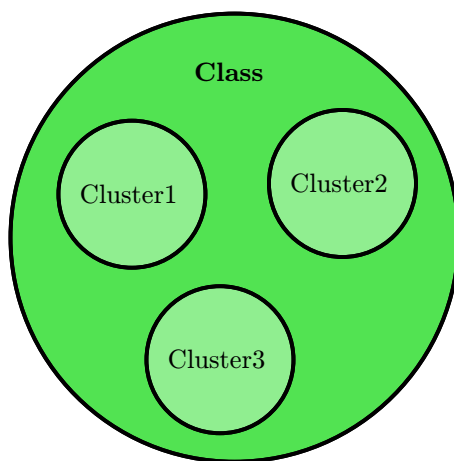
V čase spuštění druhé části analýzy je v databázi přítomen již hotový graf objektu s jejich základními informacemi. Služba z databáze extrahuje objekty, které obsahují DOM strom, a pro každý z těchto objektů nejdříve vytvoří své místo ve stromu souborového systému, kam bude služba později ukládat informace o objektu. Cesta k tomuto adresáři se skládá z názvu projektu a speciálního identifikátoru přiřazeného Orient databázi. Po dokončení přípravy započne spuštění 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 dokument a jako graf. Každý z těchto dvou módů se hodí pro jinou aplikaci. Jak již bylo zmíněno v sekci 3.4.1, pro ukládání webových objektů 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 také 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 má schopnost dědičnosti. Pokud chce uživatel vytvořit třídu, která bude uzlem nebo hranou v grafu, vytvoří třídu, 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 chápána jako dokument.

Orient Databáze ukládá záznamy tříd do tzv. clusterů. Každá třída má alespoň jeden cluster, do kterého ukládá svá data. Clusterů může být více, vkládání do nich může být prováděno přímo či na základě algoritmu. Clustery jsou od sebe fyzicky odděleny, tato vlastnost se využívá obzvláště pokud databáze běží v redistribučním modu na několika serverových jednotkách [10]. Mnoho databází 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 začínají objevovat data, ke kterým se bude přistupovat méně či téměř vůbec. Pokud se taková data přesunou do jiných clusterů a databáze pracuje s jedním 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 cluster, 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 vykonávání dotazů s větším množstvím již analyzovaných projektů.

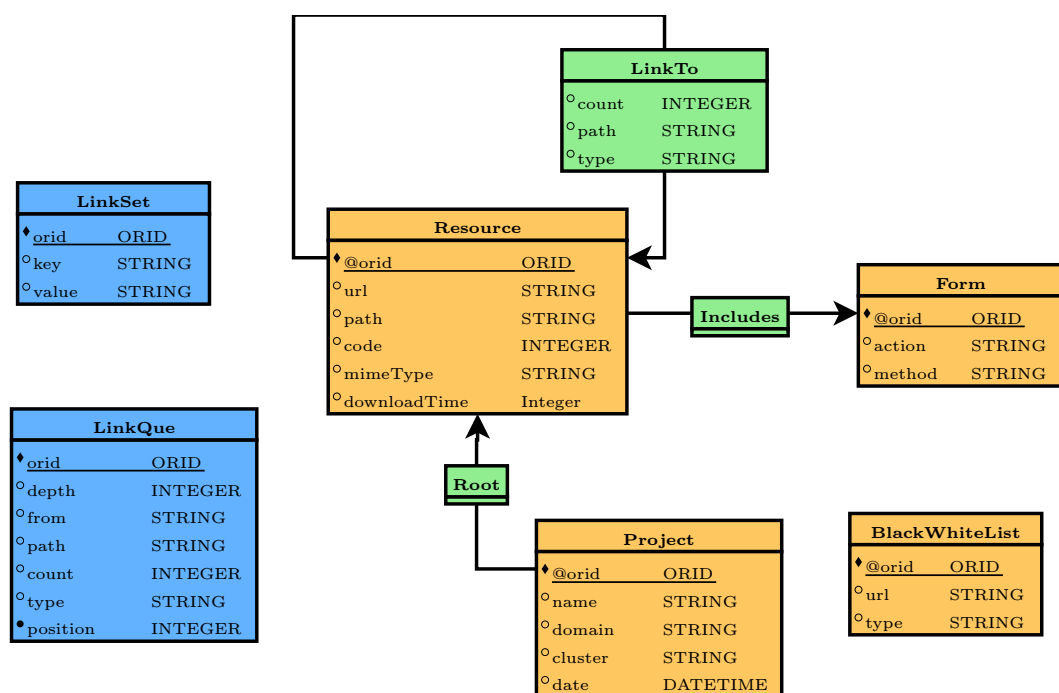
#### 4.2.1 Schéma databáze

V tabulce 5.1 a diagramu 4.3 je znázorněno jak vypadá finální schéma databáze. Projektové cluster se tvoří v třídách `Resource`, `Form`, `BlackWhitelist`, `LinkTo`, `Includes`, `LinkQue`, `LinkSet`. Název těchto clusterů 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



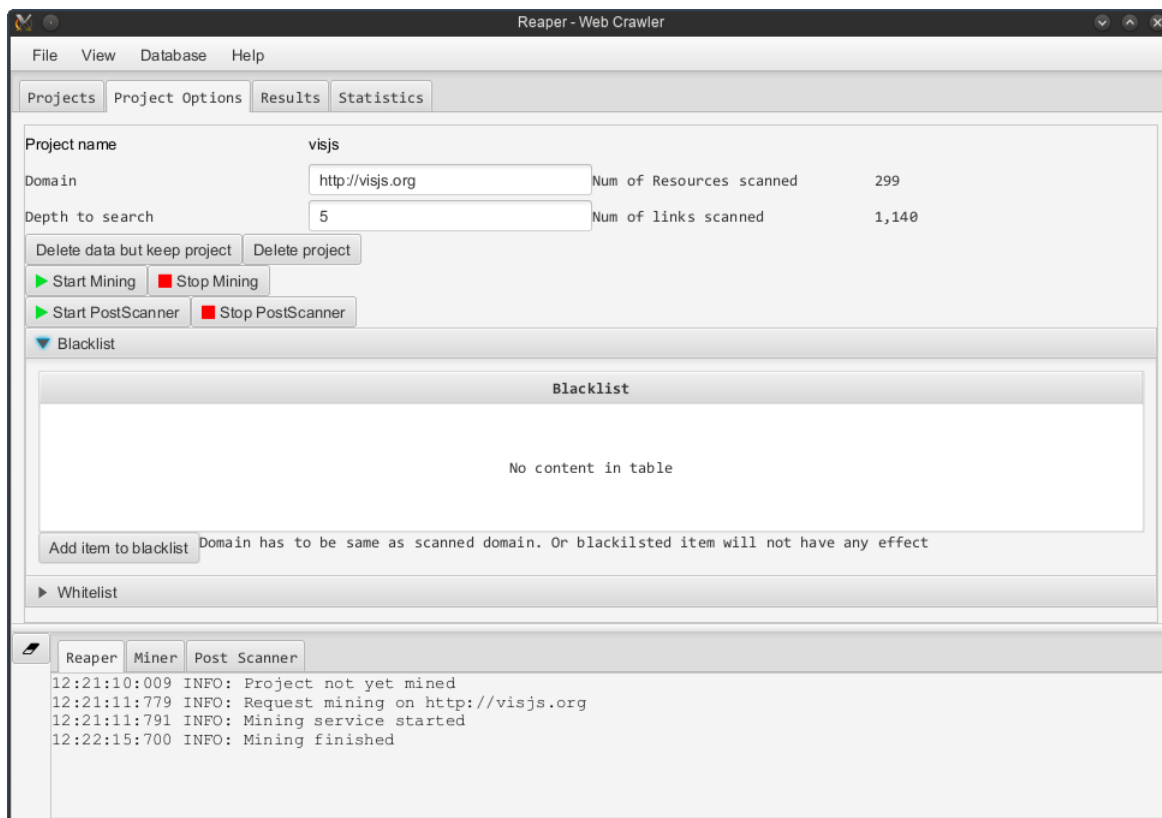
Obrázek 4.3: Diagram databáze

### 4.3 Grafické rozhraní

JavaFX umožňuje možnost navrhnout design ve speciální 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 je mezivrstva mezi modelovací a zobrazovací vrstvou.

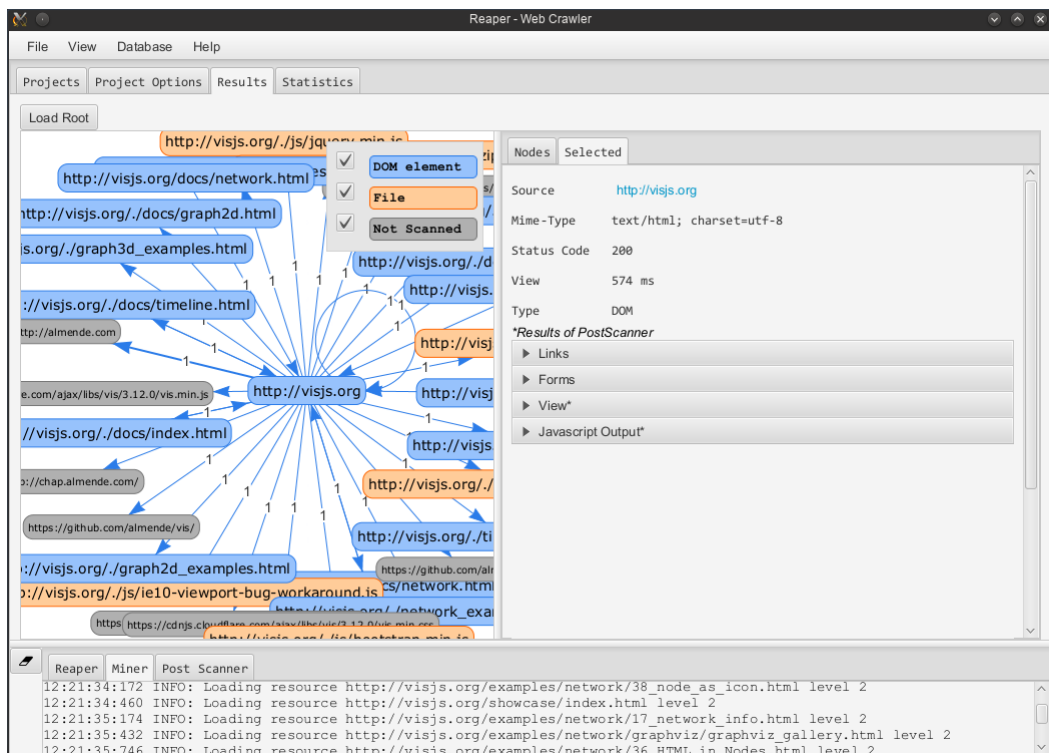
V návrhu jsem grafickému rozhraní nevěnovala příliš mnoho času, finální návrh aplikace vznikl množstvím iterací a experimentů s neustálou obměnou designových komponent. Aplikace je rozdělena do několika záložek. Kromě záložek uživatel ovládá aplikaci přes menu panel v horní části aplikace. Ve spodní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





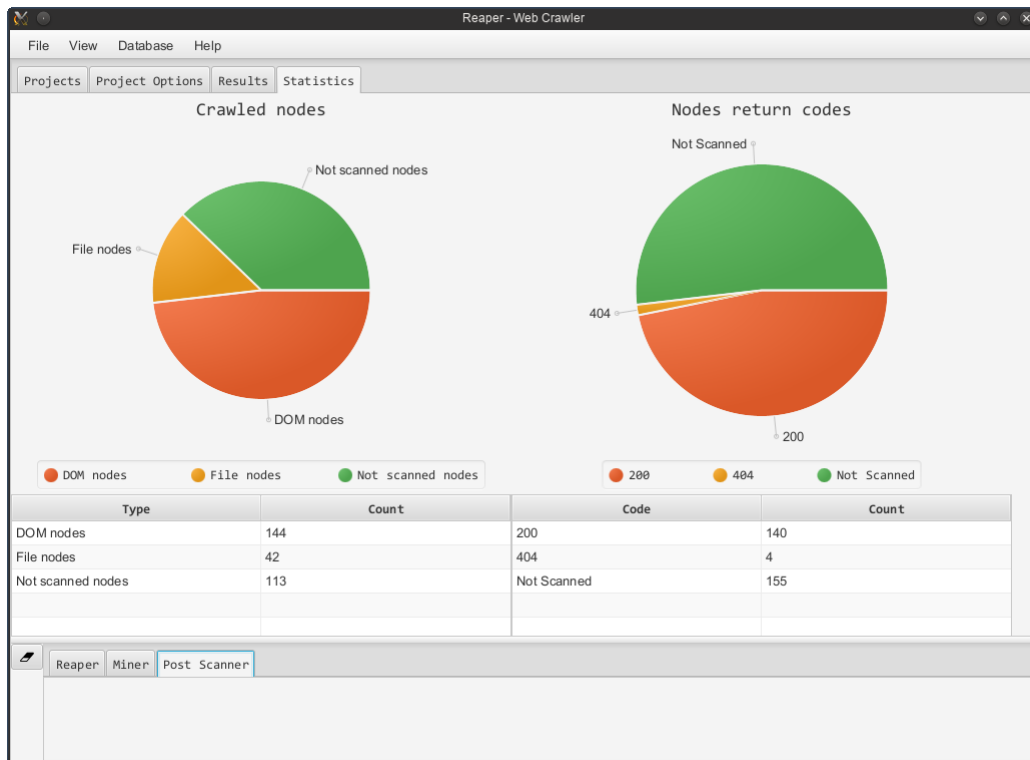
Obrázek 4.5: Ovládání analýzy

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 je vsazen do prohlížeče, kde jej vykresluje Javascriptová knihovna *Vis.js*. Tato knihovna nabízí jednoduchou funkcionalitu pro vykreslování časových diagramů a grafů. Graf je vykreslen několikabarevně, každá z barev označuje typ webového objektu. U hran mezi uzly je také zobrazeno číslo, toto číslo reprezentuje kolik odkazů dohromady vede mezi dvěma webovými objekty. V pravém horním rohu jsou 3 checkboxy, pomocí 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 zobrazené v detailu závisí na typu webového objektu. Např. u objektu, který nebyl skenován je zobrazeno pouze adresa. Oproti tomu objekt s DOM stromem 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 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: Statistika



# Kapitola 5

## Výsledky

Experimenty jsem prováděla na následující konfiguraci.

CPU	FX-6300 3,5GHz
RAM	12GB DDR3
OS	Fedora 20
Java	JDK 1.8.0_40

Tabulka 5.1: HW/SW konfigurace

Analýzy jsem prováděla na doménách s různým počtem webovým objektů. Jako menší označuji všechny domény, jejichž analýza objevila méně jak 1000 objektů. Větší domény jsou pak ty, které toto číslo přesáhly. Během analýzy byla každá stránka stažena a zpracována.

Spotřeba RAM paměti aplikace po několika hodinách spuštěné analýzy nepřekročila 1 GiB. Toto považuji za veliký úspěch, protože se tím podařilo vyřešit problémy prototypu zmíněného 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.

URL	Webové objekty			Odkazy	Fronta <sup>1</sup>	T <sup>2</sup>
	Stránky	Soubory	Neskenované <sup>3</sup>			
http://jsoup.org	503	11	110	6440	0	16:32
http://visjs.org	144	42	113	1440	0	1:05
http://seznam.cz	313	155	267	1416	21998	3:31:15
http://phantomjs.org	193	2	505	12458	0	1:05:15

Tabulka 5.2: Tabulka výsledků analýz

Žádná analýza domén s větším počtem objektů nebyla dokončena během jednoho spuštění. Stávalo se, že během analýzy vypadlo internetové připojení, či dočasně neodpovídala

<sup>1</sup>Počet odkazů čekající na zpracování

<sup>2</sup>Délka první části analýzy

<sup>3</sup>Mezi neproskenované objekty se počítají objekty, které se nacházely mimo prohledávaný prostor, či objekty za hranicemi hloubky analýzy.

databáze. Z těchto důvodů jsem provedla úpravy v programu, aby se veškerá data během analýzy ukládala do databáze. Výsledkem tohoto snažení jsou třídy `LinkQue` a `LinkSet`. V případě opětovného spuštění analýzy aplikace pokračuje v analýze tam, kde při předchozím spuštění přestala.

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



## 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 jsou nad rámec rozsahu bakalářské práce.

Jak první, tak 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, spuštěním několika procesů zároveň by se dosáhlo lepších výsledků. Další podstatně náročnější možností by bylo rozdělení služeb provádějící analýzu a síť klientských programů na více zařízení. Tyto programy by pracovaly na jedné doméně současně a jako svůj hlavní komunikační prostředek by využily databázi.

Záložka statistik zmíněná v sekci 4.3, je chudá na důležité informace. Uživatel se může prostřednictvím grafu dozvědět, že se na doméně nachází objekty, jejichž pokus o zpřístupnění způsobil chybu 500, ale nedozví se, které objekty to byly. V záložce statistik chybí také jakákoliv analýza rychlosti přístupu na objekty a velikosti objektů. Analýza tyto hodnoty zaznamenává, ale nejsou ve statistice prezentovány. 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 ve specifických případech.

# Kapitola 7

## Závěr

Podařilo se mi vytvořit aplikaci, která je schopna analyzovat doménu 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 prezentovat výsledky analýzy uživateli. Data jsou uložena na uživateli přístupné místo k dalšímu zpracování/využití.

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

Při implementaci aplikace mě napadlo mnoho dalších rozšíření a směrů, kam by vývoj aplikace mohl dále pokračovat, tato 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 Hypertext Data*. Morgan Kaufmann Publishers, 2003, iISBN 1-55860-754-4.
- [3] Google: The Final Countdown for NPAPI. 2015.  
URL <http://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html>
- [4] Google: Googlebot. 2015.  
URL <https://support.google.com/webmasters/answer/182072>
- [5] Hidayat, A.: PhantomJS. 2015.  
URL <http://phantomjs.org/>
- [6] internetlivestats: Total Number of Websites. 2015.  
URL <http://www.internetlivestats.com/total-number-of-websites/>
- [7] Microsoft: Which Crawlers Does Bing Use? 2015.  
URL <http://www.bing.com/webmaster/help/which-crawlers-does-bing-use-8c184ec0>
- [8] Oracle: Java Platform, Standard Edition (Java SE) 8. 2015.  
URL <http://docs.oracle.com/javase/8/javase-clienttechnologies.htm>
- [9] Souders, S.: Domain Sharding revisited. 2013.  
URL <http://www.stevesouders.com/blog/2013/09/05/domain-sharding-revisited/>
- [10] Technologies, O.: OrientDB Manual - version 2.0. 2015.  
URL <http://orientdb.com/docs/last/>

# Příloha A

## Obsah CD

- Složka `src`
  - Zdrojové soubory aplikace.
- Složka `dist`
  - Přeložené balíčky apletu a desktopové verze aplikace.
- Složka `libs`
  - Knihovny potřebné k přeložení.