Tallinn University of Technology
Department of Computer Engineering
Chair of Computer Engineering and Diagnostics

Artjom Kurapov 020655

IAG40LT

Agile web-crawler: design and implementation

Supervisor: Tarmo Robal (M.Sc)
researcher

Tallinn 2007

Author declaration

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Date:

Author: Artjom Kurapov Signature:

Lõputöö ülesanne

Kinnitan:_____

dots.Margus Kruus

Üliõpilane:_____Artjom Kurapov_____matr.nr_____020655_____

Lõputöö teema:_____Paindlik web-crawler: disain ja rakendus_____

_____

Lõputöö teema inglise keeles:_____Agile web-crawler: design and implementation_____

_ _____

Teema päritolu: Arvutitehnika instituut

Juhendaja: _____TTÜ, tehnikateaduste magister Tarmo Robal

(töökoht, teaduslik kraad, nimi, allkiri)

Konsultandid:_____

Lähtetingimused: Veebisaidi eksiteerimine avalikus veebiruumis ja selle võimalus selle indekseerimiseks crawlimise meetodil.

Lahendatavad küsimused: Avaliku veebi automaatne indekseerimine ning sellelele vastavalt struktuuri tekitamine.

Eritingimused: Lõputöö on kirjutatud inglise keeles.

Nõuded vormistamisele: vastavalt Arvutitehnika instituudis kehtivatele nõuetele

Lõputöö estamise tähtaeg: “____” _____

Ülesande vastu võtnud:_____kuupäev:_____

(lõpetaja allkiri)

Registreeritud kaitsmiseks:_____

(üldjuhendaja allkiri, kuupäev)

Acronyms and abbreviations

AJAX	Asynchronous JavaScript and XML method of making a request to the server without page refresh.
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CSS	Cascading style sheets. Additional descriptors of html constructions.
CSV	Comma separated values. Simplest format for data export and import
DNS	Domain Name Service/System
FIFO	First-in-first-out. A simple queue order.
IP	Internet protocol
LAMP	Linux, Apache, MySQL, PHP platform
MVC	Model-View-Controller is a method of dividing code to several layers.
pph	Pages per hour. Measure of crawling speed
pps	Pages per second
RSS	Really Simple Syndication. An XML W3C specification for data news exchange format.
TB	Measure of data size, 1024 GB.
URL	Uniform resource location. Consists of protocol, domain, port, path and query.
wget	Data retrieval tool from servers under GNU project

Agile web-crawler: design and implementation

Abstract

The object of this research paper is the development of the web-crawler software component which is able to map Estonian Internet domain structure, and index its web-page source. This work consists of two main parts. In the first part, the author explains algorithms and various ways the web-crawling can be implemented, its advantages and disadvantages. And in second part, design and programming of the web-crawler are reviewed with certain data-processing and visualization functionalities as well. Program agility must make it possible to index and show not only one domain but defined areas of sub-domains as well.

Work is written in English

Гибкий web-crawler: дизайн и реализация

Аннотация

Целью данной работы является обработка данных эстонского интернета при помощи специально разработанного программного обеспечения, использующего исходный код страниц. Работа состоит из двух частей. В первой части я рассматриваю возможные алгоритмы, их плюсы и минусы. Во второй части я описываю разработку действующего опытного образца с особенными функциями обработки и визуализации. Гибкость программы должна позволять индексировать и отображать не только конкретно взятый домен, но конкретно взятую область доменного пространства.

Работа написана на английском языке

Paindliku web-crawler: disain ja rakendus

Annotatsioon

Antud töö põhieesmärk in Eesti Internetiruumi andmete töötlemine, kus andmeobjektiks on lehe lähtekood ja indekseerivaks vahendiks on rakendatud web-crawleri komponent. Bakalaureusetöö koosneb kahest osast. Esimeses osas vaadeldakse võimalikke algoritme veebi indekseerimiseks, nende eeliseid ja puuduseid. Töö teises osas on kirjeldatud süsteemi tegelik realisatsioon ning selle erifunktsioonid. Kõnealuse indekseerimisprogrammi paindlikkus seisneb tema võimekkuses indekseerida ja näidata mitte ainult ühte domeeni, vaid ka määratud alamdomeeni.

Töö on kirjutatud inglise keeles.

Acknowledgements

This work could not be initiated by me without proper experience gained in two companies that I worked for, Mikare Baltic and Web expert.

I received a good gesture from Moritz Stefaner of Interaction Design Lab at University of Applied Sciences Potsdam, that provided Relationship browser based on Adobe Flash for this work. I am thankful to Tamara Munzner of the Department of Computer Science at University of British Columbia for her lection, which showed how large data can be effectively visualized.

Table of contents

Table of figures

Introduction

The Internet has over 90 million domains [1], over 70 million personal blogs [2] which are viewed by over 1 billion people around the world [3]. In this vast space of information it is important to create order even without global scale. This may be done either by building classification catalog or a search engine. Both of them require a web-crawling tool to ease the burden of manual data processing. How it can be built is a question this work raises.

Deciding architecture scale is the first step. Data management functions, such as link extraction, can be divided among system components – crawler, indexer, ranker, lexicon due to robustness requirements. For example Google has over 200 000 servers [4], and dividing processing and network loads is essential for building fast and reliable computation processes.

The way crawler interprets data and how it is used further leads to the following types:

- Knowledge focused crawlers classify page by topic, extract only relevant data, use data mining filters and interact with data more like humans do.
- Freshness crawlers extract data fast and regularly, analyze what has changed, adjust to update frequency. This is used by news and RSS feed crawlers.
- Experience crawlers focus on making full document copy and storing it for a long time. A good example is Internet archive [5] and backup services.

In practical part of this thesis, a general crawler is designed, which incorporates basic methods from these types, but its main aspect is the agility if offers for the developers. This data extraction tool, can be used on most of modern hosting platforms. For source storage - a single MySQL database [6] is used and entire crawler application source is written in PHP [7] server-side language which is run in CLI mode on Apache server [8].

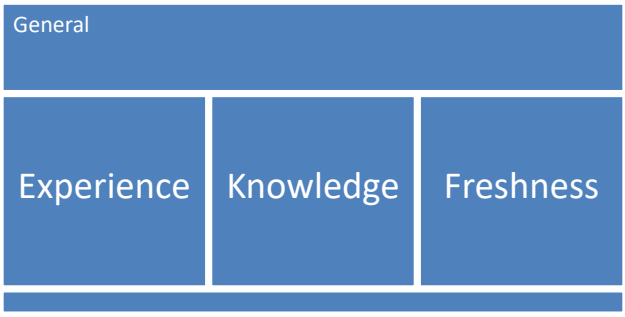


Figure 1. Crawler areas of expertise

Although market offers many services that index, order and provide data, the author found few solutions that were available with open source and under LAMP. Data extraction tools are needed more than ever and not only with indexing a page, but with tracking changes, exporting data to other applications, analyzing an existing web-site. That is why the author has decided to make own program which would be available for changes to open public. Examining different algorithms should clear out its purposes and what is the most suitable for this work.

1 Graph data management

From mathematical point of view, Internet can be seen as a large directed graph, where nodes are resources and connections between them – links, otherwise known as URLs. This graph can also be seen as a tree data structure by using domain name system and file system as media, where sub-domains and sub-directories are considered as different levels of this tree.

Crawling is a process of mapping dynamic graph and storing node data. This process is done by examining a node, finding edges, and repeating the process with nodes that are connected to this node. This process cycle is infinite if no tracking is made or if process has history of viewed resources, but has revisit policy as well.

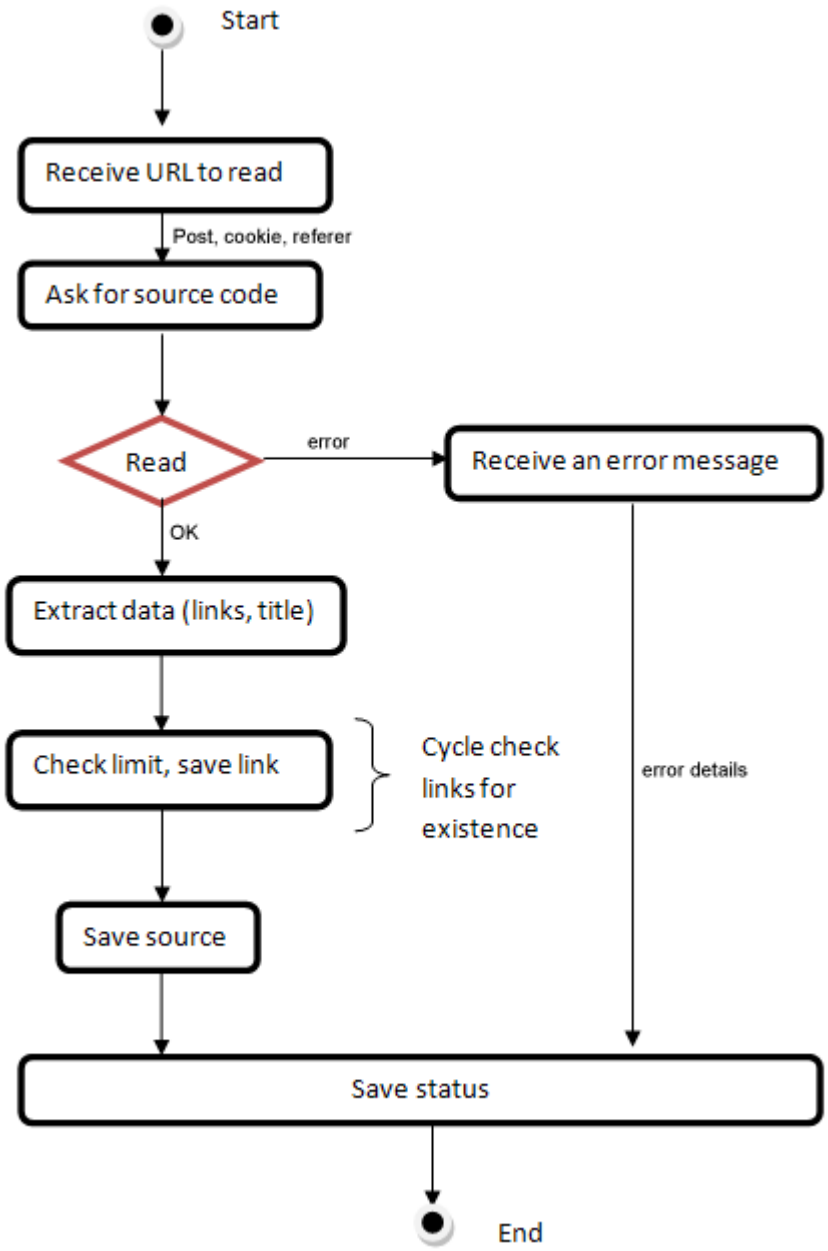


Figure 2. Crawling cycle functional diagramm

For selection policy and search engine result ordering the graph must be ordered. This can be done by analyzing general graph structure, assigning order with human aid, adding knowledge discovery algorithms, tracking popularity of web-sites. Some of them are explained further.

1.1 Page ranking algorithms

Ranking a web-page or entire domain is needed to set a value of its importance. Automatic ranking cannot do what human does, since it has no real interest or field of research and that is why importance is evaluated as popularity.

In global scale, popularity cannot be measured by making a counter of visits for every page, because this cannot be solved technically without proper agreement with web-site owner. Although many counter services have been offered, they are all voluntary. The major disadvantage is sharing of web-site privacy policy with the statistics service.

Ways to analyze popularity based on indirect facts are used in modern search-engines.

1.1.1 Importance of domain origin

An automatic consideration of domain root has a good impact on strategies not only involved in revisiting policy (.edu and .gov domains are less frequently updated than .com [9]), but also in search query relevance computation, because web-site geographical location matters if user is from the same region, language or culture.

That is why a crawler may want to inspect data, such as

- WHOIS information
- page original language
- server geographical location

For example Hilltop algorithm [10] ranks links to external domains with higher grades if they are geographically closer to the source. As example, domains from the same university network have higher page rank than pages from mass-media from the same region.

1.1.2 Evaluation of site structure

Domain system created for ease of use by humans and tied with IP resolving, serves now a more commercial role than before - a three-letter domain is a valuable resource and a good example of how popularity affects ranking.

Page level in domain hierarchy is the number of links user has to click from the root page. Higher level pages are easier to access. Pages, which cannot be accessed by means of hyperlinks, are *robot-isolated.* Pages which are visually not accessible for human observer are *human-isolated* pages.

Considering page location as an independent file and how deep it is located in URL (divided by / symbol) takes place from old ways, when html files were static. This has no effect now, since dynamic server-scripting is in effect and web-engines try to use one gateway (index.php file for example).

Ranking a page based on its level comes from a psychological assumption that a random user is most likely to type short domain address than entire and exact path to the required document. By doing so it is possible, that more related information will be found in the process.

Page level cannot be speculated though – like in every hierarchy, a higher level element is more probable to have relevant information the user seeks, but at the same time, due to higher abstraction level, it cannot give more specific details. It will also be noted further, that pages with higher level tend to update more frequently.

1.1.3 Calculation of page weight

Page weight calculation is one of the ways to measure its popularity. Since every web-site fights for visitors, calculation algorithm must be smart enough to distinguish documents made for engines from those that are made for visitors.

Page rank [11], Hypertext-induced topic selection [12] , On-line Page Importance Computation [13] , Topic Citation Index [14] are algorithms, which are focused on creating **labeled weighted graph**, where each vertex and edge is associated with a value or several values, like width, weight, length. Vertex weight is calculated based on the number of incoming and outgoing edges.

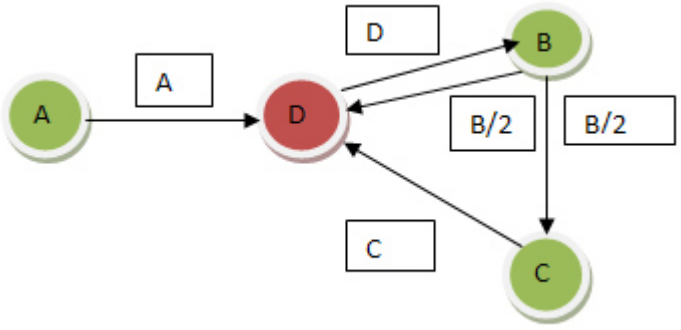


Figure 3. Page rank computation example

Every algorithm has its differences; further the author explains how simplified Page Rank is done [15]. Consider there is a small graph (figure 3) with nodes which have some variable ranks (named after vertex names). Every directed edge this way would give a surplus of its weight to a vertex it is pointing to, ergo vertex B has some page rank B that is equally divided between all links.

It is clear that if A gives entire page rank to D, then A would have 0 page rank, which is not exactly a right thing, that's why a limiting factor was introduced – *damping factor*, which basically states that some part of vertex weight stays with the source. This can be seen as observer's physical limitation of viewing links with unlimited deepness and statistically is introduced as 15 % [11].

The calculation of A,B,C,D real values can be done iteratively and be represented as a matrix [16]. In the calculation process new page rank does not depend on its previous value.

The main one-iteration simplified formula for calculating page rank for vertex A is

$$A = (1-d) + d*\left(\frac{B}{L(B)} + \frac{C}{L(C)} + \frac{D}{L(D)}\right)$$

Iteration process is shown on figure 4.

$$R1=\begin{bmatrix}0.15\\0.3625\\0.25625\\1.625\end{bmatrix}\quad R2=\begin{bmatrix}0.15\\1.53125\\0.3040625\\0.649375\end{bmatrix}\quad R3=\begin{bmatrix}0.15\\0.70196875\\0.80078125\\1.186734375\end{bmatrix}\quad R4=\begin{bmatrix}0.15\\1.15874219\\0.44833671\\1.38333672\end{bmatrix}$$

Figure 4. Page rank matrix change with each iteration

Page that has no incoming link has eventually page rank 0.15. New pages that get indexed automatically gain weight of 1. With each iteration Page Rank normalizes, yet the differences in rank values of different pages is on logarithmic scale.

A more simple way to order graph, is to consider tree structure of the web and to use sum number of incoming links to the current page from different domains. Major disadvantage is a possibility that someone decides to create a link farm on several domains.

1.2 Human-aided ranking

Automated ranking systems were developed because voting by human operators costs a lot of time and money. Web-sites which collect links to internet resources and order them by popularity and topic have been made:

- Social encyclopedia (Wikipedia.org) makes it possible to edit any article by anyone. A good example of human-aided information submission.
- Social resource ranking (Digg.com) grants any user a possibility to vote for any web-page
- Social bookmarking (del.icio.us) acts the same way as resource ranking, except that it motivates public to save links as favorite pages.
- Social games like ESP game [17] sees Earth as huge processor, where every human brain is considered as a cluster. This makes it possible to make complex data processing while sustaining human stimulation of doing so.
- Social catalogs are perhaps the thing of the past (yahoo.com, Dmoz.org), where each topic is filled with links to resources by appointed public moderators.

Major disadvantage of hand-made data processing is not even its cost, but trouble with subjective judgment of every human operator. Wikipedia has to deal with lots of vandalism actions and views presented by opposing political, cultural and other forces.

1.3 Scheduling page visits

Assuming that every internet page has not only subjective value but objective informational value (facts, references etc.) as well, every crawler need a scheduling algorithm, which would give high priority queue on indexing most valued pages.

Crawler is a part of search-engine or other info system that has to request information first, and then manage it. Similarity with astronomy is very close. The observer first has to choose what part of the sky he wants to observe, then questions of revisiting, politeness and parallelization start to form as policies.

The same it is with the Internet. It is hard to choose what pages should be travelled first, hard to distinguish how important they are, what is their age and how frequently they change, how often to visit them without raising questions with server overloads.

Starting with a seed, crawler, if not being limited, will eventually find more and more links. Depending on the pages-per-domain limit, this ratio may vary around 3 new links per page and considering virtually unlimited (over 100 billion pages) Internet galaxy, it is critical to use the right algorithms to receive right information in time.

1.3.1 First-in-first-out frontier (aka Breadth-first)

Considered the most simple and reliable, FIFO algorithm looks through documents that were added first, creating a queue of URLs to look through. In case repository is full and no new links are found, FIFO algorithm tells crawler to look through oldest documents. This makes sense in general, since otherwise some kind of ordering, planning and analysis is required, which is often not processing-friendly.

Advanced FIFO algorithms include increased attention to time-modified response in HTTP request header and to weights of the graph node. Simplified, this makes front pages of the domain to be updated more frequently than deeper pages. Though some domains can be considered subjectively more important and may gain certain weight by overriding general rules. This is often done in mass-media and blog services, where quick partial indexing is more important than full sweep.

1.3.2 Context-dependent frontier

A simple FIFO crawling algorithm does not take into account link context, as human does. For human eye text with small font or low contrast is not considered important. On the contrary, blinking images, bold text or text headings focus observer’s attention. And attention means that this part of document has higher value, higher probability for links to be clicked.

1.3.3 Weight-focused frontier

Page-rank based selection and uniform-cost search algorithms use page weight to evaluate what pages have priority to be crawled more often. Some weight value is considered to be equal to frequency value. The conversion is done by evaluating size of the entire database, sum of all weights and estimation of how quick crawling is done.

1.3.4 Harvesting entire domain

Since domain has a tree-structure, its source can be acquired in several ways. Using these algorithms in general way of indexing pages without paying attention to selected domain is not a smart move, since some of them focus on specific areas of crawling.

- **Breadth-first search** (Figure 5) focuses on indexing pages as they appear on the first levels of the page. Each link is inserted in the queue to be traversed later. This is the most simple and obvious way of differentiating pages by deepness, as user has to follow several pages to it. A limitation can be either stopping at certain level, or/and at certain number of pages or may implement **beam-search** algorithm by limiting number of pages to be evaluated on every level.

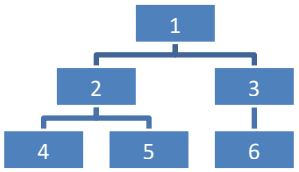


Figure 5. Breadth - first search example

- **Depth-first search** (Figure 6) relies on the need of quick indexing of deep resources. Though it does suffer from the problem of infinite nodes, often produced by auto-generation of pages, it has its advantages of simulating human behavior by visiting documents which lie deeper, as human would do if he knew what he looks for. A solution is to declare maximum level of traversing.

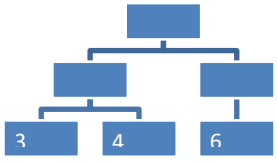
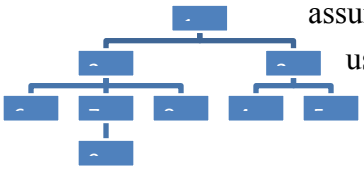


Figure 6. Depth-first search example

- **Reversed uniform-cost search** (Figure 7) has graph weight attribute associated with each vertex. Root vertex is given maximum weight, which is equally distributed among child vertices recursively. Vertices with maximum weight are crawled first, since algorithm assumes that user does not know what information he needs, or where it is located, due to misleading structure of the web-site, because of what high accessibility pages are viewed first. As opposition to this

Figure 7. Cost-first search example



assumption, crawling through minimum weight pages assumes that user knows exactly what he looks for and where it is located.

- **Best-first search** assumes that crawler has heuristics function that evaluates all relevant data and decides which link to index. This may include link ordering by its location in the document, Bayesian statistics calculation that page itself is not used to fraud crawler, using global page weight (back-links), even **randomization** of links, incorporation of other algorithms etc.

1.3.5 Focused crawling

Major disadvantage of algorithms listed above is that selection policy has only one purpose – to cover as much documents as possible, even if they are the same. Focused crawling is different – its selection policy depends on a set of dictionary keywords. The page which keyword subset is found to coincide statistically with the dictionary is accepted. This method is also known as Bayesian filtering and is used in email spam distinction.

Some focused crawling algorithms [18] update dictionary with found top keywords, increasing its agility, while others [19] use a database of indexed resources (search engines like google.com) to find pages that link to this document and assume that they have topic in common. Event when focused crawling is not required as main strategy, it may improve general crawling. Trust rank [20] algorithm focuses on finding pages and links, which should not be crawled.

Although focused crawling has good perspectives, since relevance is very important in search engine life, it has its drawbacks – spam filters require training.

1.3.6 Constrains of crawling process

Crawling limitations have already been listed as different approaches to indexing strategy. Every indexing algorithm though may require additional limitations, due to limited crawling software storage and bandwidth resources. Here are some of them:

1. GET parameters. A lot of modern Internet is done using scripting language like PHP, JSP, ASP. Static HTML is long gone, yet indexing dynamic parameters as different pages may lead to

infinite combinations that will fill up limited memory space. Accepting GET parameters is under question, but deleting session and random identifiers are required with no doubt.

2. Page count. Limiting page count to 15 pages per domain is a hard solution to make – it is hard to tell how big the web site is and how important information there may be. One of the solutions is to make page count dynamic, depending on the domain popularity/weight/rank.

3. Level deepness. In conjunction with page count, this may prove a good way of balancing information quality per site. The deeper information lies, the few users will reach it.
   Since domain can have a tree of sub-domains which affect ranking as well, some second-level national domains can be seen as first-level – co.uk and pri.ee.

4. MIME filter [21]. Downloading page that is not in plain text is like listening to foreign language – the system can say how big the data is and what characters can be found, but without proper software for sound, video and image recognition, there is no use of it.

5. File size. Web-pages over 500 KB at the present time are considered heavy, since statistically average page size (as will be shown further) is around 16 KB.

6. Server timeout. Statistics shows [22], that in average standard page request does not take longer than 2 seconds. Configuring it to 25 seconds may slow down crawling several times.

1.4 Parallelization policy

Once crawling and database is run in industrial way, parallelization is inevitable in order to overcome timeouts from fetching pages that are far-far away or do not exist at all and in order to increase processing speed and bandwidth.

Centralized way of implementing parallelization is to use a front-end server, which will manage all processes, resolve DNS names and pass them on to processes that await response. Obvious problem occurs if central planning server is down or is unreachable; otherwise it is a good way of managing database without creating repeated requests of the same URL from multiple processes. A simple example is to store a number of threads and assign each thread its number, which would affect the URL to be fetched.

Decentralized processes manage entire cycle on their own, though they must have some static way of self-management to minimize same URL re-fetching. In this work the author uses selection policy with randomization algorithm, which makes sure that probability of requesting the same page by two or more crawling processes is minimized.

Parallelization is not only required to increase processing speed, but also is a good way to combine different strategies of search engine. These strategies may include resource type focus, independent language and revisit policies [22].

1.5 Steps after crawling is done

Judging by modern average-class IT business requests [23], it is clear that solutions nowadays need simple data extraction either from a single domain, or a group of predefined pages. Source not only has to be extracted, but parsed in a manageable database for further use.

The “dark-side” of the business which is connected with crawling includes email harvesters, spam bots, text and image recognition software. Managing information in the way that it does not go to the wrong hands and does not harm anyone is an important issue, taking in account that google.com has been used to hack web-sites and how much attention it spends on pornography filters.

1.5.1 Search engine from the inside

After storing page source code in the database and finishing its role, crawling is ended and next components (indexer, lexicon etc.) start their work. Indexer parses HTML source code and extracts pure data with accommodation parameters like font size. This text is evaluated by natural language processing component and every word is inserted and tied in the inverted index structure, joining words with where they were found.

Data structure size may reach million words and have its semantic network. Because information size grows along with the number of online users, scaled search process is twice as hard (over 100 million requests a day [3]). That is why every search result has to be cached for some period of time. Caching is typically done if request has not been done before. This eliminates huge processing load on the servers.

1.5.2 Transforming data into information

According to Alex Faaborg of the Mozilla foundation [24], future browsers will likely evolve into information brokers, helping humans in their everyday life. Semantic-web agent will have more rich access to resources on the net and on the desktop. Joining services with single API is either an idée fix, or a panacea, which still continue to develop.

W3Consorcium states [25] that for knowledge management, web masters can use RDF with OWL support which can describe entities, connections and properties (figure 8). But its development along with SPARQL [26] is so specific, that most of modern solutions are still based on XML, RSS and SOAP, and market chooses itself what technology is better.

Microformats [27] is another solution to exchanging data among different applications. These constructions are embedded in HTML source, have open API and may be readable by human observer as well. Microformats cover different topics – from curriculum vitae to genealogy, and are developed by independent authors. The disadvantage is that each format has its own structure and data extraction is hard to implement for all of them in one application.

Another speculation is that global search engines will likely evolve to Artificial Intelligence [28], based on text recognition at first, reaching to image, sound and video. A web-crawler in this context would probably be used as a sensor, connecting real-life world with first layers of AI.

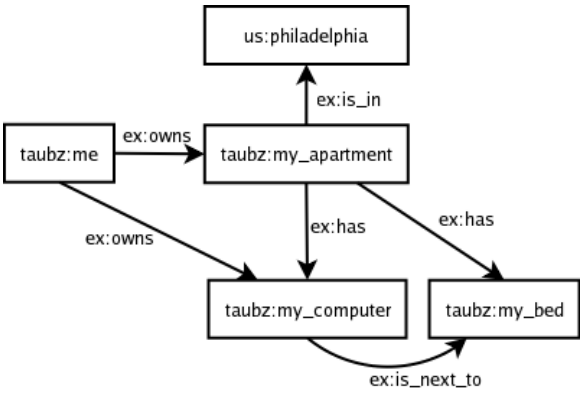


Figure 8. RDF as graph. Source: xmlhack.ru [29]

2 Web-crawler design and implementation

For motivation and ease of further development general requirements were specified:

- Agility. A crawler that is easy to modify is crucial to extend system quickly, without losing time on modifying entire application. This also includes ease of integration in other systems and data exchange with applications that may require access to the database for visualization or other needs.
- Robustness and speed. Crawling has to withstand anomalies that occur with certain pages, domains or connections and must be optimized so that cost-per-performance would be as little as possible. Speed increase in the future should be done using distributed processes. Once limitations are specified, seed links can be added and crawler should start its cycle, boosted by multithreading system. The number of simultaneous threads is available for user to change.
- Manageability. Configuration is not written solely in code - it is dynamic and user may change limitations, crawling etiquette and activity. Crawling cycle console is also provided for user to monitor process status. Crawler user interface should be simple enough to specify limitations of the crawling method; either it is entire domain, part of the domain or domain with some parts of external domains.
- Structure analysis. Web crawler not only indexes the web, it also tries to build a domain structure – how deep every page is located, what response the server sent, what are the 404-pages. This analysis should be interesting for the web-site managers, since they often cannot see the big picture.

- Data extraction. Crawler user interface provides a possibility to add filters. Every filter is used in crawling cycle to find data, matching its criteria. Data is then saved and is possible to save independently of the source.
- Search system. Searching among the indexed results is not as complicated as a search engine system. This option is only an example of possible extensions of a simple crawler.

Similarities can be seen with other works [30], which focus on the same topic.

2.1 Used design patterns in crawler programming

Design patterns are methods in programming, which make life easier by decreasing time spend on writing similar portions of code while sustaining readability and ease of usage. They are independent of the programming language, but in some cases cannot be implemented because of architecture structure.

Active Records [31] is a method of creating objects (figure 9), each of which correspond to database table. An object inherits functions to manage information stored in table. This way finding required row in database can be simplified and reused – no need to manually write SQL queries, writing over and over routine functions.

```
//Save spider
$updSpider->ID=$spider->ID;
$updSpider->date_updated='NOW()';
$crawler_spiders->update($updSpider);
```

Figure 9. Active Record example in code.

MVC pattern (figure 10) separates processing logic in different parts, scaling entire project by making it easier to reuse, integrate and extend. Model in other applications is usually a written class, which has access to database, while in this project case it is an active record object. Controller is a class, which has processing functions and variables, which in this case are primarily focused on crawling and user interface. As a view which deals with user interface, Smarty template engine is used, which has also caching possibilities along with html generation.

Figure 10. MVC as a data process from DB to the user

Database abstraction layer is often used in agile products, since they are often ported to other databases. A db class was written, which communicates through mysql class with MySQL database, but, depending on configuration should be able to interact with Postgre and Oracle.

2.2 Code architecture.

Crawler platform is based on PHP 4/5, MySQL 4.3/5.1, Apache 2, optionally with crontab daemon support. Crawler code, user interface and database system (further - system) also uses open source software products:

- Smarty template engine 2.6.18
- nncron 1.91 [32]
- EasyPHP 2.0
- Prototype.js 1.5 library
- Silk icons [33]
- Relationship browser [34]

System is based on several classes and objects (figure 11):

1. Database class implements access to MySQL database and acts as an thin abstraction layer. Class has functions which generate SQL queries, such as SELECT, UPDATE, INSERT and DELETE. Class is later inherited by crawler package through system class.
2. System class implements forwarding a request to package object, since system may have multiple packages, it also deals with loading package one within another, managing package default inner variables and general package initialization.
3. Package class in this case – “crawler” implements entire functionality, which concerns user interface, parsing, cycles and saving data. Crawler has primary inner objects called models, which act as a database table objects and inherit database class. Any function of crawler class in order to gain comfortable usage of them, must access them through *extract*() function.

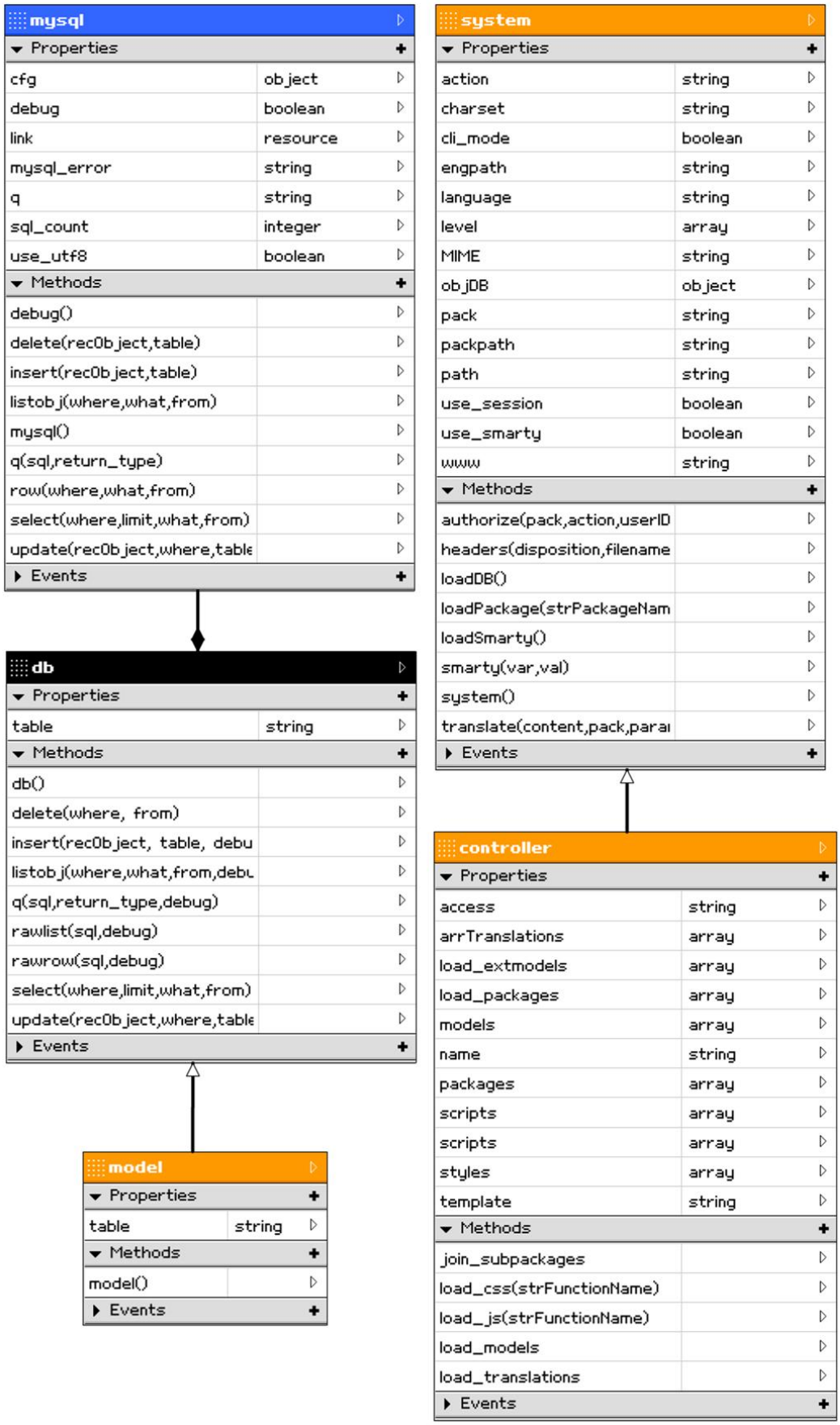


Figure 11. UML class diagram of the platform core. (Drawing done with gmodeler [35])

2.3 Data entities

Defining information system sets is an important step in order to move on to user interface design and database design.

Crawler system main data sets are:

- Domains
- Links
- Pages

Why not just one “pages” set? Certainly in a more abstract way, Domain with its Link may become a Page element, but separation is needed, to gain processing speed and memory reduction. Domain has n links (n>0), and link has 0 or 1 pages.

Additional data sets include:

- Spiders
- Filters
- Resources
- Threads
- Referrers

A spider set is an abstract crawler entity, needed for user-interface simplifications and configurations. Such virtual crawler has configuration variables that describe limitations. Every domain is tied to this spider, thus spider is the most general data set.

Filters and resources are the subsets of spider and are needed only for data extraction routine.

Threads and referrers are a technical subset required for multithreading robustness and links cross-references.

2.4 Database design

Having data structures known, creating database tables (figure 12) can be done with phpMyAdmin [36], Navicat [37] or MySQL Workbench [38].

Tables have relations through PRIMARY KEYS, although use of FOREIGN keys does have its advantages, the author uses MyIsam table architecture and try to avoid relation mapping in order to increase processing speed. All frequently used keys have indexes.

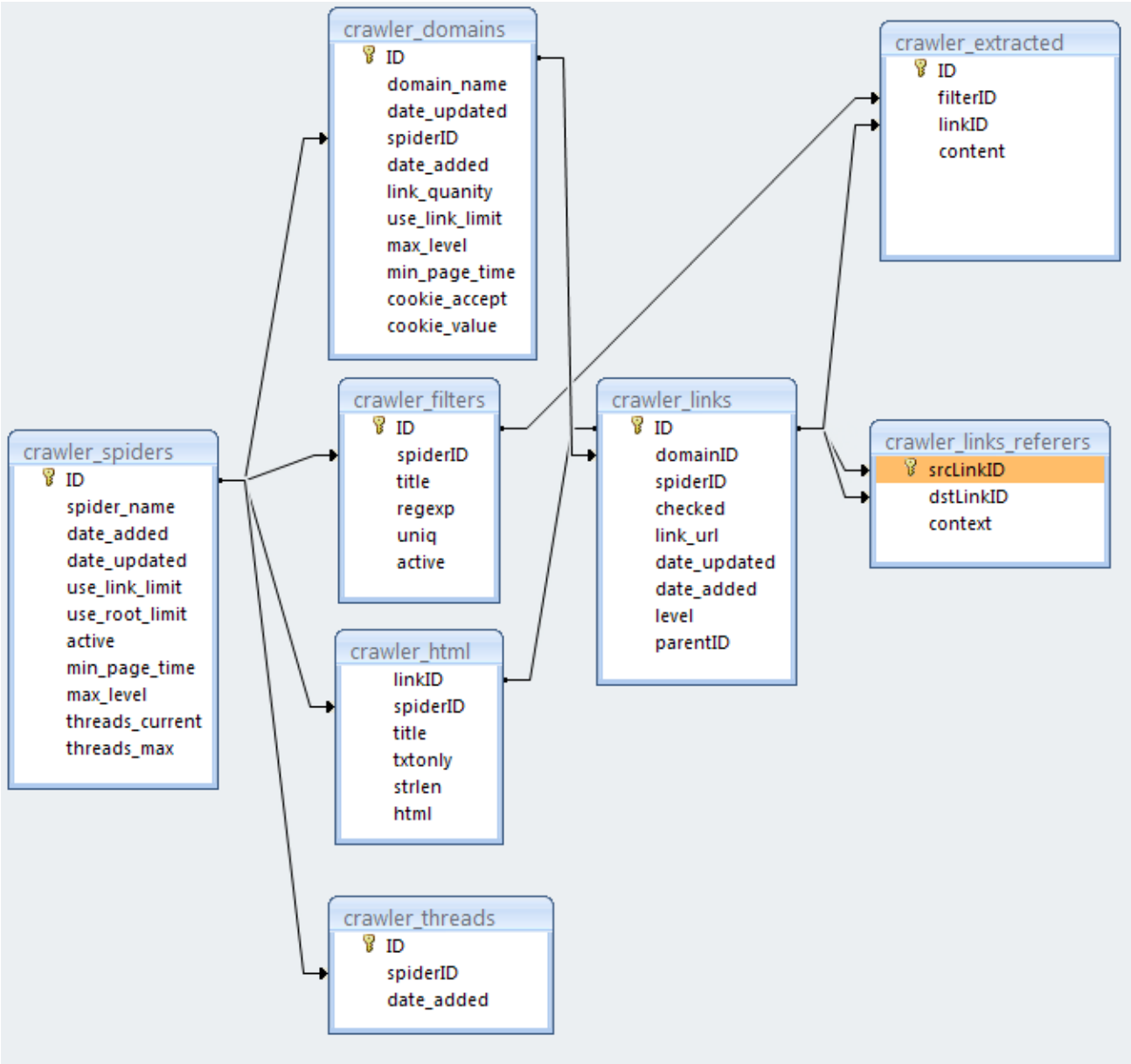


Figure 12. Crawler database structure and relations

In order for full-text search to work, html source is cleaned and remaining text is saved in crawler_html.txtonly. This eliminates problems with searching through html tags, comments, CSS and script code.

2.5 Crawling cycle in practice

After crawling function start, process loads limitation variables and goes through the following stages:

1. Fetching URL to read. To avoid collisions between different threads, link is fetched by picking random link that was not visited. Current implementation does not have thread specialization, and crawling is done only one time, without revisiting crawled pages.

2. Requesting page is done with specified parameters like timeout, cookie value, filesize limit. The function also supports POST request, which is planned for future, and should be specified by user.

3. Receiving and parsing response. Headers are analyzed while content is still being fetched and depending on file size limit can skip this step. Response status and encoding are extracted.

4. If response status is not 200 OK, link is updated but no content is saved, process ends. Otherwise depending on the encoding, source is transformed in UTF8. Page is parsed, links, title and other filter data are retrieved. Each link is normalized and checked for existence in database. Page source and links are saved.

In order to speed the process up, function *multithreads* is called through PHP CLI (figure 13), which does not respond to client and does all processing on client side. This is a better solution than starting a thread by means of wget or ajax.

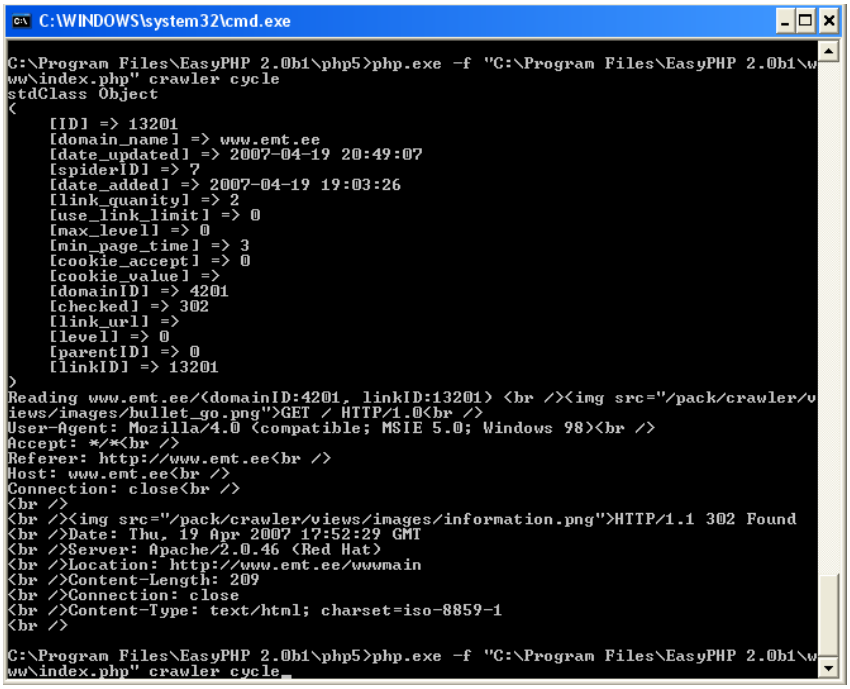


Figure 13. PHP Command-line interface as a thread

2.6 User interface and use-case

User interface is simple enough and consists of two types – list and view. List UI contains listing of some data, while view type may provide changing data possibilities. Most of user interface functions have “ui_” prefix in crawler controller class, which makes it easier to focus on what functions are made for.

Interface functions include:

- Spider listing / management
- Domain listing / management
- Link listing / source management / sitemap export [39] to XML
- Filter listing / filter management and testing / extracted data export to CSV

A typical user-interface interaction would be:

1. User adds new spider, enters crawling limitations, where most important is “links per domain” number.
2. User adds filters for data extraction, if any are needed.
3. User goes to domain listing, clicks on “Add” button and adds new seeds for crawling cycle
4. Triggered system crawling cycle reads pages and graph expansion begins, new domains and pages are added.
5. User checks domains of interest, extracts structure or source information, uses search if necessary.

The main page offers a list of spiders, whereas each crawling process is shown and dynamically updated using AJAX (via prototype.js library). A ratio shows how much pages were indexed comparing to the number of links. Reaching 100% will mean the end of crawling process. Speed is calculated in pages per hour and at max reaches 6000.

Below the spider list, a console of the process is shown – what page is currently evaluated and what response is given.

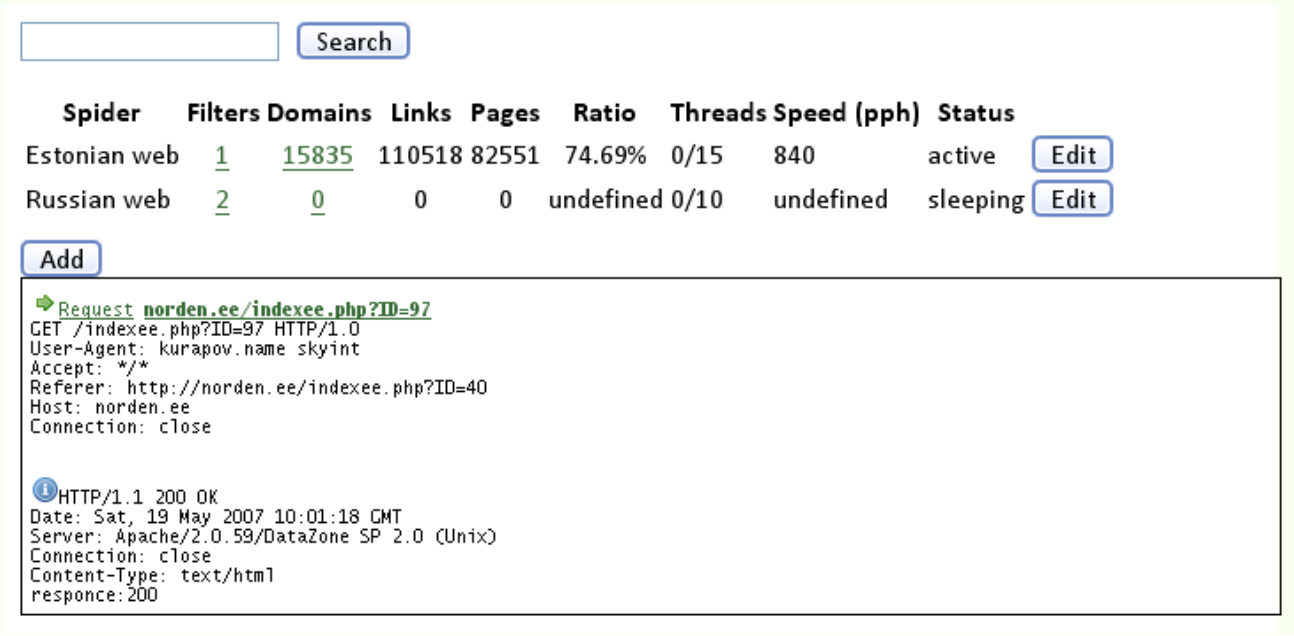


Figure 14. Main page user interface

2.7 Statistical results of made crawls.

Crawler development was done not as a single package, but was developed step after step, where each iteration was different from the other not only in features it had, but also in hardware, scripting language and database versions.

First crawling cycle used php 4, mysql 4, lasted 1 hour on Intel Xeon 2.2ghz with 100mbit channel as single thread, with javascript iframe meta refresh with delay of 1 second scored:

- 284 domains found
- 1521 pages found (graph nodes)
- 735 pages indexed with pure text of 1.2 MB
- 6944 links (graph edges)

Assuming that the web will reach 100 billion web-pages in near future, total indexing with this kind of approach would take 15 thousand years, still – indexing one web site takes satisfactory time.

Gaining speed can be done two ways – process parallelization and code optimization.

Multithreaded system is tied closely with spider configuration. Every spider has predefined limit of threads that can be run. Threads can be run chronologically by special system daemons, such as crontab in Linux, or nncron on Windows, or simply by client HTTP requests from browser.

Multithreading is done by running multithreads() function that has 10 minutes of processing time limit. Multithreads function calls in circle function cycle(), which does all the work.

Second statistical run used php 5, mysql 5 and lasted 1 hour on Intel Core 2 duo with 2 mbit channel and had the following results:

- 2931 domains
- 1158 pages
- 8430 links

Third statistical run lasted 12h and gathered were:

- 15283 domains
- 101079 unique links
- 69391 pages
- 251527 cross links
- 1114 MB of data

Additional interesting results were found:

- 16 KB per page
- 3-4 outgoing links per page
- 90.05% Pages found (200 OK)
- 5.5 % Redirect (301, 302)
- 4.6 % Not found

Additional analysis was made, which counted number of incoming links from unique domains. Most linked-in were:

- www.eenet.ee (132 unique domains linked to this domain)
- www.cma.ee (59)
- www.hansanet.ee (48)

The major disadvantage of this analysis though is that the system did not recognize domain structure and because of this, third-level domains are counted as unique. Because of that, this kind of ranking doesn't truly reflect popularity of the domain.

Comparing to other software implementations - 48.5 pps in Google [4], 112 pps in Mercator [40], 400 pps in AlltheWeb [41], statistical run was made with maximum at 2 pps which is due to usage of PHP instead of c++ and because of severe network bandwidth limitations.

2.8 Optimization and future work

Code optimization is done by finding bottleneck areas of code that need optimization. In order to do this, entire process needs to be devided by time markers ($arrTime), which should tell us at the end of the process what part of code took the most amount of time.

This way, for example it was found that

1. Using randomization of links algorithm in the part of fetching URL to be read, the process takes a large amount of time to process, because SQL query to MySQL server takes too long to process, since ORDER BY RAND() statement needs to order entire table that is growing all the time, and by 50 thousand rows it takes over 1.5 seconds.

   An alternative variant [42], which generated random link without ordering entire table, but by fetching ID as random number, took 0.00086 seconds, boosting speed of from 1000 pages per hour to 4000 pages per hour.

2. Usage of session_start() in general platform brought reduction of speed of 0.5 seconds and was disabled for CLI mode.

3. Bottleneck area of the crawling process stays network connection and database queries. Network optimization can be solved by using high-bandwidth channel and database query slowdown is inevitable when tables grow. This can also be solved by more precise database structure, hard-drive defragmentation and more powerful DB engines.

As the result, architecture development for the future should involve:

- PostgreSQL [43] support.
- SPHINX [44] full-text search engine support and page weight calculation

Conclusions

Developing web-crawler is not innovative nor it is important in establishing healthy business structure – it is a tool that can be written, bought or used from open sources. The author developed this tool from scratch, so that it would be easier for others to continue this work and make it more specific.

Architecture in this work is on medium scale because being developed, it has not came under the influence of single selection policy, which means it is still abstract enough to modify. On the other hand it lacks its specialization, which would give speed and data processing surplus.

The software is able to crawl a domain or group of domains, create its map, index and make full-text search. As a mean of focused data extraction, filters can be used to extract images, RSS, emails or other URI. Open source and multithreading possibilities should be good motivation for further development.

In the development process, the author learned new features of MySQL usage, as well as php 5 when switching was done from php 4. The author also discovered PostgreSQL and SPHINX, which are planned to be used in the future.

References

1. Whois.net [WWW] http://www.whois.net/ (13.06.2007)
2. Technorati [WWW] http://technorati.com/weblog/2007/04/328.html (13.06.2007)
3. How Much information? 2003 [WWW] http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/internet.htm (13.06.2007)
   Internet growth statistics [WWW] http://www.internetworldstats.com/emarketing.htm (24.06.2007)
4. How Google works? http://www.baselinemag.com/print_article2/0,1217,a=182560,00.asp (13.06.2007)
5. Internet archive http://www.archive.org/index.php (13.06.2007)
6. MySQL AB http://www.mysql.com/ (13.06.2007)
7. PHP http://php.net/ (13.06.2007)
8. Apache software foundation http://www.apache.org/ (13.06.2007)
9. Effective Web Crawling, Castillo C. 2004 [Online] http://www.dcc.uchile.cl/~ccastill/crawling_thesis/effective_web_crawling.pdf (13.06.2007)
10. Hilltop algorithm [WWW] http://pagerank.suchmaschinen-doktor.de/hilltop.html (13.06.2007);
    Hilltop: A Search Engine based on Expert Documents, Krishna Bharat; George A. Mihaila [Online] http://www.cs.toronto.edu/~georgem/hilltop/ (10.05.2007)
11. The Anatomy of a Large-Scale Hypertextual Web Search Engine; 1998 Brin, S.; Page, L. [Online] http://dbpubs.stanford.edu:8090/pub/1998-8 (13.06.2007)
12. Method and system for identifying authoritative information resources in an environment with content-based links between information resources. Kleinberg J.M. 2000 http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PALL&p=1&u=%2Fnetahtml%2FPTO%2Fsrchnum.htm&r=1&f=G&l=50&s1=6112202.PN.&OS=PN/6112202&RS=PN/6112202 (13.06.2007)
13. Adaptive On-Line Page Importance Computation. Abiteboul S., Preda M., Cobena G. 2003 [Online] http://www2003.org/cdrom/papers/refereed/p007/p7-abiteboul.html (13.06.2007)
14. Индекс цитирования [WWW] http://help.yandex.ru/catalogue/?id=873431 (13.06.2007)
15. Растолкованный Pagerank [WWW] http://www.searchengines.ru/articles/004516.html (13.06.2007)
16. How Google Finds Your Needle in the Web's Haystack. Austin D. Grand Valley State University http://www.ams.org/featurecolumn/archive/pagerank.html (13.06.2007)
17. ESP game [WWW] http://www.espgame.org/ (13.06.2007)
18. Focused crawling: a new approach to topic-specific web resource discovery. S. Chakrabarti, M. van der Berg, and B. Dom. 1999.
19. Focused Crawling Using Context Graphs. M. Diligenti, F. M. Coetzee, S. Lawrence, C. L. Giles and M. Gori, 2000 [Online] http://clgiles.ist.psu.edu/papers/VLDB-2000-focused-crawling.pdf (19.06.2007)
20. Combating Web Spam with TrustRank. Gy¨ongyi Z., Garcia-Molina H., Pedersen J. [Online] http://www.vldb.org/conf/2004/RS15P3.PDF (13.06.2007)
21. MIME types list [WWW] http://www.iana.org/assignments/media-types/ (13.06.2007)

22. Search Engines and Web Dynamics. Risvik K.M., Michelsen R., 2002 [Online] http://citeseer.ist.psu.edu/cache/papers/cs/26004/http:zSzzSzwww.idi.ntnu.nozSz~algkonzSzgenereltzSzse-dynamicweb1.pdf/risvik02search.pdf (19.06.2007)
23. WEB lancer [WWW] http://weblancer.net/projects/ (18.06.2007)
24. Mozilla Does Microformats: Firefox 3 as Information Broker [WWW] http://www.readwriteweb.com/archives/mozilla_does_microformats_firefox3.php (18.06.2007)
25. W3C RDF [WWW] http://www.w3.org/RDF/ (13.06.2007)
26. W3C SPARQL [WWW] http://www.w3.org/TR/rdf-sparql-query/ (13.06.2007)
27. Microformats [WWW] http://microformats.org/ (19.07.2007)
28. Novamente: An Integrative Architecture for General Intelligence. Moshe Looks, Ben Goertzel and Cassio Pennachin [Online] http://www.novamente.net/file/AAAI04.pdf (18.06.2007)
29. Краткое введение в RDF [WWW] http://xmlhack.ru/texts/06/rdf-quickintro/rdf-quickintro.html (18.06.2007)
30. Shkapenyuk, V. and Suel, T. , Design and implementation of a high performance distributed web crawler. 2002 [Online] http://cis.poly.edu/tr/tr-cis-2001-03.pdf (26.06.2007)
31. Active record in Ruby on Rails [WWW] http://wiki.rubyonrails.org/rails/pages/ActiveRecord (18.06.2007)
32. nnCron [WWW] http://www.nncron.ru/index_ru.shtml (13.06.2007)
33. Silk icons [WWW] http://www.famfamfam.com/lab/icons/silk/ (19.06.2007)
34. Relation Browser [WWW] http://der-mo.net/relationBrowser/ (19.06.2007)
35. UML Gmodeler used [WWW] http://gskinner.com/gmodeler (18.06.2007)
36. PhpMyAdmin [WWW] http://www.phpmyadmin.net/home_page/index.php (19.06.2007)
37. Navicat [WWW] http://www.navicat.com/ (19.06.2007)
38. MySQL AB GUI tools [WWW] http://dev.mysql.com/downloads/gui-tools/5.0.html (19.06.2007)
39. Sitemaps standart [WWW] http://www.sitemaps.org/ (13.06.2007)
40. Mercator: A Scalable, Extensible Web Crawler Heydon, A. and Najork, M. 1999 [Online] http://www.cindoc.csic.es/cybermetrics/pdf/68.pdf (19.06.2007)
41. Search Engines and Web Dynamics.Risvik, K. M. and Michelsen, R. 2002. http://citeseer.ist.psu.edu/rd/1549722%2C509701%2C1%2C0.25%2CDownload/http://citeseer.ist.psu.edu/cache/papers/cs/26004/http:zSzzSzwww.idi.ntnu.nozSz%7EalgkonzSzgenereltzSzse-dynamicweb1.pdf/risvik02search.pdf (19.06.2007)
42. Alternative ORDER BY RAND() [WWW] http://jan.kneschke.de/projects/mysql/order-by-rand (26.06.2007)
43. PostgreSQL [WWW] http://www.postgresql.org/ (13.06.2007)
44. Sphinx Search [WWW] http://www.sphinxsearch.com/ (13.06.2007)