Design and development of an automated web crawler used for building image databases

Y. Kalmukov*, I. Valova*
* University of Ruse/Department of Computer Systems and Technologies, Ruse, Bulgaria
ivalova@ecs.uni-ruse.bg, jkalmukov@ecs.uni-ruse.bg

Abstract – **Every day people worldwide upload millions of images to social networks, personal blogs, community forums and other web-based applications. To increase impact and public popularity however all these images should be indexed by search engines. Building an efficient non-textual search engine is far from a trivial task. It should employ modern information retrieval and image processing techniques to extract, index and store proper metadata from images that allows fast subsequent processing and searching.**

Before any image processing to apply, the search engine should be able to find and process the large amount of data being constantly added to the Internet. The WWW represents an enormous directed weightless cyclic graph by nature. Blind crawling of such a structure is a pointless waste of time and may never end. To achieve any efficiency, the crawler itself should be able to determine if the current crawling direction is perspective and lead to desired resources or not. Thus calculating weights of graph components (vertices and edges) is absolutely necessary to allow the automated crawling tool to navigate through the web.

This paper suggests various ways of calculating weights and proposes architecture of a web crawler designed for building image databases. Choosing the most appropriate search strategy seems to be the key point in building an efficient special purpose web crawler.

Keywords – web crawling; image databases; search engines; graph traversing (key words)

I. Introduction

The development in IT and modern computer systems in both hardware and software aspects allows them to be used in all areas where information can be given as a digital representation. They are also capable of storing and organizing huge amounts of different types of information which has led to the development of databases and information systems. In turn this has driven service improvement offered to customers using the information. Due to this development, these days databases are not only capable of dealing with written information but also images, audio and video formats. Naturally, the problem of combining, organizing and structuring all of this information emerges. The information needs to be useful for all users and the subjective factor when structuring and organizing all the data has to be avoided as much as possible. Thus, the aim is to automate the process of organizing and accessing the information. The solution to this problem with the current amount of information on the internet and the daily update of that information is the use of web crawlers or spiders for specific data type.

II. Theoretical Aspects

Web crawlers are applications or programs that use the graph structure of the Web space to crawl through it by page-by-page. This type of programs are also known as spiders, travelers, worms, robots and other similar concepts that have come from the essence of their features and functions. Web robots arise in order to organize and search for web pages content based on any of their representations stored in local databases of the web search engines [18]. Subsequently, these databases can be used for the needs of different applications.

If the web space was a static set of sites, it would significantly simplify the concepts and tasks of these searching robots, because once they have crawled through all the pages and created the database with the relevant information, it would not be necessary to crawl and search again. Web space, however, is too dynamic, which complicates the web robot's functions in order to maintain the integrity and up-to-date of the information in the relevant databases [1]. The result of the aim of this program to maintain the absolute integrity results in all areas is a very cumbersome application. To avoid this, web robots should be implemented to employ a certain selectivity with respect to the pages they crawl or the information they are looking for. These are the so-called heuristic robots that use thematic or focused crawl and search [2, 3, 4, 5]. Most often they are used as a combination of search engines for different types of information.

The main problem with the automated crawling of the Internet is the creation of an adequate strategy to ensure a high efficiency of crawling. The speed of new resources being published on the Internet is so quick that achieving high quality and quantifiable results for minimal time is critical to any spider.

III. Related Work

The issues that web crawlers generally face are the large volume and the rate of change of information in the web since every day there is a large number of pages added, edited or removed. An additional problem is the fact that current processing speeds and storage capacities have improved significantly more compared to network speed. To deal with the large volume of data the web crawler has to prioritize its tasks as it is unable to download all information within a given time. Hence, due to the high rate of change, the last pages to be downloaded might have been edited by the time the crawler has reached them [6, 7].

Matthew Gray, who studied physics in Massachusetts Institute of Technology (MIT) implemented ***World Wide Web Wanderer*** [8] in 1993. It was a standalone application written in Perl and the main purpose was to research the size of the World Wide Web for academic purposes and the Web was still relatively small. A single machine was used to develop and run ***Lycos Crawler*** [9]. Lycos used Perl's associative arrays to maintain the set of URLs to crawl. When the application went public on July 20, 1994 it had a catalog of 54,000 documents. It provided ranked relevance retrieval, prefix matching and word proximity bonuses. Lycos' capability of indexing tens of millions of pages was its main difference but unfortunately the design of this crawler remains undocumented. The first published Web crawler was *RBSE* [10] and it was based on two programs – spider and mite. The former was a program that creates and manipulates an Oracle database of the Web graph, traversing links having patterns of "*.html" or "http:*/ ". Mite, when given a URL as a command line argument, retrieves the document into a local file and prints any URLs found in the document on standard output. Spider spawns mite, reads the list of URLs and stores URL source–target pairs in the primary database table. ***WebCrawler*** [11] was the first comprehensive full-text search engine for the World-Wide Web and it is still working on the web. A full text index and a graph representation of the Web form the database of the WebCrawler, which is stored on disk. The database is updated as new documents are added and to protect it from system crashes these updates are made under the scope of transaction that are committed every few hundred documents. ***Google Crawler*** [12] was implemented in C++ and Python at Stanford. Only the early version of its architecture is described in some detail - the original crawler consisted of five functional modules running in different processes. Google's fast distributed crawling system could scale to hundreds of millions of Web pages. A single URL server serves lists of URLs to a number of crawlers (typically about 3). Roughly 300 connections are kept open at once by each crawler. This is necessary to retrieve Web pages at a fast enough pace. Using four crawlers the system could crawl over 100 Web pages per second at peak speeds. Written in Java, ***Ubicrawler*** [13] is a distributed crawler that has no central process. It is composed of several identical "agents" that ensure no page is crawled twice unless one of the agent crashes causing another agent to re-crawl the pages from the failed agent. The assignment function is calculated using consistent hashing of the host names. UbiCrawler offered platform independence, full distribution of every task, tolerance to failures and scalability as essential features..

IV. Working Algorithm and Architecture

The World Wide Web (www) is a cyclic oriented weightless graph by nature that could have finite but enormous number of vertices. Blind crawling of such structure is meaningless as it cannot finish in a reasonable amount of time. So crawling should be navigated somehow in order to reach high efficiency in retrieving useful resources only. To achieve that, evaluation criteria should be introduced that measure if and how perspective the selected crawling direction is. The overall numerical representation of these criteria could be directly used as a weight of vertices or edges. That converts www from weightless to a weighted graph. And this transformation is mandatory in order to allow the crawling algorithm to navigate itself, i.e. to decide by itself which vertex (web site) to continue with. Then any of the well-known heuristic algorithms for informed (guided) graph search could be used as a crawling algorithm. The most challenging task in web spider development is defining the way weights are calculated. Their accuracy and precision directly impact decision making and crawling efficiency.

There are three ways of calculation vertices weight is general:

- the spider calculates weights by itself based on some heuristics or evaluation criteria;
- the weight of a vertex is calculated as a sum of the weight of all edges pointing to it (similar to PageRank algorithm);
- combination from the previous two.

In the early years of www, search engines calculated the page ranks based on the second way - as a sum of weights of the vertices/edges pointing to them. At that time however search engines designers had not think of sophisticated ways of calculating weights of edges. So the weight of a vertex depended entirely on the number of edges pointing to it. In other words the position of the web site within the search results depended primary on the number of hyperlinks pointing to it. Having that in mind, many ambitious entrepreneurs created large networks of similar interconnected websites (for example online shops) that points to each other. Or SEO experts posted plenty of comments in discussion forums, while adding a hyperlink to their website at the end of each post. That tricks worked well for some time, moving highly interconnected websites higher in the search results.

However this way of calculating page ranks started to decline over time and gave place to some more adequate methods allowing the spider to calculate weights (page ranks) by itself based on proper criteria – not only the number of pages pointing to a website, but also the importance/significance of the referrer, its weight, number of visitors, rate of update and others. Furthermore in case of a domain-specific search the spider can also assess how much the crawling resource is related to the specified subject domain and if it satisfies other pre-defined requirements.

Every spider maintains its own data structure containing the URLs of the non-crawled vertices (or those already crawled but should be re-indexed). For simplicity, this structure will be called a list, but in practice it could be of a more complex type. The goal of the crawling strategy algorithm is to calculate the weight of all vertices in the most accurate way. The rest is just sorting the list of URLs by their importance (i.e. weight) in descending order. The weight of a newly discovered url (hyperlink) could be calculated as follows:

$$w(l_i) = h^* + w(p) + Sim(uSC, l_i) \qquad (1)$$

Where:

$w(l_i)$ – weight of the *i*-th hyperlink, found in currently-processed page *p*.

h^* – optional parameter. If present, could be 1 if the link belongs to the same web domain and 0 if the url points to an external domain. The presence of this parameter partly converts the best-first search strategy to breadth-first "best-second" search and allows crawling an entire domain before going to the next one.

$w(p)$ – weight of the page *p*. If the page is highly related to the user's search criteria then its hyperlinks should have higher weight.

$Sim(uSC, l_i)$ – semantic similarity between the user-defined search criteria (keywords used to focus the search) and the text describing the hyperlink l_i. Could be calculated by any existing similarity measure or method known from the Information Retrieval – *vector space model*, *latent semantic analysis* and others.

A text describing a hyperlink could be available on multiple places within or around the link itself. If such is present on more than one place, then just one of them is taken into account, determined in the following order (priority):

- The "title" attribute of the HTML <a> tag. If present, it is supposed to contain the most precise description of the page the link points to.
- The text immediately before or after the <a>…</a> tag. If there is no title attribute in the hyperlink's <a> tag then we can look the text around it. It usually states what the user could find in the target page.
- The text between opening <a> and closing </a>.

Focused crawling is achieved by applying user-defined search criteria (denoted as *uSC*), usually typed in the form of keywords. The similarity between them and the links description could be calculated by implementing some basic text analysis methods, for example the Vector Space Model. In this case *Sim*(*uSC*, l_i) could be calculated as the cosine similarity between the vectors of the query and the link's l_i description (2).

$$Sim(uSC, l_i) = \frac{\sum_{j=1}^{n} w_{qj} w_{ij}}{\sqrt{\sum_{j=1}^{n} (w_{qj})^2 \sum_{j=1}^{n} (w_{ij})^2}} \qquad (2)$$

Where:

w_{qj} – weight of the *j*-th term in the query (the j-th keyword of the user-defined search criteria).

w_{ij} – weight of the *j*-th term, describing the *i*-th discovered hyperlink. The *i*-th link is the one currently being processed.

n – number of terms within the entire document (hyperlink) collection built so far.

When implementing vector space model, IR experts (Grossman and Frieder [14]; and Salton and Buckley [15]) suggest to use lnc.ltc weighing scheme to calculate the weight of query and document terms. However using inverse document frequency (*idf*) is a bit tricky in crawling and could lead to potentially bad results. In general, *idf* is employed to lower the weight of a term if it appears in many documents. Its high frequency of appearance is assumed to mean it is low informative. And this is true in general, but it is not the case in focused crawling guided by user-defined keywords. Let's assume we are searching for airplanes and that is the keyword provided by the user. Then the frequent appearance of "airplane" within the descriptions of many hyperlinks does not mean it is insignificant keyword. It means there are many URLs that satisfies user's search criterion. On the other hand inverse document frequency reliably detects and excludes words which we can assume uninformative related to users' search - prepositions, conjunctions and etc. Therefore, if no *idf* is applied then a list of stop words [17] should be used to remove semantically insignificant words. Additionally words could be stemmed by using Porter's algorithm [16] before computing their weights.

If no *idf* is used, then the weight of terms of both the query and the hyperlinks could be calculated as their frequencies or their logarithmic scale down (3).

$$w_{ij}, w_{qj} = 1 + \log(tf_j) \qquad (3)$$

Where tf_j is the number of times the term *j* appears in the query or in the hyperlink description.

If crawler designer decide to use *idf*, then the term weighs of hyperlinks could be calculated by the traditional ltc scheme.

$$w_{ij} = \left(1 + \log\left(tf_{ij}\right)\right) * \log\frac{N}{n_t} \qquad (4)$$

Where *N* is the number of all discovered hyperlinks and n_t is the number of hyperlinks that contain the term *t* in their description.

It should be noted here that in context of crawling we do not have an entire document collection in advance, but it is constantly growing as crawling progresses. So *N*, n_t and the inverse document frequency of a term *t* are constantly changing as well.

The weight of a page w(*p*) could be determined as the similarity between the user-defined criteria (keywords) and the page content. So it could be calculated in the same way like the one between the keywords and the hyperlink descriptions. Additionally the place of appearance of a term inside the page content could also be taken into account when computing its weight. We can assume that words inside titles and headings (h1, h2 and h3 html tags) are actually more informative than the words within the body of the text. So they can get higher weight.

Figure 1 presents a generalized algorithm of a web crawler, used to automatically extract, process and store image URLs. It should be mentioned here that not all image urls are stored but just those of images that meet the pre-defined requirements for data format and size – JPEG or PNG images with aspect ratio > 9:21 and < 21:9, and width higher than 350 pixels. All the other images are considered to be advertising banners, menu buttons and etc.

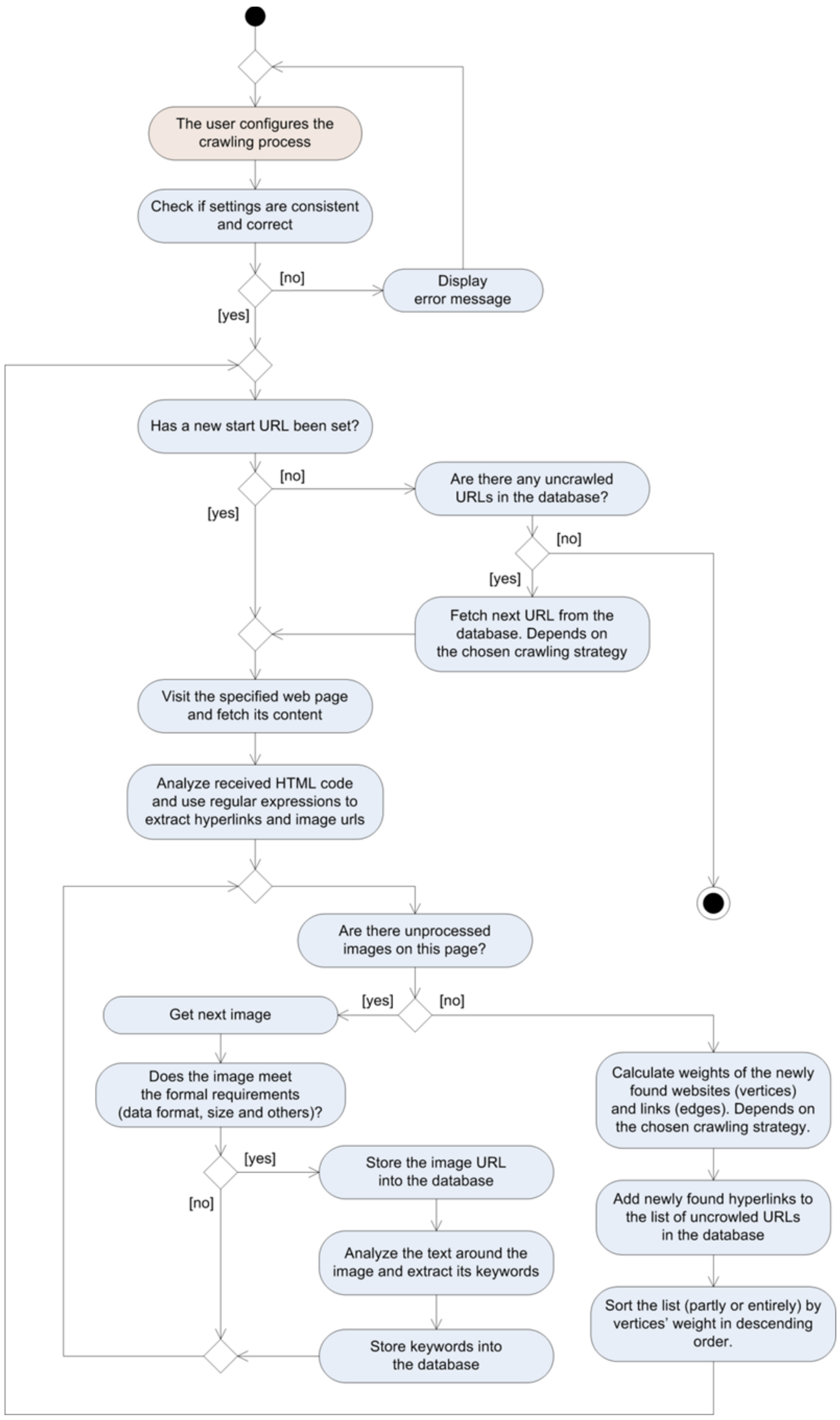
The user configures the crawling process
Check if settings are consistent and correct
[no]
Display error message
[yes]
Has a new start URL been set?
[no]
Are there any uncrawled URLs in the database?
[yes]
[no]
[yes]
Fetch next URL from the database. Depends on the chosen crawling strategy
Visit the specified web page and fetch its content
Analyze received HTML code and use regular expressions to extract hyperlinks and image urls
Are there unprocessed images on this page?
Get next image
[yes]
[no]
Does the image meet the formal requirements (data format, size and others)?
Calculate weights of the newly found websites (vertices) and links (edges). Depends on the chosen crawling strategy.
[yes]
Store the image URL into the database
[no]
Add newly found hyperlinks to the list of uncrowled URLs in the database
Analyze the text around the image and extract its keywords
Sort the list (partly or entirely) by vertices' weight in descending order.
Store keywords into the database

Figure 1. Generalized working algorithm of a web crawler

At the beginning the list of URLs is empty and the user should set up a new starting address. The crawler connects to it and fetches its content including its HTML code. All hyperlinks are located and extracted from the html code, and temporarily stored in the operating memory. The same is done for image URLs as well. Then the crawler gets each image by its URL and analyzes its size and format. If the image meets the requirements, its url is kept within the operating memory until the crawler retrieves its metadata. The latter is done as follows: The HTML code is removed from the page so that just the text left. Then the text is processed by one or more semantic analysis methods (such as latent semantic indexing, vector space analysis employing proper tf-idf schemes and etc.) in order to obtain its keywords. A word could be classified as a keyword based on its meaning, grammatical (lexical and syntax) significance, frequency of appearance in the text, user-specified importance and others. The extracted keywords are actually used as image metadata.

Once keywords are retrieved, they, along with the URLs of the images that meet the requirements, are stored into the crawler's database. Needless to say, keywords and their relevance to the chosen subject domain are also used in calculating the weight of the respective vertex (website). That is why weights are calculated right after the text analysis. Then extracted hyperlinks together with the calculated weights are added to the list of non-crawled vertices/URLs. Finally the list is sorted (partly or entirely) by vertex/website weight in descending order and the crawler continues with the first URL positioned at the beginning of the already sorted list. The first element on the list represents the most perspective website that should be crawled next. Although text analysis and keyword extraction from it are quite time–consuming operations, they are mandatory in order to perform an automatic image classification and categorization. Otherwise there will be complete chaos. The crawler may have downloaded a large amount of images but they are useless if there is no way to automatically determine their content and semantics.

Figure 2 presents an architecture of a web-based crawler used for retrieving images that meet certain requirements for format, size and semantics. As most of the web applications, the crawler implements a 3-tier architecture where user interface, business logic, and data are separated into three functionally-independent non-overlapping layers. That provides easier data replication and portability; higher reliability and flexibility; and allows fast and easy personalization or complete change of the user interface.

The crawler's core functionality is implemented by the following modules:

- ***Web Crawler and Content Analysis*** – responsible for accessing pages on the Internet, fetching and analyzing their content.
- ***Crawling Strategy Planner*** – calculates weights of the graph vertices and edges based on the data provided by the content analysis and image analysis modules.
- ***Image Analysis and Processing*** – checks whether the newly discovered images meet the formal requirements and if so stores them into the local database. Optionally this module can extract and index additional image features such as color content and its spatial distribution.
- ***Crawling Statistics*** – maintains and provides real-time statistics about the crawling process and its progress.
- ***Settings*** – Allows administrative users to manage the entire crawling process in details.

Once the user sets up the crawling and the indexing processes, and provides a starting URL, the spider starts working. It visits every web page, analyzes its HTML code and extracts hyperlinks, image URLs and keywords from it. Image URLs are passed to the "Image Analysis and Processing" module that checks their format and size, and decides if they meet the requirements and should be stored into the local database. Similarly, hyperlinks and keywords are sent to the "Crawling Strategy Planner" that calculates the weights of the current and the newly discovered vertices (web pages) of the graph. Then again the Crawling Strategy Planner adds the newly extracted URLs to the list of unvisited ones and sorts the list by vertices weight in descending order. Sorting could be done partially or entirely based on user settings and the chosen crawling strategy. After processing the current vertex (web page), the "Web Crawler and Content Analysis" module deletes it from the list of non-crawled URLs and continues with the next one. As the list is sorted by weight in descending order the next vertex represents the page evaluated as most perspective by the "Crawling Strategy Planner". All these activities are cyclically repeated until the user stops the crawler or the list of unvisited URLs gets empty.

V. CONLUSION

Millions of images are uploaded to the Internet every day by people all over the world. To be useful however these images should be easily discoverable and accessible, i.e. they need to be organized and indexed in a way that will allow fast subsequent search among them.

The World Wide Web represents an enormous directed cyclic weightless graph that features an extremely high rate of change. Blind (brute force) crawling of such structure is literally impossible and it cannot finish in a reasonable amount of time. Thus searching should be navigated by proper heuristics in order to reach efficiency in retrieving useful resources. To achieve that the graph should be converted from weightless to weighted one by introducing vertices weights. We propose that in focus crawling, the weight of a vertex (newly discovered hyperlink) should depend on: the semantic similarity between its description and the user-defined search criteria (keywords); and the importance of its referrer. Additionally the importance of the referrer can also include its number of visitors and rate of change.

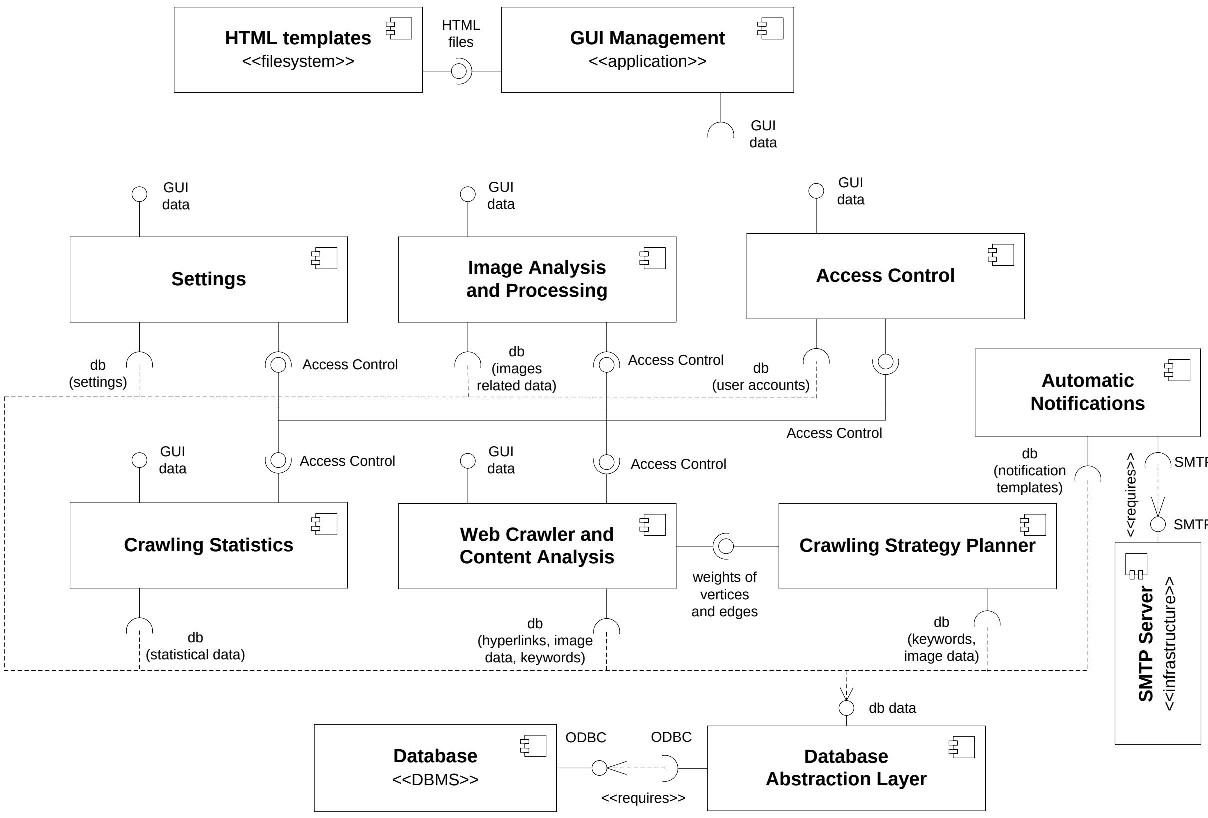

Figure 2. Architecture of a web-based web crawler

We have implemented a web crawler that we use to build and maintain our own image database (imdb). Its primary objective is to provide us with a large enough dataset for testing various content-based image retrieval (CBIR) techniques, methods and algorithms. However our crawler implements a breadth-first search strategy that is an exhausted search crawling rather than focused one. We will soon modify it according to our suggestions presented in this paper. Then we will perform a comprehensive experimental evaluation to determine the most suitable term-weighting schemes. Results will be presented in another paper that will be entirely devoted to experimental evaluation of the proposed focused web crawler.

REFERENCES

[1] A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the Web. ACM Transactions on Internet Technology, 1(1), 2001.

[2] S. Chakrabarti. Mining the Web. Morgan Kaufmann, 2003.

[3] J. Cho, H. Garcia-Molina, and L. Page. Eficient crawling through URL ordering. Computer Networks, 30(1-7):161-172, 1998.

[4] S. Chakrabarti, M. van den Berg, B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. Computer Networks, 31(11-16):1623-1640, 1999.

[5] A.K. McCallum, K. Nigam, J. Rennie, K. Seymore. Automating the construction of internet portals with machine learning. Information Retrieval, 3(2):127-163, 2000.

[6] Olston and M. Najork , “Web Crawling”, Foundations and Trends in Information Retrieval, vol. 4, No. 3 ,pp. 175–246, 2010

[7] Marinov, M., T. Pavlov, Rule-Based Decision Support Tools for Injection Moulding, International Journal of Knowledge-Based and Intelligent Engineering Systems, IOS Press, Vol. 19, No. 2, pp. 97-107 (2015).

[8] M. Gray, Internet Growth and Statistics: Credits and Background, available at: http://www.mit.edu/~mkgray/net/printable/

[9] M. Mauldin, Lycos: Design Choices in an Internet Search Service, IEEE Expert, vol. 12, pp. 8-11, 1997.

[10] D. Eichmann, The RBSE spider: balancing effective search against web load. In Proceedings of the first World Wide Web Conference, Geneva, Switzerland, May 1994.

[11] B. Pinkerton, Finding what people want: Experiences with the WebCrawler. In Proceedings of the first World Wide Web Conference, Geneva, Switzerland, May 1994.

[12] S. Brin, L. Page, The anatomy of a large-scale hyper textual Web search engine. Computer Networks and ISDN Systems, 30(1–7):107–117, April 1998.

[13] P. Boldi, B. Codenotti, M. Santini, S. Vigna, UbiCrawler: a scalable fully distributed Web crawler. Software, Practice and Experience, 34(8):711–726, 2004.

[14] Grossman, D., Frieder, O. Information Retrieval: Algorithms and Heuristics 2nd Ed. Springer, The Netherlands, 2004, ISBN: 1-4020-3004-5.

[15] Salton, G., Buckley, C. Term-weighting approaches in automatic text retrieval. Information Processing and Management, 24(5):513-523, 1988

[16] Porter, Matrin F. Analgorithm for suffix stripping. In J. S. Karen and P. Willet, editors, Readings in information retrieval, pages 313-316. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997

[17] List of stop words, http://ir.dcs.gla.ac.uk/resources/linguistic_utils/stop_words

[18] C. Saini, V. Arora, "Information retrieval in web crawling: A survey", 2016 International conference on Advances in Computing Communications and Informatics (ICACCI), pp. 2635-2643, Sept 2016.