IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 19, Issue 2, Ver. III (Mar.-Apr. 2017), PP 45-55
www.iosrjournals.org

Scalable Framework for Locating Deep Web Entry Points

John Onihunwa[1], Olufade Onifade[2], IsaacAriyo[1], Stephen Omotugba[1], Deji Joshua[1]

[1](Computer Science Department, Federal College of Wildlife Management, New Bussa, Niger state, Nigeria)[2](Computer Science Department, University of Ibadan, Ibadan, Oyo state, Nigeria)

Abstract:*In this information age, searching for information from the internet becomes wide spread, but some of the valuable information are hidden behind dynamically generated page which the search engines never crawled. It becomes imperative to find a way of making them available. This research work provides a scalable framework for locating forms that serves as the entry point for information in web databases. The methodology focuses on the link classifier by using Path-Ascending algorithm which has the advantage of searching as many resources as possible from a particular site, since forms are sparsely distributed, After implementing this framework, it was observed that the framework searches more than 90% of the links in each site tested which is comparatively higher than using search algorithm in the link classifier. This framework suggest using path ascending algorithm in the link classifier which make the crawler to search almost all the links on each page of a site providing a better opportunity to locate a searchable form.*

Crawler: *a program that systematically browse the world wide web in order to create an index of data, deep web: information buried far down on dynamically generated sites, and standard search engines never find it,* ***Frontier:****it consist of URL and Crawling modules used to crawl the web pages to extract the meta data,surface web: information that common search engines like Google, Yahoo, AltaVista, Ask, and Bing can crawl.*

I. Introduction

Searching for information on the Internet today can be compared to dragging a net across the surface of the ocean. While a great deal may be caught in the net, there is still a wealth of information that is deep, and therefore, missed. The reason is simple: Most of the Web's information is buried far down on dynamically generated sites, and standard search engines never find it (Bergman, 2001). The common search engines like Google, Yahoo, AltaVista, Ask, and Bing can only crawl static and linked pages. Search engines cannot "see" or retrieve content in the deep Web, those pages do not exist until they are created dynamically as the result of a specific search. Conventional crawlers cannot search below the surface; the deep Web has been hidden for quiet sometimes [1] (Bergman, 2001). Figure 1 shows how search engines harvest information on the net.

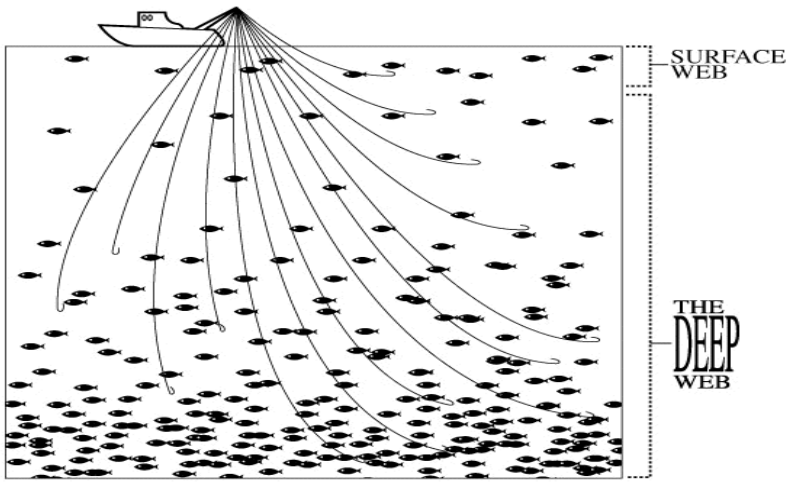

Figure 1: Harvesting the Deep and Surface Web with a Directed Query Engine
(Source: http://dx.doi.org/10.3998/3336451.0007.104)

The roles of search engines have become complex with millions of websites being added to the web each day. In the earliest days of the Web, there were relatively few online documents and websites. It was manageable to create these online documents as static pages. Because all results were persistent and constantly available, they could easily be crawled by conventional search engines. However, with commercialization of web and database technology being introduced to the internet, more and more websites minimized the use of

static pages and became database driven **(**Bergman, 2001). Major search engines have done great work in crawling the large collections of static web pages; the valuable data hidden behind the search forms in the databases still possesses challenges even though these pages are not private or restricted pages. This hidden data consists of valuable information to the academicians, librarian, scholars, and businessmen. The entry to deep web requires access to databases. These databases are generally accessed through searchable forms. When a user fills out these forms, a dynamic page is generated showing results from the database. There are also some information hidden within File Transfer Protocol (FTP) that are not accessible to search engines (Onifade& Fagbenro, 2011). (Chen- Chuan et al., 2000) size only 60 popular deep web sites together was 40 times larger than that of the known surface web not just in size, but also in quality. According to his study in 2007, Deep web is of large scale of 307,000 sites, 450,000 databases, and 1,258,000 interfaces. Since there are numerous databases on the web, a user finds it difficult to search for useful databases. Deep web coverage is broad and relevant, this explain why we cannot ignore or forget about these wealth of information.

However not all deep web content is hidden. Any deep web content whose URL address is listed on a static Web page is discoverable by crawlers and is indexable by the search engines. This can occur when a Web page author discovers some useful Deep Web content and posts its dynamic URL address on a static Web page. The major challenges for any search engine to crawl deep web are: a) How to efficiently locate the searchable forms that serve as the entry points for the hidden Web? i.e. database driven websites. b) How to monitor new pages that are added and old sources that are removed or modified? **c)** How to locate forms that are very sparsely distributed? Scattered and disconnected web pages are the major problem of search engines, if these pages could be discovered there would be no problem indexing them.

1.2 Significance of the Study

Even within the strict context of the Web, most users are aware only of the content presented to them via search engines such as Google, AltaVista, Yahoo, and Bing. Eighty-five percent of Web users use search engines to find needed information, but nearly as high a percentage cite the inability to find desired information as one of their biggest frustrations. This is because the conventional search engines crawler cannot crawl the content of the deep web, these quality and relevant information are available in the deep web but are hidden behind search forms, and a user finds it difficult to search for useful databases. Our framework is to search for resources that are sparsely distributed and provide access or entry point to these vital information that are hidden behind web databases.

II. Literature review

Central to deep web access is to provide an entry point to these information that are hidden in forms behind web databases and FTP protocol. There are different types of search engines methodologies available for web crawlers, such as Page Ranking Algorithm, Focused Crawling Algorithm, Path-ascending crawling and Breadth First Search Algorithm. Google, Yahoo and Bing are the most widely used search engines, but their crawling algorithm is a highly confidential business secret. Few standard crawling algorithms are discussed below.(Soumen et al., 1999) proposed a best-first focused crawler called baseline which uses a page classifier to guide the search. The classifier learns to classify pages as belonging to topics in taxonomy. Unlike the breadth first focus crawler that follows each link in a page, it gives priority to link that the classifier considers being relevant. The main disadvantage best-first strategy is that its lead to suboptimal harvest rates, since even in domains that are not very narrow, the number of links that are irrelevant to the topic can be very high.

(Sriram & Hector, 2001) Built a prototype hidden Web crawler called HiWE, whose basic idea is to extract some descriptive information for each element of a form and a task-specific database that is organized in terms of a finite number of concepts, each of which is also associated with labels. With a matching algorithms that attempts to match form labels with database labels to compute a set of candidate value assignments. They also use LITE (Layout-based Information Extraction), which uses the physical layout of a page in addition to the text to aid in extraction. HiWE's main challenge is how to accurately analysis form extraction of the labels and domains of form elements. In the same vein (Raghavan et al., 2001) Proposed a basic crawler data structure is the URL List, that contains all the URLs that the crawler has discovered and a crawl manager (CM) controls the entire crawling process. Figure 2 shows the architecture of HiWE. The Parser extracts hypertext links from the crawled pages and adds them to the URL List structure. Pages that do not contain forms are handled solely by the Parser and CM modules. The Form Analyzer, Form Processor, and Response Analyzer modules, together implement the form processing and submission operations of the crawler. The LVS table is HiWE's implementation of the task-specific database and it manages additions and accesses to the LVS table.

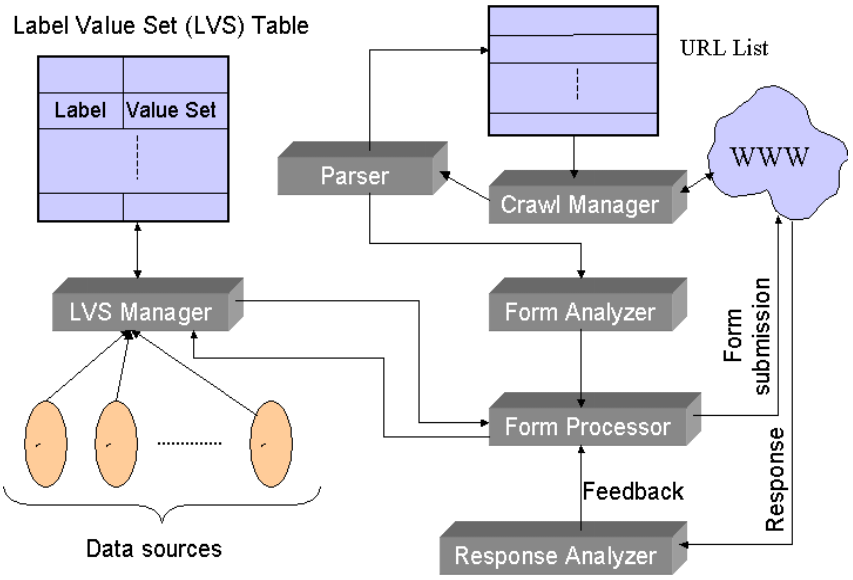

Figure 2: HiWE Architecture
(Raghavan et al., 2001) Crawling the hidden web.

(Barbosa & Freire 2005) Combined a focused crawler that takes the page contents into account, with a mechanism that allows the crawler to select links that have delayed benefit using best-first algorithm. This limit the search to a particular topic by learning the features of links and paths that leads to pages that contain searchable forms and employing appropriate stopping criteria. The crawler uses the page and the link classifiers to guide its search, a form classifier to filter out non relevant forms. Due to the sparseness of searchable forms, form Crawler stopping criteria might not be able to search out all available forms by learning to identify promising links only. It performance depends mostly on the richness of links in the specific topic being searched, and usually relies on a general Web search engine for providing starting points.

(Jayant et al., 2008) Based their work on Iterative probing as a means to retrieving documents from a text database. These approach the goal of achieving maximum coverage of specific sites. In their study, they only consider one text box per form and based their work on a manual analysis, they believe that in a large fraction of forms, the most likely generic text box is the first text box. Hence, they apply the iterative probing approach to the first text box. This assumption might not be true in all cases, it is possible to have multiple text boxes in a HTML forms. Their works have the goal of achieving coverage on specifically chosen sites and hence employ site-specific techniques to varying degrees without consider the efficiency of deep Web crawling. (Tian, 2010) Proposed a new type of algorithm of page ranking that combines classified tree with static algorithm of Page Rank, which enables the classified tree to be constructed according to a large number of users' similar searching results, and can obviously reduce the problem of Theme-Drift, caused by using PageRank only, and problem of out dated web pages and increase the efficiency and effectiveness of search (Shaojie, 2010). Proposed a new page rank algorithm based on similarity measure from the vector space model, called SimRank, to score web pages. They proposed a new similarity measure to compute the similarity of pages and apply it to partition a web database into several web social networks. However, when a web page receives links from an important page then certainly it should have a high rank. Therefore, Page Rank of a web page corresponds to the weighted sum of input links. (Anish et al., 2013) Studied the various issues important for designing high performance crawling system. The crawler visits the URLs; it identifies the entire hyperlink and adds them to the list of the URLs to visit, called the crawl frontier. These URLs are visited recursively from the frontier according to the policies. The policies are: selection policy, re-visit policy, politeness policy, and parallelization policy. They described the various policies their advantages and disadvantages and they concluded that all the policies are to be implemented in parallel or collectively in order to design high performance system. (Priya et al., 2014) Concentrate on focus ontology which search for the relevant web pages based on the keyword we give to form a hierarchy of links. It searches for link on that seed URL and after that switch to that link and find another link on that web page but it should match with the keyword, it will do that until it reaches the set limit. While fetching the links the user profiles ensure that it does not revisit the same link twice and at the end a txt file is generated on which they run their three pattern matching algorithm. The major concerns is, it manually selected its indexed pages for testing and the computed results are at webpage level not at individual URL level.

For this work there are few essential features of Web pages that should be noted. First, the components that occur and are tagged within an HTML page may be URLs of other pages. Since a given page may contain many URLs of other pages, and since many pages may contain the URL of a popular page, the pages comprising the Web are linked together in an elaborate structure of vagarious complexity.

Second, the presence in a given page, P1, of a URL pointing to a second page, P2, implies some association between the two pages. But there are no general uniform rules, let alone enforcement mechanisms, for ensuring that there is some reasonable connection, e.g., by author or topic, between any two pages linked by URL. Any individual or organization can create a "home page," a page expressing the interests or concerns of the given individual or organization. Any author of a page can link her page to any other, e.g., an individual may choose to link her page to other pages on diverse topics that she/he personally "likes." P2 may have been created by the author of P1, or by another individual working for the same organization, and may be on the same computer as P1 in which case they are said to be part of the same web site. But P2 may just as easily be a page created by a completely different author, and be located on a computer located in a different part of the world. A sentence in P1 containing a tagged URL to P2 may, but need not, explain the reason for the reference. Even if such an explanation is present, it will probably be in natural language, not readily interpretable by any search engine. The URL itself is a string of text, consisting of a number of standardized components in a standardized order. The most common URL format specifies the official registered name popularly called host name or server name or domain name of the computer that provides access to the page being addressed, and the path name within the filestructure of the given host of the file where the page actually resides. Similarly, the file path name is a series of directory and sub-directory names, ideally chosen to give some clues to the subject or author or purpose of the given page, e.g., "ui.postgraduateschool/sciences/ computer-science/john.html" specifies a directory named "ui.postgraduateschool," a sub-directory within "ui.postgraduateschool" named "sciences," a sub-directory within "sciences" named "computer-science," and a file within the "computer-science" directory named "john.html" containing URL's to numerous john resources

Third, a URL may point to a file that is not a Web page. This may mean that a Web server is available on the host, but the page is not in HTML format, and hence does not contain as much metadata as a document that has been properly "marked up." In that case, the page can be retrieved by a client that supports HTTP (a Web "browser"), but it may require some additional program to display the file, e.g., an ordinary text editor for an ASCII text file, an image viewer fora file in some standard image format, etc. Or, it may mean that the file cannot be retrieved via the HTTP protocol because its host does not contain a Web (HTTP) server. In that case, some other kind of server must be used to retrieve the given file. The two most common non-Web servers are FTP and Gopher. An immense number of files, both textual and binary files, are available for transfer free of charge to the local host of any Internet user via an FTP server. Textual files may contain any conceivable kind of textual or numeric data, including documents on any conceivable subject and program source code for distribution. Binary files may contain images, video, audio, executable programs, etc. The files at an FTP site may be arranged in a structure, typically hierarchical, that can be browsed by the Internet user. Any file that the user encounters in her browsing that "looks" interesting (perhaps on the basis of its path name), can be retrieved using the FTP server and really looked at. However, these FTP files are not Web pages, and hence will not contain URL links to files (whether Web pages or FTP files) at other sites. The only browsing that can be done at an FTP site is up or down the local hierarchy of FTP files.

Gopher servers provide a hierarchy of menus. When the user selects an item from a given menu, e.g., with a mouse click, she gets either another menu or a data file. The data file may be any of the types that can be accessed via FTP. However, the file selected by a menu selection may be on a different host than the one where the menu was located. Hence, Gopher supports a form of "hyperlink" equivalent to a URL link between Web pages. In fact, the Gopher servers, menus, files, and links to files at other Gopher servers formed (and still form) an early form of hyperspace, called gopherspace, still quite useful and widespread, although it has been superseded as the hyperspace by the Web.

Finally, a Web page may be a gateway to a set of structured databases, textual databases, or other services outside of the Web itself. In other words, once such a page is reached, further access to a database or service may involve servers other than web servers, and local links other than URLs. Moreover, further access may depend upon payment of a fee, e.g., as in the case of the commercial databases described above. For example, you can reach the DIALOG home page free of charge. From there, you can jump freely to other pages that advertise the products and services that the DIALOG company provides commercially. The user can log on to these services from a DIALOG Web page. However, logging on requires a password and hence an account with DIALOG; this, in turn, requires payment of a fee. Hence, DIALOG databases and services can be accessed from the Web, but it is a commercial service and its databases are not a part of the Web.

III. Methodology

Accessing deep web is a major focus when searching for information from the internet, because of volume and quality of information hidden behind database forms. We drill down into various aspects of our Generic Model for Locating Deep Web Entry Points, how the nodes match an incoming document against its filter set, how documents and filters are partitioned across nodes, and how clients are notified about searchable forms. To deal with the sparse distribution of forms on the Web, our framework avoids crawling through unproductive paths by: limiting the search to a particular topic; learning features of links and paths that lead to pages that contain searchable forms; this is represented in Figure 3.

3.1 Frontier

This consist of URL and Crawling modules used to crawl the web pages to extract the meta data; User enters the URL which has to be indexed, then the meta data are stripped from web page which consists of keywords, description, the various links, and other relevant information like existence of form or forms on the page using meta data extractor process. The crawler would be used to keep information of the pages which is already visited, the pages that have not been visited yet, and it usually relies on a traditional Web search engine for providing starting points. We have to start from any good URL called "Seed".

This framework uses three classifiers to guide its search: the page classifier, the link classifiers, and the form classifier. The page classifier is trained to classify pages as belonging to topics using a search algorithm to classify a page as being on-topic or off-topic, forms and links are extracted from it. A form is added to the Searchable Form Database if the form classifier decides it is a searchable form, and if it is not already present in the Searchable Form database. The link classifier is trained to identify links that are likely to lead to pages that contain searchable form interfaces.

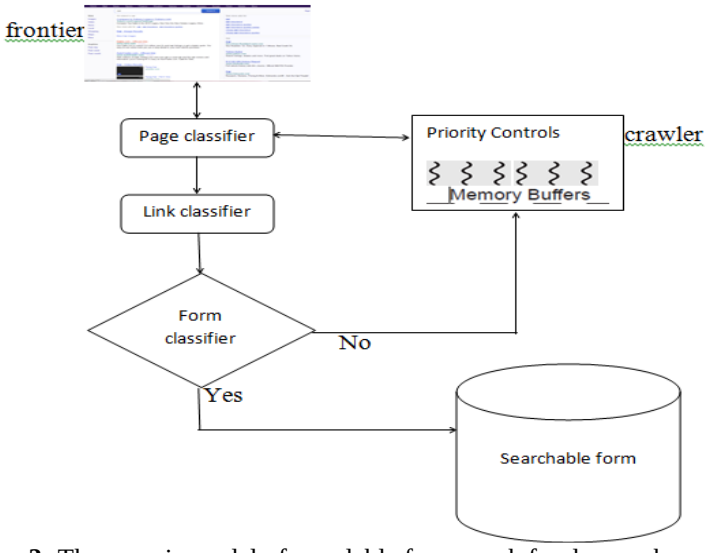

Figure 3: The generic model of a scalable framework for deep web access.

3.2 Page Classifier

The page classifier takes the seed as input and runs a topic distillation algorithm to identify pages containing large numbers of relevant resource links, the (re)visit priorities of these pages and immediate neighbours are raised. Similar to several works on page classifier (Barbosa & Freire, 2005).

3.3 Link Classifier

We use Path Ascending Search Algorithm to search through the output of page classifier since forms are sparsely distributed, by selecting only links that directly point to pages containing searchable forms, the crawler may miss good target pages. This algorithm has the advantage of searching as many resources as possible from a particular Web site; it ascends to every path in each URL that it intends to crawl. Figure 4 depicted how the algorithm works, any URL received from the page classifier is divided into paths and each path is sequentially searched to locate form or forms. For example, when given a seed URL of http://ui.postgraduateschool/sciences/computerscience/john.html, it will attempt to crawl /ui.postgraduateschool/, /sciences/, /computer-science/ and /john.html. This makes it very effective in finding isolated resources, or resources for which no inbound link would have been found for it in regular crawling. The link classifier stops crawling a given site if it retrieves a pre-defined number of distinct forms or if it visits the maximum number of pages on that site. This is depicted in fig 5.

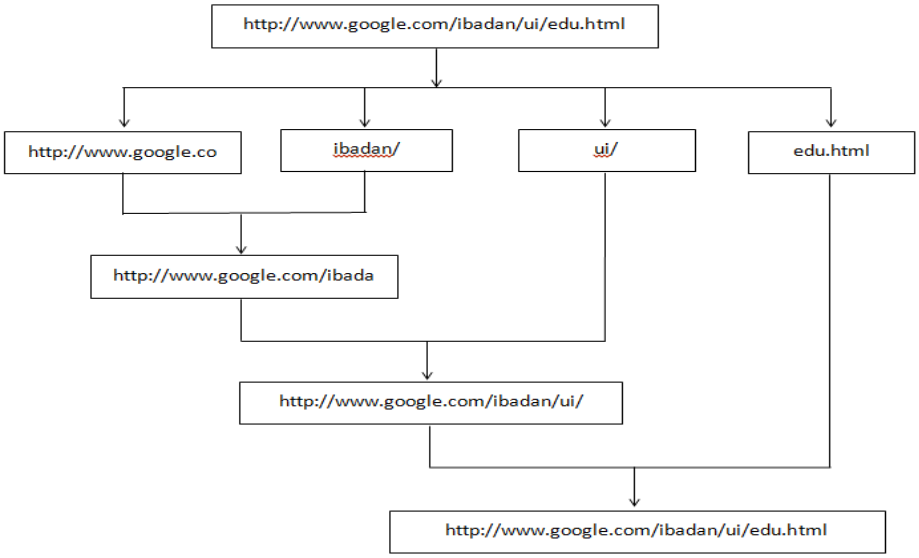

Figure 5: A Link Classifier Using Path Ascending

3.4 Form Classifier

The form classifier is a general domain-independent classifier that uses a decision tree to determine whether a form is searchable or non-searchable. . Forms can have different types of input control, the most popular ones are text boxes, select menus always defined in a separate select tag, check boxes, radio buttons, and submit buttons. . This research work focuses on select menus and text boxes in a form. Check boxes and radio buttons can be treated in the same way as select menus. When a form is submitted, the web browser sends an HTTP request with the inputs and their values to the server using one of two methods: get or post. Using get, the parameters are appended to the action and included as part of the URL in the HTTP request. With post, the parameters are sent in the body of the HTTP request and the URL is simply the action. The URLs obtained from forms that use **get** are unique and distinguishable, while the result pages from a **post** are indistinguishable and hence not directly indexable. For this reason we restrict our attention to **get** forms by ignoring forms that require any kind of personal information.

3.5 Crawler

The crawler has a watchdog thread and many worker threads. The watchdog is in charge of checking out new work from the frontier, which is stored on disk. Workers save details of newly explored pages in private per-worker disk structures and the classifier is invoked by each thread as it encounters a new page and sometimes the crawler is also stopped to execute the link classifier.

3.6 Searchable Form

These are forms from the form classifier that are accessible and can also serve as the entry point to deep web database. The path ascending algorithm used by the link classifier would enable this framework to search for resources that are sparsely distributed.

IV. System Implementation

The practical part of this work, a general crawler was designed, which incorporates basic methods from these types, but its main aspect is the legerity it offers for the developers. This data extraction tool can be used on most of modern hosting platforms. For source storage - a single MySQL database is used and entire crawler application source is written in PHP server-side language which is run in CLI mode on Apache server. Although market offers many services that index, order and provide data, we found few solutions that were available with open source and under Linux, Apache, MySQL, and PHP(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, analysing an existing web-site. That is why this research work has decided to make own program which would be available for changes. Examining the different algorithms clear out its purposes and why it is most suitable to use path-ascending algorithm for this work.

4.1 Program Implementation

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

- Legerity. 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, which focus on the same topic.

A. 4.2 Code Architecture

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

- Bootstrap 3.0
- Macromedia Dreamweaver
- Silk icons
- Mysql Query browser

System is based on several classes and objects (figure 4.3):

1. Database class implements access to MySQL database and acts as a 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.

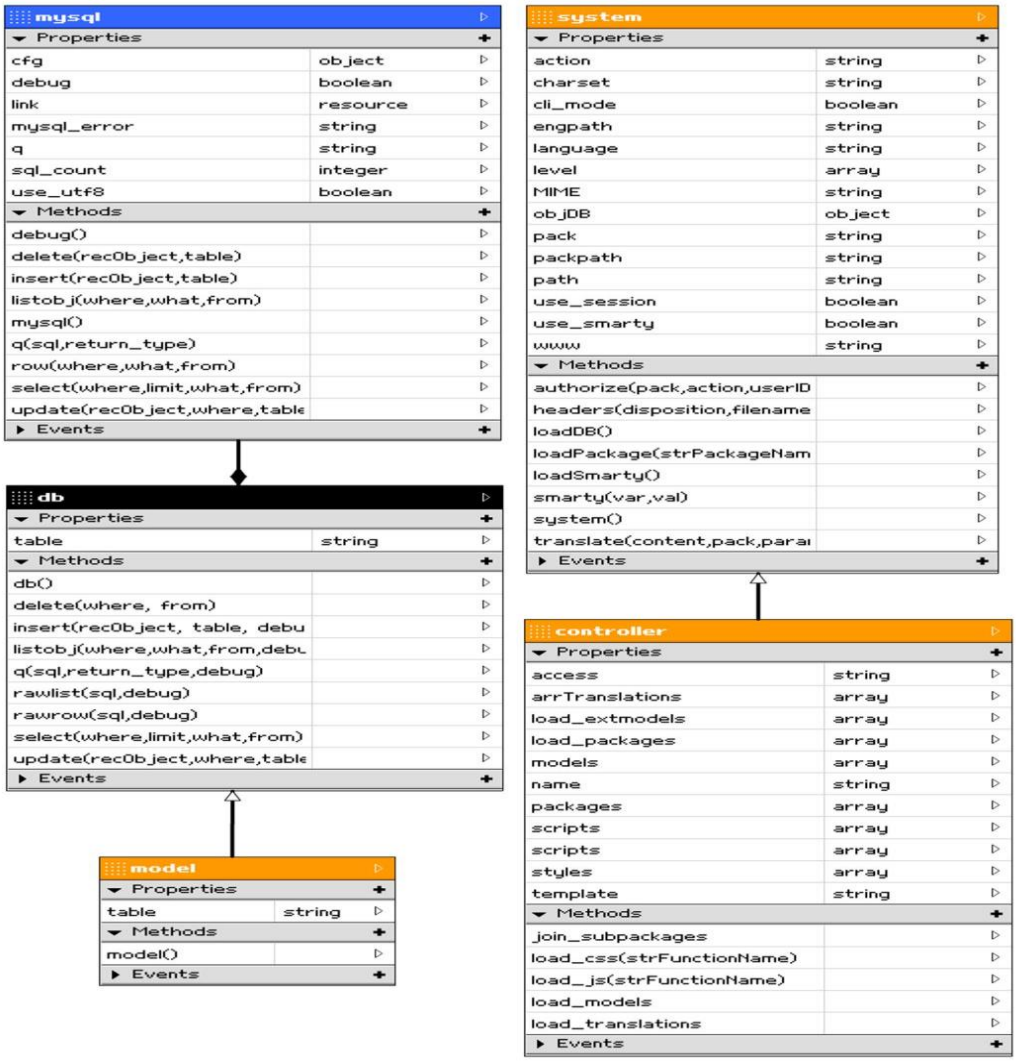

Figure 6: UML class diagram of the platform core

4.3 Path-Ascending Crawling

Path-ascending crawling intends the crawler to download as many resources as possible from a particular Web site. That way a crawler would ascend to every path in each URL that it intends to crawl. For example, when given a seed URL of http://www.google.com/ui/edu.html, it will attempt to crawl /ui/edu/, /ui/, and /page.html. The advantage with Path-ascending crawler is that they are very effective in finding isolated resources, or resources for which no inbound link would have been found in regular crawling. The importance of a page for a crawler can also be expressed as a function of the similarity of a page to a given query.

Pseudo code

$U = \{u_1, u_2, \text{--------}u_n\}$

$E = \{e_1, e_2, \text{---------}e_n\}$

Where

U – list of URL

E – list of URL to be search

E_f - explicit file requested

DB_L- database list

S - Site

1: Start ()

2: Specify starting U

3: Add u_1 to E

4: While $E \neq \emptyset$

5: If $u_1 \neq$ "HTTP" then

6: GOTO 3

7: $\exists$ robots.txt $\in$ S, then

8: If robots.txt ∈ S ⟶ "Disallow"
9: then GOTO 3, else
10: Open
11: If $u_1 \neq$ ("HTML" ∧ E_f)
12: GOTO 3
13: Set iteration to equate Next HTML file
14: If URL in HTML file GOTO 3
15: Else If URL previously searched = True
16: Add URL to DB_L
17: UpdateDB_L
18: }

4.4 User Interfaces

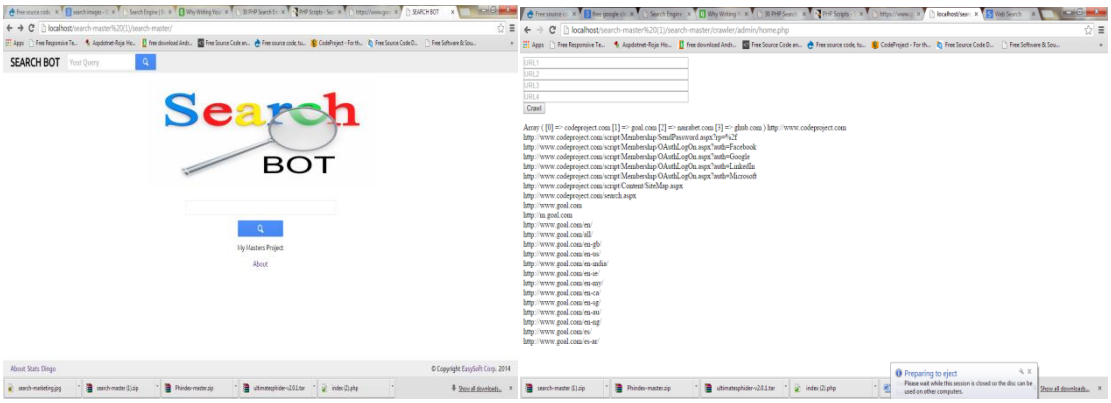

Figure 7: Main page user interface**Figure 8:** Crawling and Indexing page user interface

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 to XML
- Filter listing / filter management and testing

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.

4.5 Results

First crawling cycle used php 5, mysql 5, and lasted 20min on Intel Core 2 duo with 2 mbit channel and had the following results:

- 223 domains found
- 1521 pages found (assuming 8 pages per domain)
- 6944 links (assuming 6 links per page)

Additional interesting results were found:

- 16 KB per page
- 3-4 outgoing links per page
- 90% Pages found

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

- www.sourceforge.net (132 unique domains linked to this domain)
- www.flashscore.com (59)
- www.livescore.com (48)

Table 1: Unique domains linked to this domain

	Previous work	Path-Ascending	Available link
www.sourceforge.net	120	139	143
www.flashscore.com	55	59	66
www.livescore.com	45	48	53

Using HiddenWeb Exposer HiWE algorithm (Sriram & Hector, 2001) discover that their crawler finds it difficult to accurately analysis form extraction of the labels and domains of form elements. The Path-Ascending performs better in this regard.
The pictorial representation of table 1 is depicted below

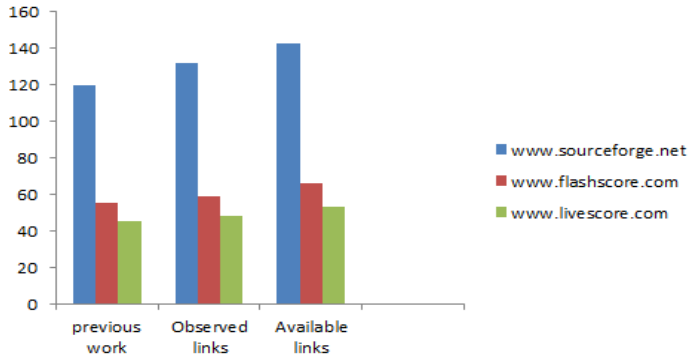

Figure 4.10: Unique domains linked to this domain

A script is written to search out links which possibly contain forms which serve as the entry point to hidden information behind web databases.

V. Conclusion

Developing a web-crawler is not innovative nor is it important in establishing healthy business structure – it is a tool that can be written, bought or used from open sources. We developed a 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 come under the influence of single selection policy, which means it is still abstract enough to be modified. On the other hand it lacks specialization, which would give speed and data processing surplus.

5.1 Recommendation

This framework address the issue of locating sparsely distributed information resource. The software is able to crawl a domain or group of domains, create its map, index and make full-text search. As a means 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, we learnt new features of MySQL usage, as well as php 5 when switching was done from php 4. We also discovered good programming ethics that can make programming fast and error free. It is recommended for educational and research purposes as it possesses the necessary information and steps needed to develop a functional database and algorithm based system.
In the future, the framework can be extended to search for Files Transfer Protocol files and search for forms saved using "Post" forms which cannot be crawl be the search engine.

References

[1] Bergman, M (2001). The deep web: Surfacing Hidden Value. Retrieved September 18, 2015 from http://www.brightplanet.com/technology/deepweb.asp
[2] Onifade, O.F.W& Fagbenro, M.S (2011). Extending the functionality of web crawler to include File Transfer Protocol, Africa Journal of Computing & ICT, Vol. 9, No.2, pp. 17 – 24.
[3] Khandelwal, B, Abbas, S. Q. (2014). Confluent Analysis of Challenges and Scalability of Efforts for Deep Web Data Retrieval, International Journal of Emerging Trends & Technology in Computer Science, Vol. 3, Issue 1,pp.223-227.
[4] Soumen C, Van den Berg M, Byron D (1999). Focused crawling: a new approach to topic-specific Web resource discovery, Computer Science and Engineering, Indian Institute of Technology, Bombay, pp. 545 – 562.
[5] Raghavan S and Garcia-Molina H (2001). Crawling the Hidden Web, In 27th International Conference on Very Large Data Bases (VLDB 2001)
[6] Barbosa L, Freire J (2005).Searching for Hidden-Web Databases, Proceedings of the Eighth International Workshop on the Web and Databases -WebDB 2005, pp. 1 - 6
[7] Jayant M, David K, Lucja K (2005). Google's Deep Web Crawl, The Journal of Electronic Publishing VLDB, Vol. 7, No. 1, pp. 1241- 1252.

[8] Shaojie, Q, Tianrui, L, Hong, L, Yan, Z, Jing, P, Jiangtao, Q (2010)SimRank: A Page Rank approach based on similarity measure Shaojie, 2010
[9] Anish G, Singh, K. B, Singh, R. K (2013). Study of Webcrawling Polices, International Journal of Innovative Technology and Exploring Engineering (IJITEE) Vol. 2, No. 6, pp. 65 – 67.
[10] Priya, R, Dhanalakshmi, S, Priyadharshini, S (2014). Web Crawling Forum, International Journal of Scientific and Research Publications, Volume 4, Issue 3.