LOKMAN: A MEDICAL ONTOLOGY BASED TOPICAL WEB CRAWLER

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ALTUĞ KAYIŞOĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

SEPTEMBER 2005

Approval of the Graduate School of Informatics

_____

Assoc. Prof. Nazife Baykal

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Onur Demirörs

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____

Assoc. Prof. Nazife Baykal

Supervisor

Examining Committee Members

Prof. Dr. Semih Bilgen	(METU)	_____
Assoc. Prof. Nazife Baykal	(METU)	_____
Dr. Ali Arifoğlu	(METU)	_____
Prof. Dr. Feza Korkusuz	(METU)	_____
Assist. Prof. Dr. Çiğdem Turhan	(Atılım University)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname : Altuğ Kayışoğlu
Signature : _____

ABSTRACT

LOKMAN: A MEDICAL ONTOLOGY BASED TOPICAL WEB CRAWLER

Kayışoğlu, Altuğ
M.S., Department of Information Systems
Supervisor: Assoc. Prof. Nazife Baykal

September 2005, 92 pages

Use of ontology is an approach to overcome the "search-on-the-net" problem. An ontology based web information retrieval system requires a topical web crawler to construct a high quality document collection. This thesis focuses on implementing a topical web crawler with medical domain ontology in order to find out the advantages of ontological information in web crawling. Crawler is implemented with Best-First search algorithm. Design of the crawler is optimized to UMLS ontology. Crawler is tested with Harvest Rate and Target Recall Metrics and compared to a non-ontology based Best-First Crawler. Performed test results proved that ontology use in crawler URL selection algorithm improved the crawler performance by 76%.

Keywords: Web Crawler, Topical Web Crawler, Topical Web Crawling Algorithms, Ontology Based Crawling, Ontology Based Topical Crawler

ÖZ

LOKMAN: TIBBİ ONTOLOJİ TABANLI ODAKLANMIŞ İNTERNET GEZGİNİ

Kayışoğlu, Altuğ
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Doç. Dr. Nazife Baykal

Eylül 2005, 92 sayfa

Ontoloji kullanımı internette bilgi arama/süzme sorununu çözmek için etkili bir yoldur. Bu tez çalışması, ontoloji kullanarak internet kaynakları üzerinde odaklanmış arama yapmanın internet gezginin performansına katkısını tespit etmeyi amaçlar. Gezgin Best-First algoritmasıyla kodlanmış ve UMLS tıbbi ontolojisine göre tasarlanmıştır. Gezginin performas ölçümü Hasat Oranı (Harvest Rate) ve Hedef Bulma (Target Recall) metriklerine göre gerçekleştirilmiş ve ontoloji tabanlı olmayan başka bir Best-First gezginle karşılaştırılmıştır. Sonuçlar ontoloji tabanlı çalışan gezginin performansının ontoloji kullanmayan gezginden %76 daha yüksek olduğunu ortaya koymuş ve ontolojik bilginin gezgin performansını arttıracağı savını doğrulamıştır.

Anahtar Kelimeler: İnternet Gezgini, Odaklanmış İnternet Gezgini, Odaklanmış İnternet Gezgin Algoritmaları, Ontoloji Tabanlı İnternet Gezgini, Bilgi Geri Getirme

To my family

&

People of my country who
have funded 12 years of my education….

ACKNOWLEDGEMENTS

I thank my advisor Assoc. Prof. Nazife BAYKAL for providing guidance and encouragement which promoted this study. I am grateful to Assoc. Prof. Nazife BAYKAL for her confidence to my study and her illuminating ideas during overall work performed.

I also express my thanks to Timur BEYAN, the most exceptional physician I have ever met, for his precious ideas. This "engineer-minded physician" catalyzed the mental preparation process and helped me a lot by evaluating the system from an outer observer's eye.

And, my old friend and project mate Mustafa KUBİLAY. I enjoyed working, discussing and sometimes even struggling with him throughout every step of this study. His contributions as a companion are invaluable as well as his ideas, comments and technical assistance.

Finally, my family and especially my beloved mother… I can find no phrase to express the value of a warm "hello" coming out of the phone in times of utter stress. Even I am away from them physically, their spiritual and emotional support have no place less than my own effort in this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

API	: Application Programming Interface
BFS	: Breadth-First Search
CUI	: Concept Unique Identifier
DFS	: Depth-First Search
FIFO	: First-In-First-Out
HITS	: Hyperlink-Induced Topic Search
HTTP	: HyperText Transfer Protocol
IP	: Internet Protocol
LAN	: Local Area Network
LIFO	: Last-In-First-Out
MAC	: Media Address Control
NLM	: National Library of Medicine
RAM	: Random Access Memory
TCP/IP	: Transmission Control Protocol/ Internet Protocol
UMLS	: Unified Medical Information System
UMLSKSS	: Unified Medical Information System Knowledge Source Server
URL	: Universal Resource Locator
WebOnt	: Web Ontology Working Group
XML	: eXtended Markup Language

CHAPTER 1

Introduction

1.1 Background, Motivation and Rationale for the Study

Mankind has suffered from many problems since the first human being walked on the earth. Mostly these problems emerged from lack of needed resources. But in today's information world, a new type of problem comes into existence, which is not caused by scarcity but wealth of resources.

For most of the people, idea of having more available resources than the needed amount may seem preferable. It is no doubt that internet is such a repository with many available information resources but when the issue is finding some relevant and reliable pieces inside it, the excessive amount causes problems.

Web is a collection of billions of documents which are linked to each other in an undefined and uncontrollable topology. This massive collection is built up by millions of authors because the hyperspace enabled people publish anything they thought to be useful freely. It has a great impact on our life in means of information sharing. Considering the above advantages, one may wonder what might be bad about the web. To find out the answer to this question, a closer inspection of characteristics of the web would be useful.

Discussing the web is something different from discussing the catalogue of a company's products, past issues of a newspaper or the index of local library. Web has some properties which are characteristic only to it. It is growing day by

day. By July, 2005 Google reports to be searching its index of more than 8.5 billion web pages 0. Another great trouble embedded in the nature of web is, there is no control on removing/changing the pages as there is no control on publishing them. The documents published on the web continuously change. A document labeled as relevant to a topic may not exist or the universal resource locator (URL) for it may indicate another document the other day.

Documents forming web are of different topics, different domains and different languages unlike any collection constructed by humans. In addition, since there are no standards or controls on semantic structure of the documents, inconsistencies can be observed in the content [2].

Under these circumstances, finding some specific document is a real trouble on the web. Many pages look like they are related to the search topic, which are in essence not. On the other hand, too many pages are encountered related to search topic, which need to be filtered and prioritized. Therefore, an effort to satisfy the information need from the web is either like dying of thirst in the middle of the ocean or trying to drink water from a fire hose.

A common internet user usually tries to find out pages of her/his interest through generic/domain search engines or meta-search engines. These systems respond user needs by generating answer sets using the information they get from the system index. In other words, they prepare a list of documents which are indexed with the user query term. Therefore, probability of a random user's satisfaction is directly related to the way he/she represents his need; in more formal terms, how user specifies her/his query.

In communication, there is one golden rule: "The message sender sends is not the expression sent to the communication channel. It is what receiver understands." This statement also summarizes the problem between natural languages and formal languages. When a user specifies her/his query to a search system from the sender side, s/he types down an expression which is affected by

her/his culture, language, personality, personal life, profession, family, religion and so on. But on the receiver side, there are some machines and codes which are expected to send a response in return. Receiver is not affected by the facts given above. It just accepts some input, performs some function and generates an output. Since it does not care about these factors, the satisfaction degree of the user by the query results would be degraded. As an expected conclusion, user finds her/himself stuck in an answer set with many documents some of which belong to other domains, jargons, languages or cultures.

Domain search systems which perform on the web help the users find their way out in such a situation. These systems promise to retrieve documents from a single area of interest. They shrink the index term set to a set of domain specific terminology from an undefined set of user query terms. It is mentioned as undefined because when a general search engine is of concern, the index term set cannot be defined by means of a language or a domain. Another advantage of domain search systems is that they shrink the social factors affecting human thoughts, needs and representation of those to a single domain. Strength and satisfaction rate of domain search systems heavily rely on the representation power of domain knowledge. A weak or an incomplete representation degrades the quality of the answer sets.

MedicoPort project was proposed in September 2004 to find out the advantages of domain knowledge use in medical search on web. It is a medical domain web search engine enhanced with medical ontology. This search engine does not address health professionals but ordinary people who wish to get information on a particular health issue. Basically MedicoPort is designed to restrict the search list to those containing the medical information only. It is not designed for medical literature database search. MedicoPort uses a complete medical lexicon (Unified Medical Language System [UMLS] Specialist Lexicon) and a powerful medical ontology (UMLS Metathesaurus). Therefore it has the ability to take a simple specification of the user need and match it to a concept or a set of

concepts of a context in medical terminology. Also it exploits the advantage of ontology use to filter irrelevant web pages. Unlike most medical search systems on the web, it is not designed to work on medical literature databases or some specific sites. Its design purpose is constructing a domain search engine through which users can traverse web pages related to medicine.

MedicoPort adds the advantages of ontology use to the search engine strategies. It is compound of three major subsystems. These include crawler module, indexing module and retrieval-result ranking module (Figure 1). Other modules in the system exist to support these major subsystems.

This study is performed in order to implement the crawler subsystem used in the project. Crawler has been named as Lokman after the physician who sought the medicine for eternal life all over the world in Turkish mythology. Indexing and search subsystems are implemented by Mustafa Kubilay [3].

1.2 Ontology

1.2.1 Definition

Ontology is a powerful way of modeling information. It is a structured system of categories. Each concept in the domain can be organized through this categorization [4]. A typical ontology has a taxonomy defining the concepts (classes) and a set of inference rules which describe the relations among these concepts [5]. It is designed to define knowledge, reuse it and share it [6]. Ontology is data defining the relations and concepts in a domain.

Controlled vocabulary and semantic search are two of the major use cases of ontology which are defined in [7] by Web Ontology Working Group (WebOnt). From a more social viewpoint, ontology can be considered as a way to make information systems “think” like humans do. They have an important role to fill the gap between human cognition and machine processing. Using ontology, content-based search can be performed.

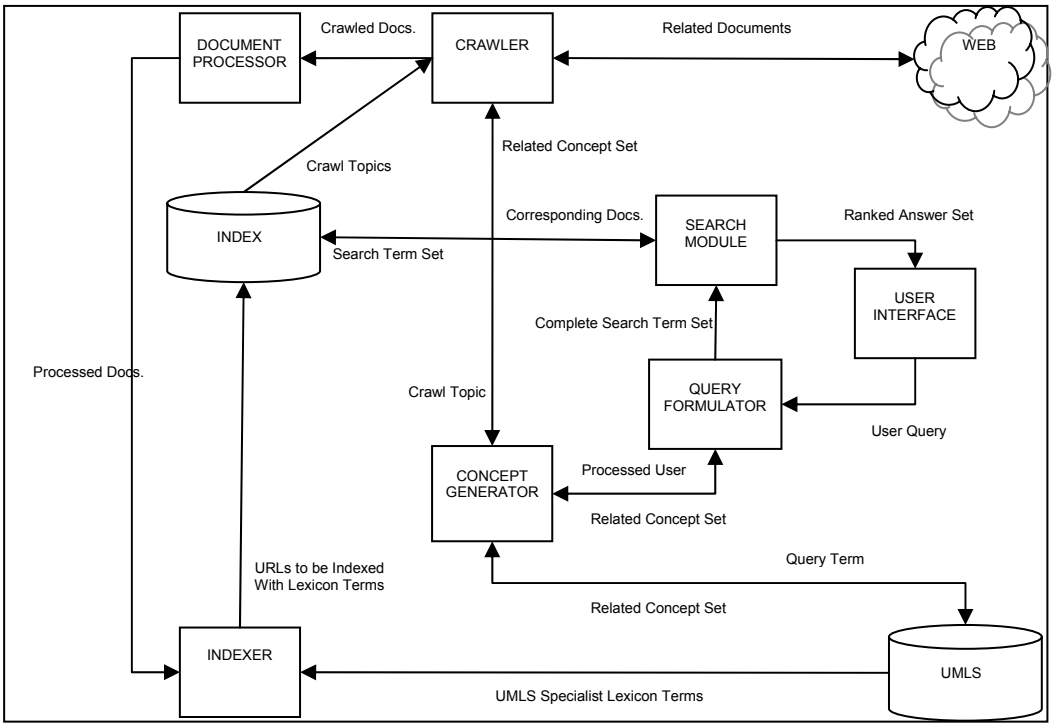


Figure 1: MedicoPort System Overview

1.2.2 Unified Medical Language System

Unified Medical Language System (UMLS) is developed by American National Library of Medicine (NLM). UMLS is designed to "facilitate the development of computer systems that behave as if they 'understand' the meaning of the language of biomedicine and health" [7]. UMLS involves several interrelated sub-components. The system is not optimized or designed to serve a single purpose. Therefore its resources can be utilized by different applications "that create, process, retrieve, integrate, and/or aggregate biomedical and health data and information, as well as in informatics research" [7].

UMLS has three main components: SPECIALIST Lexicon, UMLS Semantic Network and UMLS Metathesaurus.

SPECIALIST Lexicon is a collection of biomedical terminology and medical terms in English [9]. It also covers Lexical Tools which are a set of JAVA programs that help users manage variation in biomedical terms [10].

UMLS Semantic Network involves in the classification of Metathesaurus concepts. It provides the consistent categorization of these concepts and defines the set of relationships that can be defined among them [11].

UMLS Metathesaurus covers the semantic relations between the terms in SPECIALIST Lexicon. It is organized by meaning. Its purpose is linking alternative views of concepts and identifying useful relations between them [12]. It forms a powerful medical ontology whose concepts are defined by the SPECIALIST Lexicon and the relations are provided by the Semantic Network.

1.3 Web Crawling

A crawler is a program that can be directed to automatically find information on the Internet [13]. Crawlers search the web in the same way a person does when s/he uses a browser.

Internet can be regarded as a graph, hyperlinks as edges and pages as vertices. It uses the graph structure of web to move from one page to another [14]. It can also be instructed to make decisions about where to go next based on the information found at each site if it is a topical crawler. They are defined as "software programs that traverse the World Wide Web information space by following hypertext links and retrieving web documents by standard HyperText Transfer Protocol (HTTP)" [15].

Crawlers have also been called as wanderers, robots, spiders, fish and worms. From the beginning the key motivation designing web crawlers has been to retrieve web pages and add them to local repositories for further use.

To start its journey, a crawler needs at least one predefined URL. This URL is called a seed URL. Briefly, crawling begins from a seed page and continues visiting other pages which are referenced by the current page via hyperlinks. This process goes on until a predefined number of pages are visited, a

predefined condition is satisfied or crawl loop is terminated due to some other reason. Crawlers have been shown to be useful in various web applications. There are four main areas where crawlers have been widely used [16]:

1. Personal search: Personal crawlers try to search for Web pages of interest to a particular user.

2. Building collections: Web crawlers have been in action especially for this purpose. To create an index for any search and keep it up-to-date, crawlers collect web pages continuously [17-19]. In addition to building search engine indexes, crawlers can also be used to collect pages that are later processed to serve other purposes.

3. Archiving: "To meet the challenge of the enormous size of the Web, fast, powerful crawlers are developed and used to download targeted Web sites into storage tapes" [16].

4. Web statistics: "Large number of pages collected by crawlers is often used to provide useful, interesting statistics about the Web" [16].

1.4 Thesis Structure

Chapter 2 provides the related research on web crawling strategies and web crawling algorithms. In addition, the concept of 'topical web crawling' and factors driving the information seekers to this concept are introduced in this chapter as well as major topical crawling algorithms and their pseudo-codes.

Chapter 3 includes the analysis study, design principles and a detailed overview of the crawler system.

Chapter 4 covers a detailed discussion of the implemented system, introduction of crawler evaluation metrics and evaluation of the implemented crawler with empirical results.

Chapter 5 provides the conclusions, possible future work directions for the system and ontology based search agents on the web and function of the study in means of collaborative effort in the MedicoPort project.

CHAPTER 2

Literature Survey

Web crawling is a complex problem to tackle. There are many factors affecting web crawling strategies. Most of these factors are identical only to web. In this chapter, characteristics of the web, similarities between graph traversal and web traversal, primitive web crawling algorithms, concept of topical web crawling and topical web crawling algorithms are presented sequentially.

2.1 Characteristics of the Web

Since 1990's, web has become the prime medium for information exchange all over the world. After the wide use of internet, users gained the ability both to publish their information easily and to access published information eliminating difficulties like geographical distance.

On the other hand, the advantages mentioned above formed the core of problematic nature of the web:

2.1.1 Rapid Change Rate

Storing web pages or any kind of representation of them in a repository is a challenging task when the very rapid changing characteristic of the web is considered. This rapid change has two factors: first is the update rate of the existing web pages and second is recently added/removed pages. That means, a URL with a preferred content which is claimed to exist by a repository may not exist or may have a different content. The percentage of such invalid links

stored in search engine indices varies from 2 to 9% [20]. Brewington and Cybenko stated that a re-indexing period of 8.5 days is necessary to keep a search-engine index up-to-date [21]. Brewington and Cybenko used the estimation that the size of the web is 800 million pages by [22] but web has grown much larger than this estimation today.

Another study performed on half a million pages over a 4 months time revealed that 23% of pages changed daily [23]. It was also observed that half-life for a web document in .com domain was 10 days and 40% of the pages in this domain changed daily [23].

For any application using such repository, up-to-date information is required. Therefore crawlers are heavy workers trying to cope with the pace of the change working with no brakes. With their effort, the repositories are updated and modified.

2.1.2 Massive Document Collection

As mentioned in Chapter 0, publishers have no restrictions on the number of documents to publish on web. This fact makes the collection grow in a high speed. Another reason that enforces the growth is the developments achieved in storage media and bandwidth. Larger storage capacity enables publishers put more and more documents on the web. More bandwidth enables information seekers access documents kept on distant storage media faster and easier.

Studies on estimating the size of web has began in late 1990's. In 2000, it was estimated that the size of the static web is 4 to 10 billion web pages [24]. By July 2005, Google reports to be searching 8.5 billion web pages 0.

2.1.3 Inconsistent & Incoherent Document Collection

Before web, document collections were about a topic like newspaper articles, library book catalogues or medical literature. Such coherency not only enabled users access required information by using domain-specific terminology but also enabled them classify and sort them according to their information needs. Web,

from this point of view, is not coherent. It is a document collection with many topics and interests. One can find a document with any topic on web. Therefore, search on web is far much challenging than search on a coherent collection.

Another problem that makes the problem more challenging is the inconsistencies in the documents on web. These include mistyping and misuse of domain terminology. Documents, which include mistyped words or misused terminology, cause troubles to automatic search and information retrieval systems.

2.2 Web Crawling Algorithms

Traditional graph search algorithms have been extensively applied to computer science. Since web can be considered as a directed graph with a set of nodes (pages) connected with directed edges (hyperlinks) [16], graph search algorithms can be applied to crawling.

The base algorithm executed by a web crawler takes a list of seed URLs defined by the operators/system administrators and repeatedly executes the following steps [18] (Figure 2):

- Remove a URL from URL list (URL Frontier),
- Determine the Internet Protocol (IP) address,
- Download the corresponding document,
- Extract links in the document,
- For each link in the document,
    - Ensure that the link is a valid URL,
    - Add new URL to the URL list,
- Go to step 1.

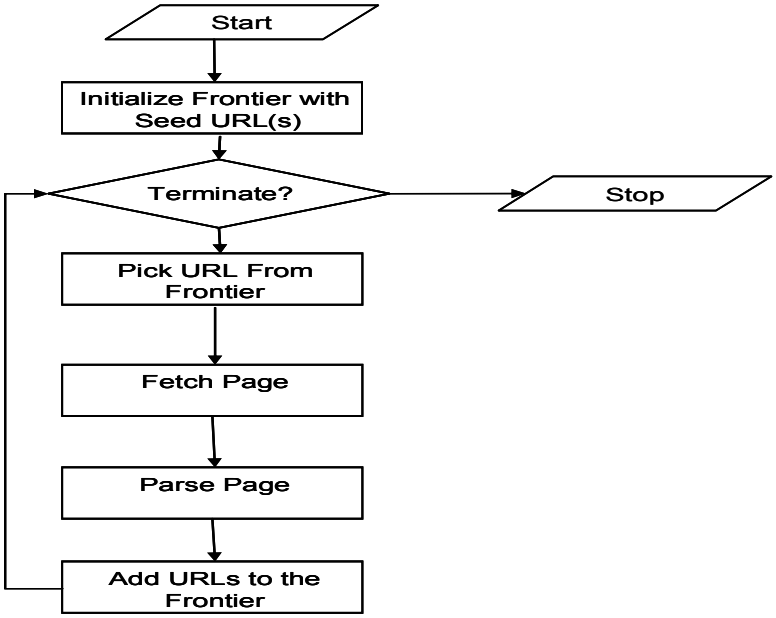


Figure 2: Flow of a Basic Sequential Crawler [14]

2.2.1 Breadth-First Search

Breadth-First search (BFS) is a tree search algorithm used for traversing or searching a tree (graph theory) or tree structure. It considers neighbors of a vertex (outgoing edges of the vertex's predecessor in the search) before any outgoing edges of the vertex [25]. Extremes are searched last. URL frontier for such a crawler is typically implemented with a first-in-first-out (FIFO) structure, namely queue. That is to mean crawler starts at the root node (vertex) and explores all the neighboring nodes. Then for each of those nearest nodes, the crawler explores their unexplored neighbor nodes, and so on until it finds the goal.

Formally, BFS is an uninformed search method that aims to expand and examine all nodes of a tree systematically in search of a solution. In other words, it exhaustively searches the entire tree without considering the goal until it finds it. It does not use a heuristic to order URLs [26][26-28].

For web search, a breadth-search crawler is the simplest strategy for crawling. It uses a FIFO queue as the frontier and crawls the links as they are appended

(Figure 3). Since it does not use any heuristics, which is to mean any knowledge about the topic, its performance is lower than more sophisticated crawlers [29]. Pseudo-code for breadth-first algorithm is presented in Figure 4.

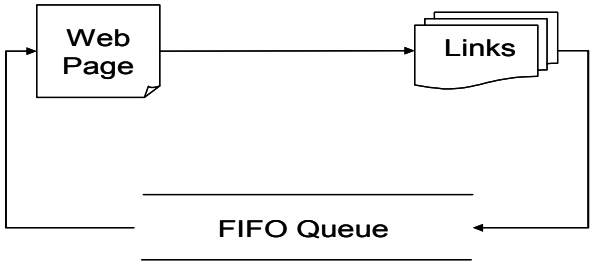


Figure 3: A Breadth-First Crawler [29]

2.2.2 Depth-First Search

Depth First Search (DFS) is a way of traversing or searching a tree (graph theory), tree structure, or graph. It considers outgoing edges of a vertex before any neighbors of the vertex that is, outgoing edges of the vertex's predecessor in the search [30, 31]. Extremes are searched first. It can easily be implemented with recursion. Intuitively, a depth-first crawler starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

```
Breadth-First (starting_urls){
  foreachlink(starting_urls){
    enqueue (frontier, links);
  }
  while (visited < MAX_PAGEs){
    link = dequeue (link, frontier);
    doc = fetch (link);
    enqueue (frontier, extractlinks(doc));
    if (# frontier > MAX_BUFFER){
      dequeue_last_links (frontier);
    }
  }
}
```

Figure 4: Pseudo-code for Breadth-First Algorithm [29]

From an algorithmic point-of view, all freshly expanded nodes are placed at the front of the search queue for expansion.

Space complexity of DFS is much lower compared to BFS (Breadth-first search). When searching large graphs that cannot be fully contained in memory, DFS suffers from non-termination when the length of a path in the search tree is infinite. The simple solution of "remember which nodes I have already seen" does not work because there can be insufficient memory. This can be solved by maintaining an increasing limit on the depth of the tree, which is called iterative deepening depth-first search [30, 27, 28].

For a depth-first crawler, a last-in-first-out (LIFO) structure or a stack shall be used as the URL frontier. This way the crawler always keeps on a track until it finds a page with no links out of it. Regarding that approximately each web page consists 7 links out [20], it is obvious that a depth-first crawler may never turn back. Therefore, maximum depth of the crawler is usually defined. Depth-first crawlers have the same structure and pseudo code with the breadth-first crawlers as given in Figure 3 and Figure 4. The only difference amongst them is the data structure of the URL frontier.

2.3 Topical Crawling

It is a fact that the users inspect only a small portion of the retrieved results in the result set which are the top hits. Therefore ranking pages is an important issue. First approach to rank search results was binary ranking. This method classifies documents in the result set as related or not with no ranking among them. For search engines, as clearly seen, it is not a proper way because of the user behavior. In addition, it is obvious that the relevance of different documents to a given query cannot be entirely the same. These facts brought several ranking models which made search engines more sophisticated.

But before ranking the results for user's choice, a search system needs to create and fill an index for the web pages. Suppose that there is such a ranking algorithm that is capable to sort the result set for a given query very close to human cognition and there are several web documents related to the given query (test documents). The system is expected to retrieve a set of results with the test documents on the top of the list. But even with such an algorithm, the result may not be as expected because the test documents may not even exist in the index of the engine.

The problem above is related with the size and the growth rate of the web. It is impossible to index the entire web although dramatic improvements have occured in the storage media, bandwidth and other hardware technologies and resources.

At this very point, to solve the problem defined above, a system which is clever enough to search and retrieve the documents which are in user's area of interest is required. This may seem to be a simple approach, but it shrinks the document space and enables the system find better matching documents in it. Such a crawling on web is called as "topical crawling" or "focused crawling". In the remainder of this thesis, term "topical crawling" is preferred. Briefly, topical crawling is an algorithm to find pages of interest from the web.

Algorithm of topical crawling can be specified as below [32]:

- Specify user interest
- Use a priority queue of URLs based on the rank of the URL.
- Start with a set of seed pages in the queue
- Get top rank URL from the queue
- Download the page P and extract URLs {U} from the page
- Measure relevance of the page P and each URL in {U}

- If relevance of the page P is above threshold then set P as one of the results
- For each URL in {U}, if it already exists in the queue then update its rank. Else add the URL to the queue
- Go to step 4

2.4 Topical Crawling Algorithms

2.4.1 Best-First Search

Best-First Search is a state-space search algorithm that considers the estimated best partial solution next [33]. It optimizes breadth first search by ordering all current paths according to some heuristic. The heuristic attempts to predict how close the end of a path is to a solution. Paths which are judged to be closer to a solution are extended first. Efficient selection of the current best candidate for extension is typically implemented using a priority queue [34].

Table 1: Priority Queue Representations [28]

Representation	Insertion	Deletion
Unordered Array	O (1)	O (n)
Unordered Linked List	O (1)	O (n)
Sorted Array	O (n)	O (1)
Sorted Linked List	O (n)	O (1)
Max/Min Heap	O ($\log_2 n$)	O ($\log_2 n$)

A priority queue deletes the element with the highest/lowest priority. At any time any element with arbitrary priority can be inserted to the queue. If the application requires the element with the highest priority to be deleted, a maximum (max) heap data structure is preferred. On the opposite, a minimum (min) heap is used. Arrays and linked lists can also be used to implement priority queues but the computational cost in deletion becomes quite high [28] (Table 1).

Different best-first crawlers can be implemented according to their strategy but the change is in only the heuristics they apply. The structure remains the same as in Figure 5.

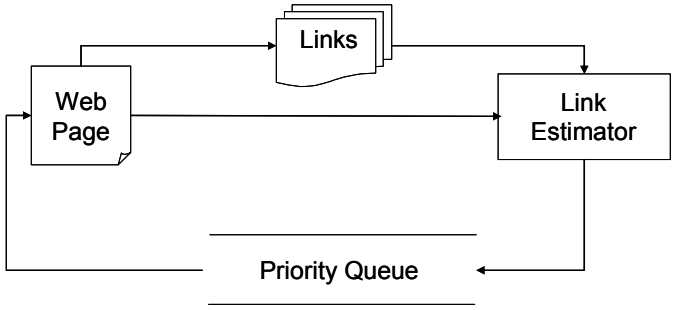


Figure 5: A Best-First Crawler [29]

$$\text{Sim}(q,p) = \frac{\sum_{k \in q \cap p} f_{kq} f_{kp}}{\sqrt{\left(\sum_{k \in p} f^2_{kp}\right)\left(\sum_{k \in f} f^2_{kq}\right)}} \qquad \text{Equation 1}$$

```
BFS (topic, starting_urls){
  foreachlink(starting_urls){
    enqueue (frontier, link, 1);
  }
  while (visited < MAX_PAGEs){
    link = dequeue_top_link (frontier);
    doc = fetch (link);
    score = sim (topic, doc);
    enqueue (frontier, extractlinks(doc), score);
    if (# frontier > MAX_BUFFER){
      dequeue_bottom_links (frontier);
    }
  }
}
```

Figure 6: Pseudo-code for a Best-First Crawler [29]

In [29], Menczer et.al. implemented a best-first crawler in which link selection process is simply done by computing the lexical similarity (Equation 1, q is the topic, p is the fetched page and f_{kd} is the frequency of term k in document d)

between the topic's keyword and the source page for the link. This crawler represents a web page using the vector-space model [20]. Thus the similarity between a page *p* and the topic keywords is used to estimate the relevance of the pages pointed by *p*.

After all estimations are calculated, the URL with the best result is selected for crawling. Pseudo-code for a best-first crawler is given in Figure 6.

2.4.2 PageRank

This algorithm was proposed by Brin and Page in [17]. It is designed to model a possible model of user surfing behavior. It makes use of the graph structure of the web to calculate a quality ranking for each web page. A page's PageRank score depends on the PageRank scores of the pages which refer to it [17, 28].Calculation of PageRank score of a page is performed as follows:

$$PR(A) = (1-d) + d(PR(T_1)/C(T_1)+\ldots\ldots+ PR(T_n)/C(T_n)) \text{ [17]}$$

Here A stands for the page whose PageRank score is computed. Pages from T_1 to T_n represent the web pages that point to page A and *d* is a damping factor such that $0< d <1$. $C(T_1)$ represents the number of links going out of page T_1. In more formal terms, PageRank calculation is given in Equation 2 where γ stands for the damping factor.

A page can have a high PageRank score if it is linked from many other pages. Also if the PageRank scores of the pages pointing the focus page are high, this improves the score of the focus page. In simpler words, the quality of any page is a function of the quality of each page pointing it. Pseudo-code for a PageRank crawler is presented in Figure 7.

$$PR(p) = (1 - \gamma) + \gamma \sum_{d \in in(p)} \frac{PR(d)}{out(d)} \qquad \text{Equation 2}$$

2.4.3 HITS

The HITS (hyperlink-induced topic search) algorithm was first introduced by John M. Kleinberg in [36]. He assumes that a topic can be roughly divided into pages with good coverage of the topic, called authorities, and directory-like pages with many hyperlinks to useful pages on the topic, called hubs.

The goal of HITS is basically to identify good authorities and hubs for a certain topic which is usually defined by the user's query. So, HITS is a query-based algorithm [37, 16].

```
PageRank (topic, starting_urls){
  foreachlink(starting_urls){
    enqueue (frontier, link, 1);
  }
  while (#frontier > 0 and visited < MAX_PAGES){
    if (multiplies_25 (visited)){
      foreach link (frontier){
        PR(link) = recompute_PR;
      }
    }
    link = dequeue_link_with_max_PR (frontier);
    doc = fetch_new_document (link);
    score = sim (topic, doc);
    if (#buffered_pages > MAX_BUFFER){
      dequeue_link_with_min_PR (frontier);
    }
    enqueue (frontier, extractlinks(doc), score);
    for each outlink (extract_new_links(doc)){
      if (# frontier >= MAX_BUFFER){
        dequeue_link_with_min_PR (frontier);
      }
      enqueue (frontier, outlink);
    }
  }
}
```

Figure 7: Pseudo-code for a PageRank Crawler [29]

Given a user query, the HITS algorithm first creates a neighborhood graph for the query. The neighborhood contains top 200 matching web pages retrieved from a content-based web search engine as well as all the pages these 200 web pages linked to and pages that refer to these 200 top pages.

$$A(p) = \sum_{q(q,p)\in G} H(q) \qquad \text{Equation 3}$$

$$H(p) = \sum_{q(p,q)\in G} A(q) \qquad \text{Equation 4}$$

Then for each page a hub score (Equation 4) and an authority score (Equation 3) is computed. Briefly, authority score of a page P is, sum of hub scores of pages pointing page P relative to query Q. Hub score of a page P is, sum of authority scores of pages to which P refers to relative to query Q.

Like PageRank, HITS is also very effective in ranking search results. But both of these algorithms suffer from computational expenses due to iterative score calculating.

2.4.4 Shark-Search

Shark-search algorithm was proposed by [37] in 1998 as an enhanced and more aggressive version of Fish-Search [38]. In Fish-Search, crawlers search more extensively in the areas of the web in which relevant pages are found while they stop crawling the areas which are not relevant. What Shark-Search adds to the main approach are, instead of Fish-Search's binary relevance function, Shark-Search uses a continuous relevance function and it has a more refined notion of potential scores for the links in the frontier. Pseudo-code for a Shark-Search crawler is presented in Figure 8.

```
Shark (topic, starting_urls){
  foreachlink(starting_urls){
    set_depth (link, d);
    enqueue (frontier, link, 1);
  }
  while (visited < MAX_PAGES){
    link = dequeue_top_link (frontier);
    doc = fetch_new_document (link);
    doc_score = sim (topic, doc);
    if (depth(link) > 0){
      foreach outlink(extract_links(doc)){
        score = (1-r) * neighborhood_score(outlink)
                  + r * inherited_score(outlink);
        if (doc_score > 0){
          set_depth (outlink, d);
        }
        else{
          set_depth(outlink, depth(link)-1);
        }
    enqueue (frontier, outlink, score);
      }
      if (#buffered_pages > MAX_BUFFER){
      dequeue_link_with_min_PR (frontier);
      }
    }
  }
}
```

Figure 8: Pseudo-code for a Shark Search Crawler [29]

CHAPTER 3

Lokman Crawler System

Chapter 3 starts with the analysis study performed for the system. Analysis study involves specifying purpose and the scope of the system, assumptions and dependencies and functional requirements.

Issues on design and implementation based on the functional requirements are depicted following the analysis study.

3.1 Purpose and Scope

Many crawlers have been implemented in order to serve for different applications with the same purpose of 'downloading the most important pages from web'. They use different algorithms introduced in section 2.4.

An important point to be emphasized at this point is the fact that crawling is the effort to estimate the value of an unvisited URL U by information obtained from a visited page V which contains a hyperlink pointing to U. Therefore quality of a visited page affects the crawler's possibility of visiting better qualified pages.

Prime factor that affects algorithm selection and modification for a crawler is the purpose of the super system the crawler serves for. Since MedicoPort system aims to find out the use of ontology in medical information retrieval, information obtained from ontology should be used in Lokman's URL ordering process.

Algorithms introduced in section 2.4 can be classified as lexical similarity based (Best-First Search, Shark-Search) and back-link/forward-link count based ones (HITS, PageRank). Enhancing relevance degree of a page to a given topic using ontological information is one of the purposes of the study. Therefore URL selection of Lokman should be performed using an algorithm based on lexical similarity.

Lokman should not be implemented as a query agent running on databases available through web. It should be designed to find out pages of interest from the static web.

System should obtain context information for a given search topic from Unified Medical Information System Knowledge Source Server (UMLSKSS) in order to use in further processing. Information exchange between UMLSKSS and the system should be done by remote request instead of local installation of the UMLSKSS resources.

Crawler should use text data obtained from the downloaded pages. Other type of information existing in the page should be out of scope and interest.

3.2 Assumptions and Dependencies

3.2.1 Assumptions

It is assumed that crawler shall be implemented as a single-threaded crawler.

It is assumed that no restrictions on downloaded web document size have been specified.

It is assumed that a maximum URL number of 10000 is specified as the size of the URL Frontier.

It is assumed that the computer system on which Lokman is run has an active internet connection.

It is assumed that all operations for the crawler shall be performed on RAM.

It is assumed that no disc access operation for downloaded documents shall be performed.

3.2.2 Dependencies

Crawler performance is affected by any possible UMLSKSS failures and any possible change in terms of use of UMLSKSS resources.

Crawler shall be run on a computer system which is registered to UMLSKSS with its IP and Media Access Control (MAC) addresses.

Crawler shall be run on a computer system which is identified to UMLSKSS with a valid UMLS Licensee identification.

Since UMLSKSS Developer's Application Programming Interface (API) requires JAVA 1.4 or higher to run, the system shall be implemented with a JAVA version higher than JAVA 1.4.

3.3 Functional Requirements

1. The system shall accept a new search topic after each crawl loop from the MedicoPort Index.

2. The system shall establish a connection to UMLSKSS.

    2.a. The system shall establish a connection to UMLSKSS using parameters host = 'umlsks.nlm.nih.gov' and client = 'KSSRetriever'.

    2.b. If the connection cannot be established, the system shall generate an error message including the reason for unsuccessful connection such as inactive host, unidentified client (user unregistered to UMLSKSS) or malformed host URL.

2.c. If connection establishment is successful the system shall generate a notification message.

3. The system shall run queries in order to get an Extended Markup Language (XML) response including the related concept set for the given search topic from UMLSKSS.

3.a. The system shall set other query parameters than search topic automatically.

3.a.i. The system shall set ‘dbYear’ to ‘2005AB’. This value determines the UMLS Release to be used in the search.

3.a.ii. The system shall set ‘Language’ to ‘ENG’. This parameter indicates the language for the query.

3.a.iii. The system shall set ‘Sabs’ to ‘(MSH, SNOMEDCT)’. These values indicate the source medical thesaurus from which concepts root.

3.a.iv. The system shall set ‘Operator’ to ‘1’. This value indicates the term matching criterion with the given search topic. 1 means an exact match.

3.b. The system shall determine the UMLS Concept Unique Identifier (CUI) for the given search topic before running any other query on UMLSKSS.

3.c. The system shall determine the concepts which are related to the given search topic according to UMLS Metathesaurus.

3.d. The system shall determine the concepts which are in same context with the given search topic according to UMLS Metathesaurus.

3.e. The system shall determine the concepts which are synonym to the given search topic according to UMLS Metathesaurus.

4. The system shall accept the XML response from UMLSKSS as a stream.

5. The system shall process the XML response to derive out concept list.

    5.a. The system shall remove the query tags existing in the XML response.

    5.b. The system shall pick the concept names within <SY></SY> and <PAR></PAR> tags which indicate a synonymy and partial relevance sequentially for any UMLSKSS relations query.

    5.c. The system shall pick the concept names within <CXT></CXT> tags that has the rank value of 0 between <rank></rank> tags which indicates contextual relevance for any UMLSKSS context query.

    5.d. The system shall pick the concept names between <cn></cn> tags which indicates a synonymy or term variance (concept name) for any UMLSKSS terminology query.

    5.e. The system shall keep a temporary record for each concept obtained from the XML response labeling them as synonym, partially relevant or contextually relevant.

    5.f. The system shall remove any duplication in the final concept list.

6. The system shall return the final concept list to link evaluation module.

7. The system shall have a seed URL to initialize the crawling process.

8. The system shall assign an initial value to the seed URL.

9. The system shall add the seed URL with the determined value to the URL Frontier.

10. The system shall fetch the corresponding web page to the seed URL.

    10.a. The system shall send a HTTP request to the URL.

    10.b. The system shall generate an error message if any TCP/IP communication problem occurs.

    10.c. The system shall generate an error message if any malformed URL is found.

    10.d. The system shall accept the HTTP response through port 80.

    10.e. The system shall send the fetched document to MedicoPort Document Processor

11. The system shall keep a log of visited URLs (URL History).

    11.a. The system shall initialize the URL History after system startup.

    11.b. The system shall load the URL History to RAM.

    11.c. The system shall save a hash value for each visited page.

12. The system shall convert the document to lower case.

13. The system shall analyze the fetched document.

    13.a. The system shall count each of the concepts existing in the concept list returned from UMLSKSS in the page.

    13.b. The system shall compute values per each concept.

    13.c. The system shall assign following factors to each concept category, 15 for the exact search phrase, 12 for synonymy, 5 for partial relevance and 3 for contextual relevance.

13.d. The system shall set the page value to sum of the values computed in 13.c.

14. The system shall parse the fetched document.

14.a. The system shall derive each hyperlink out from the page.

14.b. The system shall check if the hyperlink has been visited before.

14.c. If the hyperlink has not been visited yet, the system shall analyze the hyperlink by using the information between <a> tags.

14.c.i. The system shall count each of the concepts existing in the concept list returned from UMLSKSS in the defined field.

14.c.ii. The system shall compute tf/idf values per each concept for the defined field.

14.c.iii. The system shall assign following factors to each concept category as in 13.c.

14.c.iv. The system shall set the hyperlink value to sum of the values computed in 14.c.iii.

14.c.v. The system shall update the hyperlink value as the greater of page's value and value computed in 14.c.iv.

14.d. The system shall send the (URL, URL value) tuple to URL Frontier.

15. The system shall insert the (URL, URL value) tuple to URL Frontier.

15.a. The system shall check if the new URL exists in the frontier.

15.b. If new URL exists in frontier, weight value for this URL shall be updated.

15.b.i. The system shall remove the URL from the frontier.

15.b.ii. The system shall set the new value for the URL as greater of existing value and the sent value.

15.b.iii. The system shall insert the URL and new value computed in 15.b.ii.

15.c. If the URL does not exist in the frontier, the system shall insert the new URL to the frontier.

16. The system frontier shall be implemented as a priority queue.

17. The system frontier shall be implemented as a max heap, with the maximum valued element at the top of the queue.

18. The system shall initialize the next crawling loop by sending the top queue element to the fetcher.

19. The system shall remove the top element after sending it to the fetcher.

20. The system shall not allow number of URL Frontier elements exceed the defined MAX_URL.

21. The system shall continue crawling until URL Frontier is empty.

22. If the URL Frontier is empty, the system shall initialize URL History.

23. If the URL Frontier is empty, the system shall get the following crawl topic from MedicoPort Index.

3.4 System Design and Implementation

Level 0 and Level 1 data flow diagrams (DFD) for Lokman crawler system are provided in Appendix A and Appendix B respectively.

3.4.1 Decomposition Description

The system is developed with four packages:

Crawl: This package covers the classes that form prime crawler components, Fetcher, URL Frontier and URL History.

Identification	:	Crawl
Type	:	Package
Purpose	:	Framework for basic crawler activities.
Function	:	Performs basic crawler functions (Fetching, URL Ordering, keeping URL logs.).

Document_Parser: This package covers the document processing features embedded in the system. Features include Document Parser (Link Extractor) and XML Parser.

Identification	:	Document_Parser
Type	:	Package
Purpose	:	Framework for document parsing activities.
Function	:	Performs URL extraction and XML Parsing.

UMLSKSSConnector: This package includes the classes related with UMLSKSS. Classes in this package establish/terminate a UMLSKSS connection and run queries on it.

Identification	:	UMLSKSSConnector
Type	:	Package
Purpose	:	Framework connecting and interacting to UMLSKSS.

Function	:	Performs establishing connection to remote UMLS resources and runs queries on UMLSKSS databases.

Utility: This package includes the utility features used by other packages. These features cover concept counter, comparator computation, special data structures like LinkInfo and ConceptInfo used for information interchange among the classes.

Identification	:	Utility
Type	:	Package
Purpose	:	Framework that keeps utility features for the system.
Function	:	Contains several utility features for other classes.

3.4.2 Dependency Description

Dependency description involves classes as design entities, and provides details related to classes of packages for Lokman crawler system by addressing identification, type, purpose, function, subordinates, dependencies, and resources design entity attributes [39].

Dependencies among classes are given in Appendix C to Appendix R in form of class diagrams.

3.4.3 Detailed Description

With the scope, purpose and requirements defined above, Lokman crawler system is designed as in Figure 9.

3.4.3.1 UMLS Connector

UMLS Connector module is the component that interacts with UMLSKSS. UMLSKSS allows remote users and applications to run queries through HTTP protocol on its resources and returns XML responses.

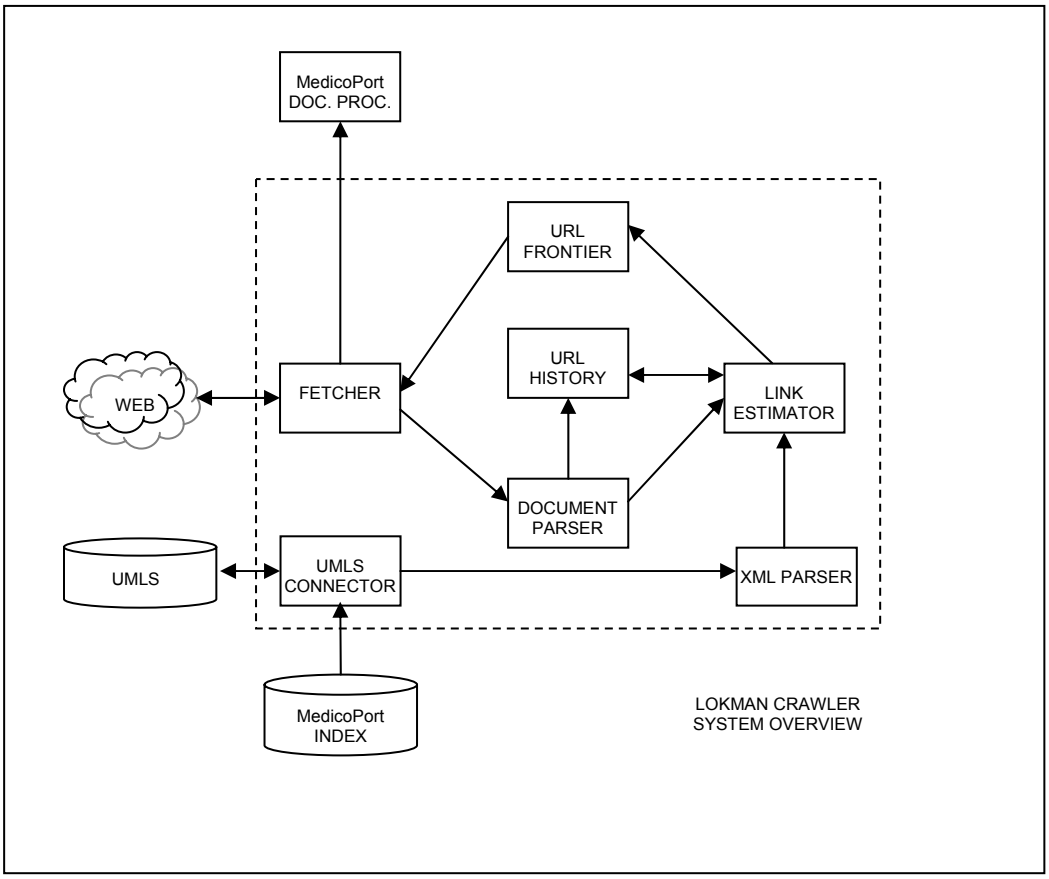


Figure 9: Lokman Crawler System

For applications, UMLSKSS provide Developer's API (For further information on terms of use of UMLSKSS, see [40]). UMLSKSS Developer's API is a set of interfaces exploiting advantages of JAVA. It makes use of JAVA's remote method invocation capability (java.rmi package). Using such an infrastructure allows remote users run queries on NLM's local resources without spending extra effort on understanding complex relations among the databases.

UMLS Connector module is encapsulated by UMLSKSSConnector package. It is compound of two classes: KSSConnector (Appendix F) and KSSQuery (Appendix G). KSSConnector involves in connection issues with UMLSKSS and KSSQuery involves in running queries on UMLSKSS.

Package UMLSKSSConnector Class KSSConnector:

Identification	:	KSSConnector
Type	:	Class
Superclass	:	Object Class
Purpose	:	This class handles the connection to UMLSKSS
Function	:	Establishes or terminates connection to UMLSKSS
Subordinates	:	private static final String clientName private static final String hostName public KSSRetriever makeUMLSKSConnection() public static void terminateUMLSKSConnection (KSSRetriever retriever)
Dependencies	:	gov.nih.nlm.kss.api
Resources	:	java.io java.net java.rm gov.nih.nlm.kss.api gov.nih.nlm.kss.util

Package UMLSKSSConnector Class KSSQuery:

Identification	:	KSSQuery
Type	:	Class
Superclass	:	Object Class
Purpose	:	This class runs queries on UMLSKSS
Function	:	Runs different queries on UMLSKSS, merges the responses and accepts the responses
Subordinates	:	public void showKSSQueryProperties() public void setKSSQueryParameters(String searchTerm) public KSSQuery(String searchTerm) public KSSQuery() public void runKSSQuery() public void runKSSQuery(String searchTerm)

		private void filterConcept(String xmlText)
		private void getCui(String dbYear, String searchTerm, Vector srcs, String language, int op)
		private void performTerminologyQuery(String dbYear, String searchTerm, Vector srcs, String language)
		private void performRelationQuery(String dbYear, String searchTerm, Vector srcs, String language, int op)
		private void performContextQuery(String dbYear, String searchTerm, Vector srcs, String language, int op)
		public String sendXML()
		public ArrayList getResultConceptSet()
		public void printConcept()
		public void setXml(String str)
Dependencies	:	UMLSKSSConnector.KSSConnector
		Document_Parser.UMLSConceptXmlParser
Resources	:	java.io
		java.net
		java.rmi
		gov.nih.nlm.kss.api
		gov.nih.nlm.kss.util
		java.util.Vector
		java.util.ArrayList
		Document_Parser
		java.util.Iterator

First task of UMLS Connector is establishing a sound connection to UMLSKSS to make to and fro information exchange possible. UMLSKSS is free of charge and open to remote users but it requires user registration. User registration is performed via internet and users are granted UMLS Licensee Numbers. Any application that uses UMLSKSS resources shall be identified to the system by a valid license number and IP number. UMLSKSS checks if any connection

attempt is sourced by a registered IP or not. If so, access to resources is granted, else system sends a database exception terminating the connection attempt.

UMLS Connector module uses the interfaces provided in Developer's API to interact UMLSKSS. It runs 3 types of queries among different queries available about the concept to be searched on the web on UMLS Metathesaurus.

Queries run on UMLSKSS resources include terminology search, relations search and context search:

- getTerminology (String dbYear, String searchTerm, Vector srcs, String language);
- getRelations (String dbYear, String Cui, Vector srcs, String language, String relationType)
- getContext (String dbYear, String Cui, Vector srcs);

To get response for two of the queries, system shall provide CUI for the given search topic. This task is accomplished by running a CUI query on the database:

- findCui (String dbYear, String searchTerm, Vector srcs, String language, int op)

Table 2: UMLSKSS Query Parameters

Parameter	Parameter Explanation	Value	Value Explanation
dbYear	UMLS Release Version	2005AB	2nd Release for Year 2005
srcs	Source Medical Thesauri	MSH SNOMEDCT	MeSH SNOMED-CT
language	Language for the Query	ENG	English
op	Type of the Word Matching	1	Exact Match

Rest of the parameters sent to UMLSKSS, their meaning and set values are as in Table 2.

To lower the communication overhead, UMLS Connector module runs all queries one after another on the database and terminates the connection after this process. Responses to each of the queries are merged before sending them to XML Parser module.

3.4.3.2 XML Parser

XML Parser Module is designed to process the query results obtained from UMLS Metathesaurus. It is the module that interprets the results of the queries run on UMLSKSS by UMLS Connector. It is implemented using UMLSConceptXmlParser class (Appendix M).

Package Document_Parser Class UMLSConceptXmlParser:

Identification	:	UMLSConceptXmlParser
Type	:	Class
Superclass	:	Object Class
Purpose	:	This class parses the UMLSKSS query responses
Function	:	Extracts concept names fulfilling pre-defined requirements
Subordinates	:	public UMLSConceptXmlParser(String xml , String searchTerm) public ArrayList extractConcept() private String getXmlLine()
Dependencies	:	UMLSKSSConnector.KSSQuery
Resources	:	java.util java.io Utility UMLSKSSConnector

XML responses to the given queries in 3.4.3.1 include a query header and a response section (Figure 10).

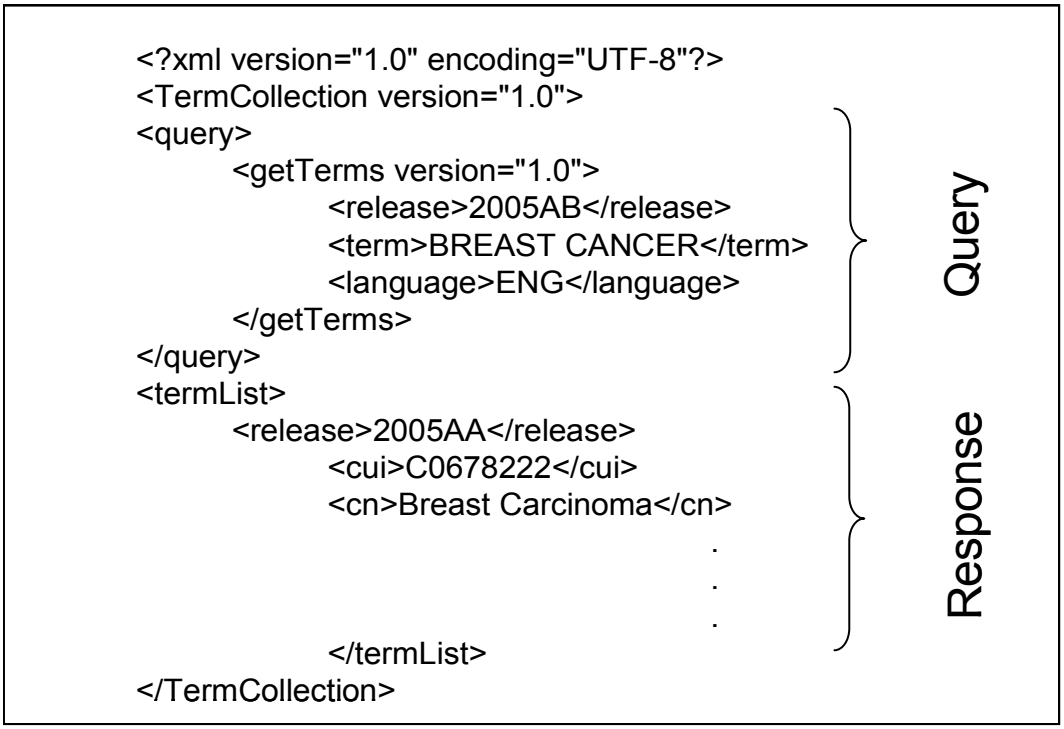


Figure 10: Sample UMLSKSS XML Response for a "getTerminology" Query (Parameters: dbYear = 2005AB, Language = ENG, searchTerm = 'BREAST CANCER')

Parsing differs for each of the queries. For a terminology query, the sub-sections of the response section are classified by related terms. In addition term variants appear under these terms. Related terms appear between <cn></cn> tags where variants are between <tn></tn> tags (Figure 11). Therefore the meaningful part for this query for the system are these concept names between <cn></cn> tags. XML Parser extracts these concept names for a getTerminology query response by seeking these tags and getting the content between them.

For a getRelations query, response section of the XML stream includes related concepts, relation types and other concept information. Concept names appear between <cn></cn> tags. Relation types are placed between <rel></rel> tags (). For the system, needed concept names are the ones with relevance values as SY (synonymy) or PAR (partial relevance). Names of concepts fulfilling this requirement are picked from the XML response.

```
<cui>C0678222</cui>
<cn>Breast Carcinoma</cn>
<term>
        <lui>L0006142</lui>
        <tn>Cancer of the Breast</tn>
        <ts>S</ts>
        <lat>ENG</lat>
        <termVariant>
                <sui>S0020508</sui>
                <stt>VO</stt>
                <str>Breast Cancer</str>
                <strSource>
                        <sab>NCI</sab>
                        <scd>C4872</scd>
                        <srl>0</srl>
                        <srcinfo>
                                <aui>A4479049</aui>
                                <tty>SY</tty>
                        </srcinfo>
                </strSource>
        </termVariant>
                .
                .
</term>
```

Figure 11: Sample UMLKSS Response for a "getTerminology" Query for "BREAST CANCER"

A context query response includes information about the context of the given search topic. A concept name appears between <cxs></cxs> tags in context search query responses. Contextual distance to a given topic and a retrieved topic is stated between <rank></rank> tags (Figure 13). Names of concepts with rank value 0 are picked through parsing process by XML Parser.

```
.
.
<relSource>
        <cui>C0006826</cui>
        <cn>Malignant Neoplasms</cn>
        <aui>A0594095</aui>
        <type>AUI</type>
        <rel>PAR</rel>
        <rui>R05218150</rui>
</relSource>
.
.
```

Figure 12: Sample UMLKSS Response for a "getRelations" Query for "BREAST CANCER"

```
.
.
<cxtMember>
        <cxs>Malignant tumor of breast</cxs>
        <cui2>C0006142</cui2>
        <aui2>A3042037</aui2>
        <rank>0</rank>
        <hcd></hcd>
        <rel>isa</rel>
        <xc>+</xc>
</cxtMember>
.
.
```

Figure 13: Sample UMLKSS Response for a "getContext" Query for "BREAST CANCER

3.4.3.3 Fetcher

Fetcher Module is the component that interacts with the web server that keeps a document specified with the URL obtained from URL Frontier. It is implemented with Crawler class (Appendix C).

Package Crawl Class Crawler:

Identification	:	Crawler
Type	:	Class
Superclass	:	Object Class
Purpose	:	This class interacts with web.
Function	:	Sends HTTP request for the URL sent from URL Frontier and accepts URL response for it.
Subordinates	:	public Crawler() public void runCrawl (String crawlTerm) private boolean isAvailable(String URLController)
Dependencies	:	Utility.ConceptInfo Utility.LinkInfo

		Utility.CalculateComparator
		URLFrontier
		URLHistory
Resources	:	java.util
		java.io
		Utility
		UMLSKSSConnector
		java.net
		Document_Parser

Fetcher sends a request to URLFrontier to obtain the URL with highest priority to the crawl topic. When the URL to bi visited is acquired, it generates an HTTP request and sends it. Method “isAvailable” handles if the URL is alive and available to download. The HTTP response is accepted as an input stream. The stream is directed to Document Parser module as it is transmitted through internet.

3.4.3.4 URL History

URL History is a lookup hash table kept to check if the URL has been visited before or not. It keeps a record for every visited URL in order to stop system visit same URLs again. It is implemented with URLHistory class.

Package Crawl Class URLHistory:

Identification	:	URLHistory
Type	:	Class
Superclass	:	Object Class
Purpose	:	This class is designed to keep record of visited links.
Function	:	Keeps a log for any link visited by the system.
Subordinates	:	public void initializeHistory()
		public void saveHistory()
		public boolean linkVisitedBefore(String URL)

		public void addLinkURLHistoryHashTable(String URL)
		public void showHistory()
Dependencies	:	None
Resources	:	java.util
		java.io

3.4.3.5 URL Frontier

URL Frontier is the core of any crawler system that defines its characteristics and purpose. URL Frontier for Lokman crawler system is designed as a maximum (max) heap priority queue that places the highest value URL according to a search topic at the top of the queue. Link Estimator module deals with relevance calculations. URL Frontier involves in ordering URLs according to values, sending the leading URL to Fetcher, updating values for existing but not yet visited URLs and reordering the queue after each insertion or deletion.

URL Frontier for Lokman crawler system is implemented with URLFrontier class.

Package Crawl Class URLFrontier:

Identification	:	URLFrontier
Type	:	Class
Superclass	:	Object Class
Purpose	:	This class serves as the URL Frontier for the system.
Function	:	Keeps a queue for URLs to be visited and performs the ordering/reordering operations on this queue.
Subordinates	:	public URLFrontier()
		public void createNewURLElement(String newLink, int parentValue)
		public void createNewURLElement(LinkInfo newLinkInfo, int parentValue)

		public int existsInFrontier(LinkInfo newElement) private void pushFrontier(LinkInfo newElement) private int removeURLElementFrontier(LinkInfo URLToRemove, int indexOfElement) public LinkInfo popFrontier() public boolean isEmpty() public boolean isFull()
Dependencies	:	Utility.LinkInfo
Resources	:	Utility

When an input is obtained from the Link Estimator, URL Frontier first checks if this URL exists in the URL list. If it does, the URL Frontier removes the URL from the list and invokes the reEvaluateURLValue function of Utility.LinkInfo. Then treating the URL as it never appeared in the queue before, URL Frontier inserts new URL to the queue, reordering the URLs. The URL at the top of the priority queue is always resembles the web document to be retrieved by the fetcher at next crawl loop.

3.4.3.6 Document Parser

Document Parser module gets the fetched web document from Fetcher Module as a stream. Its main task is parsing the stream (document) and extracting hyperlinks to be estimated further.

Document Parser is implemented with GeneralLinkExtractor (Appendix I) and HtmlStreamParser classes (Appendix L). In addition, this module is designed to support processing documents saved to local discs considering possible further developments in overall system structure (Appendix K HtmlFileParser Class).

GeneralLinkExtractor class is extended from LinkExtractor abstract class (Appendix H). File parser classes are extended from FileDocument abstract class (Appendix J).

Package Document_Parser Class FileDocument:

Identification	:	FileDocument
Type	:	Abstract Class
Superclass	:	Object Class
Purpose	:	This class defines the basic properties for a file/stream.
Function	:	Defines the basic properties for a file/stream.
Subordinates	:	public FileDocument(File file, boolean stem) public FileDocument(InputStream HTMLStream, boolean stem) public int numberOfTerms() public String nextTerm() public boolean hasMoreTerms() public void prepareNextTerm () protected static void loadStopWords()
Dependencies	:	None
Resources	:	java.util java.io

Package Document_Parser Class HtmlStreamParser:

Identification	:	HtmlStreamParser
Type	:	Class
Superclass	:	FileDocument Class
Purpose	:	This class defines parsing operations for an HTML stream.
Function	:	Parses an HTML stream.
Subordinates	:	public HtmlStreamParser(InputStream HTMLStream, boolean stem) protected String getNextCandidateTerm() public String getLine()
Dependencies	:	Crawl.Crawler

Resources	:	java.util java.io java.net Utility

Package Document_Parser Class LinkExtractor:

Identification	:	LinkExtractor
Type	:	Abstract Class
Superclass	:	Object Class
Purpose	:	This class defines basic link extraction operations for any HTML document/stream.
Function	:	Defines basic link extraction operations for any HTML document/stream.
Subordinates	:	public LinkExtractor(File sourceFile) public LinkExtractor(InputStream HTMLStream) public LinkExtractor (BufferedReader HTMLStreamReader) public abstract ArrayList extractLink() public String getLine() public void printLinks() public boolean checkLine()
Dependencies	:	None
Resources	:	java.util java.io

Package Document_Parser Class GeneralLinkExtractor:

Identification	:	GeneralLinkExtractor
Type	:	Class
Superclass	:	LinkExtractor Class
Purpose	:	This class defines link extraction operations for any HTML stream.

Function	:	Extracts links from a given HTML stream.
Subordinates	:	public GeneralLinkExtractor(InputStream HTMLStream)
		public GeneralLinkExtractor (BufferedReader HTMLStreamReader)
		public ArrayList extractLink()
		private String findLink()
		private String prepareLink(String link)
		private String findLinkInfo()
Dependencies	:	Utility
Resources	:	java.util
		java.io
		Utility

3.4.3.7 Link Estimator

Link Estimator Module is the module that assigns a weight factor for each hyperlink h in the hyperlink set H obtained from Document Parser module. To make a better estimation, it is enhanced with a term list from XML Parser module. It is implemented with CalculateComparator class (Appendix R).

Package Utility Class CalculateComparator

Identification	:	CalculateComparator
Type	:	Class
Superclass	:	Object Class
Purpose	:	This class assigns a value for a given topic to downloaded web documents and hyperlinks within.
Function	:	Sets weight values using information obtained from UMLSKSS to each hyperlink.
Subordinates	:	public CalculateComparator(String linkDesc, ConceptInfo[] termlist)
		public double evaluate()

Dependencies	:	Utility.LinkInfo
		Crawl.URLFrontier
		Crawl.URLHistory
		Crawl.Crawler
Resources	:	java.util
		java.io
		Utility

First task of this module is deciding if the hyperlink is visited before or not. This is a simple binary search operation performed on the URL History. If it is not visited before, it performs the below tasks.

The terms in the term list acquired from XML Parser have different ranks according to their exact match, synonymy, relations and contextual relation. The value assigned to a link is a function of several values details of which are described below:

A visited page P which contains the link L, to which the system assigns a value, is labeled as Father Page for the link L. An earlier visited page GP which contains the link through the system found and visited page P is labeled as Grandfather Page for the link L. Link Estimator uses the inherited values (quality of pages visited until this link is found) as well as the information obtained from the link itself.

CalculateComparator counts the number of occurrences of each Concept Set element within the text (Page or Link Information). According to the semantic relation to the crawl topic each concept type has a different weight.

The weight factors are given experimentally as 15 for exact match (Concept Relation Type 1), 12 for synonymy (Concept Relation Type 2), 8 for partial relevance (Concept Relation Type 3) and 5 for contextual relevance (Concept Relation Type 4). Occurrence count of each element in the Concept Set is multiplied with the corresponding weight value and finally these values are

summed to indicate the relevance level of the given text. To avoid the processing overhead caused by floating point operations, normalization is not performed.

To reflect the effect of referencing page's weight on page and link value calculation, a 0,1- 0,9 factor scheme is used. These values are depicted after a series of tests on crawler's performance. During these tests, it was found that decreasing the ascendant pages' effect proves better results on crawler performance. Beginning from 0,4, crawler performance was observed with 0,05 decrements.

Page value is computed as follows:

Page Value (PV) = Father Value * 0.1 + CalculateComparator (Page) * 0.9

If CalculateComparator (Page) equals 0, this indicates that even the information obtained from the link through which system visited this page reflected lexical similarity to the crawl topic, the page bore no similarity. In such a condition, system flags the links out from the page as irrelevant and stops processing them. Else each link in the page is processed and assigned a value. Any link's value appearing in the page is computed as follows:

Link Value = Father Value * 0.1 + CalculateComparator (Link) * 0.9

Or;

Link Value = (Father Value * 0.1 + CalculateComparator (Page) * 0.9)*0.1
CalculateComparator (Link) * 0.9

Therefore;

Link Value = CalculateComparator (GP) * 0.01 + CalculateComparator (P) *
0.09 + CalculateComparator (L) * 0.90

After value assignment is completed for each link, they are sent to URLFrontier and inserted to the queue.

CHAPTER 4

Discussion and Evaluation of Lokman Crawler System

Chapter 4 covers a detailed discussion of the system focusing on the algorithm performed by the Link Estimator module. Following the discussion, metrics, which are used in crawler evaluation, are introduced. Empirical results and evaluation of the Lokman Crawler System is presented finally.

4.1 Discussion of Lokman Crawler System

Crawler systems traverse the hyperspace using the same strategy. They find out the links out from a visited page and follow them to visit other pages. This is the simplest way of traversing the web but also the most brute approach. It is obvious that the fraction of pages which truly address an information need is a small portion of documents available on the web. Therefore, some algorithm inserted in the crawler which makes the whole system pick and visit URLs of interest before others make crawlers different from each other. URL ordering (selection) algorithms are based on some real world entities like reference count or lexical similarity.

Algorithms like PageRank evaluate URLs according to the number of inward links (links referencing to the URL from other URLs) or outward links (links out from the URL referencing other URLs). Such approaches decide the value of a web page as the possibility of a random internet surfer's visiting the given URL, starting the surf from any random point of the graph.

Other algorithms try to evaluate an URL according to its lexical analysis, presenting its resemblance of the crawl topic. Lexical analysis for a page may be quite simple or quite complex according to implementation. Deciding the complexity of the analysis is affected by available bandwidth, software and hardware issues. Simply such algorithms try to estimate a page's "lexical distance" to the "needed information".

Since the main target of MedicoPort project is finding out the advantages of ontology use in information retrieval systems, URL ordering algorithm selected for Lokman Crawler is a lexical based one, Best First Search (Section 2.4.1).

Best First Search algorithm provides a quite simple but also quite effective way of graph traversal. Very briefly, the algorithm executed is "go to the most relevant node and keep acting so". But the problem of finding the most relevant node still remains to be solved.

Link evaluation algorithm executed by Lokman Crawler system addresses and solves this problem with ontology support. Unlike standard crawlers implementing any algorithms introduced in Section 2.2, Lokman has a greater knowledge of the problem context than the simple crawl topic before starting the crawl task (Table 3). This information is obtained from UMLSKSS.

Lokman begins the crawl task by visiting the seed URL. Selecting a seed URL is a major fact that affects the overall performance of the system. Since it is the first page and the starting point for a long journey through many web pages, it must include plenty of references. An important issue to be emphasized for Lokman is that: System also uses a dynamic page to determine a set of relevant URLs. Namely, when a runCrawl() command is invoked, system runs the crawl topic on a search engine and gets a limited set of hyperlinks addressing related URLs. Using the major seed URL and 15 other URLs obtained from the search, Lokman begins crawling.

Table 3: Comparison of Lokman Crawler's and A Standard Crawler's Knowledge Base without Ontology Support with Crawl Topic = "Bedwetting"

	Term 1	Term 2	Term 3	Term 4
Standard Crawler	Bedwetting	-	-	-
Lokman Crawler	Bedwetting	Nocturnal Enuresis	Nocturnal incontinence of urine	Wets bed

Since seed URL is the only element in the URLFrontier, regardless of the crawl topic, it is the first document to download. While picking the URL from the URLFrontier, system also gets the value of the URL. This value is labeled as "inherited value". After download, system evaluates the value of the URL according to the given topic and its ontological related term set. This value is labeled as "page value" (See Section 3.4.3.7 for details).

Page value for a page P is updated as a function of inherited value and page's own value. This operation increases or decreases the value of the page P according to the information gained from the previous page Q, through which crawler visited the page P. Such an evaluation for a page's value also makes the system behave not only lexically but also forward/backward link count based. An important point to be emphasized is system still assigns the prime factor to the page's lexical value. While the page's value keeps growing up and up in a forward/backward link count based algorithm as long as some other pages have references to it, Lokman Crawler System keeps the positive effect in a limit.

After computing the value of the page, the links within it are extracted. This set of links form the next possible stops for the crawler. The links point to non-visited pages but they reflect and carry some information about the page they are referencing. Therefore, the hyperlink H after "href" tag and also the visible text for H between <a…> tags are processed in same manner with the page evaluation. At the end of the process, each link in the set is assigned a value resembling its relevance degree to the crawl topic labeled as "link value". Link

value is updated as a function of Link Value and Page value. This way, quality of the ascendants of the link is represented in the link value. Finally each link is added to the URLFrontier with its link value.

URLFrontier for a best first crawler is implemented as a priority queue. Priority queue adds elements according to their comparator, in crawler's case, link value. After each deletion or insertion, the URL with maximum link value should be on the top of the queue. Priority queue can be implemented in several ways (Table 1) among which maximum/minimum heap representation provides the best algorithmic computational cost and complexity.

Another important issue to decide is the algorithm to reevaluate an URL which has not been visited yet and encountered more than once. Suppose that the URL (http://www.metu.edu.tr) indicating the home page of METU is referenced by page P and page P is a downloaded page. After computations described above, the URL referencing METU home page is assigned the link value of 5 and inserted into the queue. Crawler picked the top URL from the frontier and downloaded it. R, too, included a link to METU home page and this time link value for same URL is computed as 12. Same URL, which already existed in the queue, is sent to URLFrontier to be inserted. At this point a reevaluation function should be implemented to solve the above problem.

In this thesis study, two algorithms are tested to perform the reevaluation function. First algorithm updates the new URL value as the sum of old and new values of the link, therefore increases the value of http://www.metu.edu.tr to 17 for the given example. This algorithm is named as IncrementValues algorithm. Second algorithm updates the new URL value as the greater of both computed values, updating the value of http://www.metu.edu.tr to 12 for the given example. This algorithm is called as GetGreater algorithm.

When crawler finishes the procedure for an URL, it gets back and picks the topmost URL from the URLFrontier until URLFrontier is empty.

Table 4: Explanation of Handled Exceptions by Lokman Crawler System

Exception	Explanation
MalformedURLException	The URL address is not in true format.
NoProtocolException	The URL address contains no communication protocol.
NoRouteToHostException	No route to host is found, no corresponding IP found in routers' routing table, connection timed out.
ConnectException	The URL does not address a valid document, document does not exist.
UnknownHostException	The URL does not exist; domain does not exist or cannot be reached.
IOException	HTTP connect request rejected.

To perform a successful crawl, a crawler must be implemented so as to handle any possible internet sourced failure. Design should not allow termination of crawl loop due to any exceptions. Standard HTTP errors are not designed for applications other than web browsers. Exceptions in Table 4, which are generated due to HTTP errors, are thrown by JAVA. These exceptions are handled by isAvailable() method.

4.2 Crawler Evaluation Metrics

4.2.1 Harvest Rate Metric

Harvest rate (Acquisition Ratio) is the ratio of number of important pages (P_R) to the number of all crawled pages (P_A) (Equation 5) [41]. A topical crawler's target is visiting relevant URLs to the given topic before irrelevant URLs.

Therefore it requires an algorithm to prioritize the URLs in its frontier. Harvest rate is an indicator of the success or failure of a crawler's URL prioritization algorithm.

$$\text{Harvest Rate} = \frac{\#P_R}{\#P_A} \qquad \text{Equation 5}$$

To label a visited page as important can be performed using different techniques according to implementation. It can be performed by computer using a threshold lexical similarity, occurrence of some specific phrases or all pages can be inspected manually. In this study's testing case, pages with a page value higher than 0 are regarded as important pages (See 3.4.3.7 for page value computation details).

4.2.2 Target Recall

Well-known URLs are split into two disjoint sets labeled seeds (P_S) and targets (P_T). Crawler starts with the URLs in seed set and recall of the target set is measured (Equation 6, P_C stands for pages crawled) [14, 42].

$$\text{Target Recall} = \frac{|\mathbf{Pt} \cap \mathbf{Pc}|}{|\mathbf{Pt}|} \qquad \text{Equation 6}$$

Target recall metric tries to estimate the overall recall performance of a crawler. It is obvious that it is impossible to determine the number and location of all relevant URLs for a specific topic. Therefore, a selected set of relevant URLs are used for this estimation called the target URL set. In addition it is impossible to determine all pages with a reference to a relevant page. To model such URLs, another set of URLs are selected and named as the seed URL set. Capability of a

crawler to visit target set by starting the crawl from members of the seed set is called as target recall (Figure 14) [14].

Using target recall values for different crawl topics, therefore different target and seed URL sets, estimation on crawler's recall rate can be made.

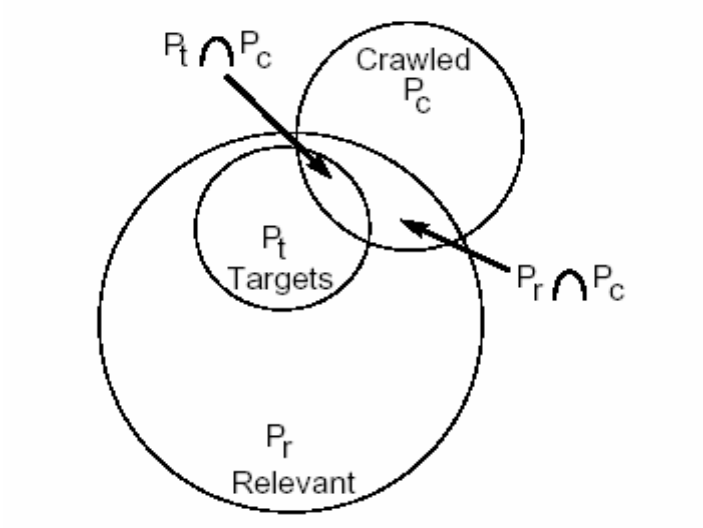


Figure 14: The performance metric |Pt∩Pc|/|Pt| as an estimate of |Pr∩Pc|/|Pr| [14]

4.3 Evaluation of Lokman Crawler System

4.3.1 Test Bed

Tests for Lokman Crawler System are performed on a Pentium 4 2.8 GHz Processor PC with 768 MB of RAM. 10 Topics are selected for test randomly from MedLine Plus web page [43] (Table 5).

Table 5: Crawl Topics for Crawler Test

Topic#	Search Topic	Topic #	Search Topic
1	Chickenpox	6	Vasectomy
2	Bedwetting	7	Tinnitus
3	Breast Carcinoma	8	Sunburn
4	Deafness	9	Motion Sickness
5	Influenza	10	Insomnia

4.3.2 Empirical Results

Empirical results for the system reflect the average results after 5 crawls for each topic given in Table 5. All results for harvest rate are presented for crawler's performance including the direct links out from seed URL (D = 1 links) and excluding such links. It is clear that D = 1 links refer to relevant pages for a given topic but content of the links afterwards (D = 2 and further) are not that predictable. Since Lokman is a topical crawler, its behavior further than D = 1 is quite important resembling its URL prioritization power.

Results begin with evaluation of two algorithms to reevaluate URLs, which are introduced in Section 4.1, IncrementValues algorithm (Algorithm I in graphs) and GetGreater (Algorithm II in graphs) algorithm. First tests were performed to find out the one which improved Lokman Crawler's performance. Lokman performed 5 crawls for each topic given in Table 5 for both algorithms. Metric selected to make the decision is Harvest Rate. Test is based on algorithm's performance after visiting first 100 URLs.

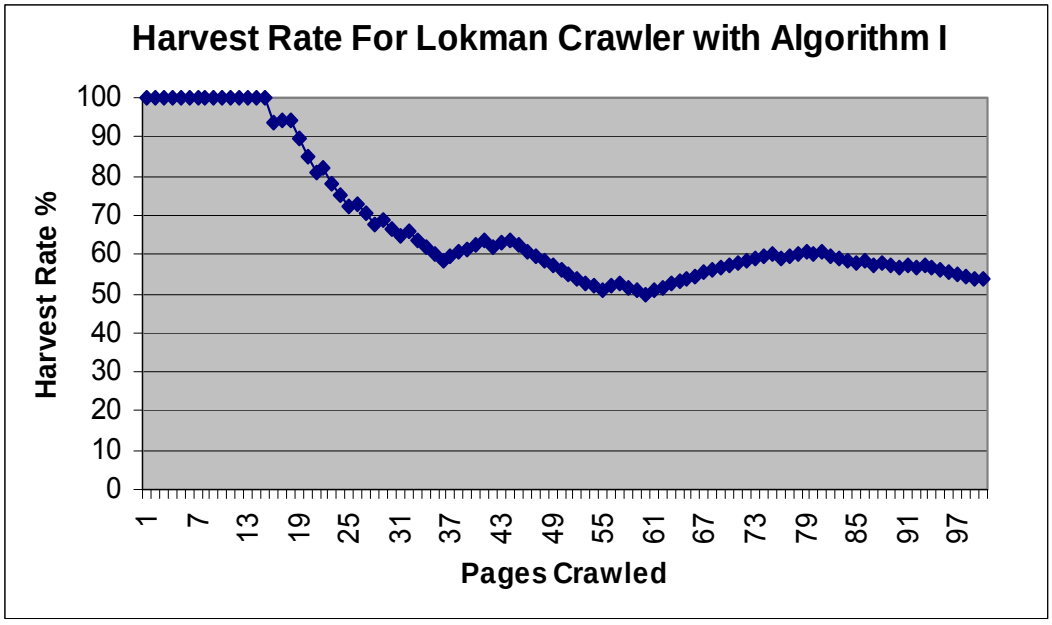


Figure 15: Average Harvest Rate for Lokman Crawler using IncrementValues Including Direct Links out of Seed URL set.

Figure 15 reflects the results of tests performed using IncrementValues algorithm. First 15 of the pages are the direct links found out of the seed URL. This test revealed that IncrementValues has an average harvest rate of 67,095% with topmost 15 hits (D = 1 pages, pages which are directly referenced by the seed URL). In Figure 16, harvest rate for same algorithm excluding topmost 15 hits are presented. Without these URLs, IncrementValues has an average harvest rate of 41,234%.

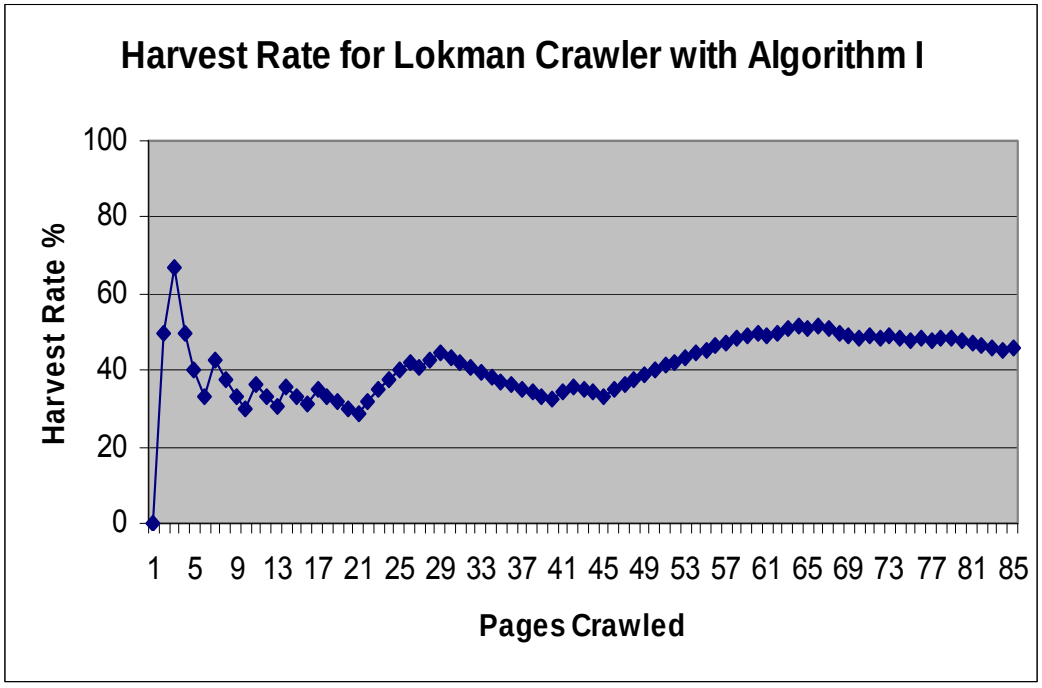


Figure 16: Average Harvest Rate for Lokman Crawler using IncrementValues Excluding Direct Links out of Seed URL set.

GetGreater outperformed IncrementValues by means of harvest rate. GetGreater reached an average harvest rate of 76,9057% with topmost 15 hits (Figure 17) and an average harvest rate of 67,2906% without them (Figure 18).

Since GetGreater increased the harvest rate of Lokman by 9,811% for first condition and by 26,0562% for second condition, reevaluation of URLs for the crawler system is performed with this algorithm. In other words, GetGreater algorithm improves the probability of Lokman's directing itself to more relevant

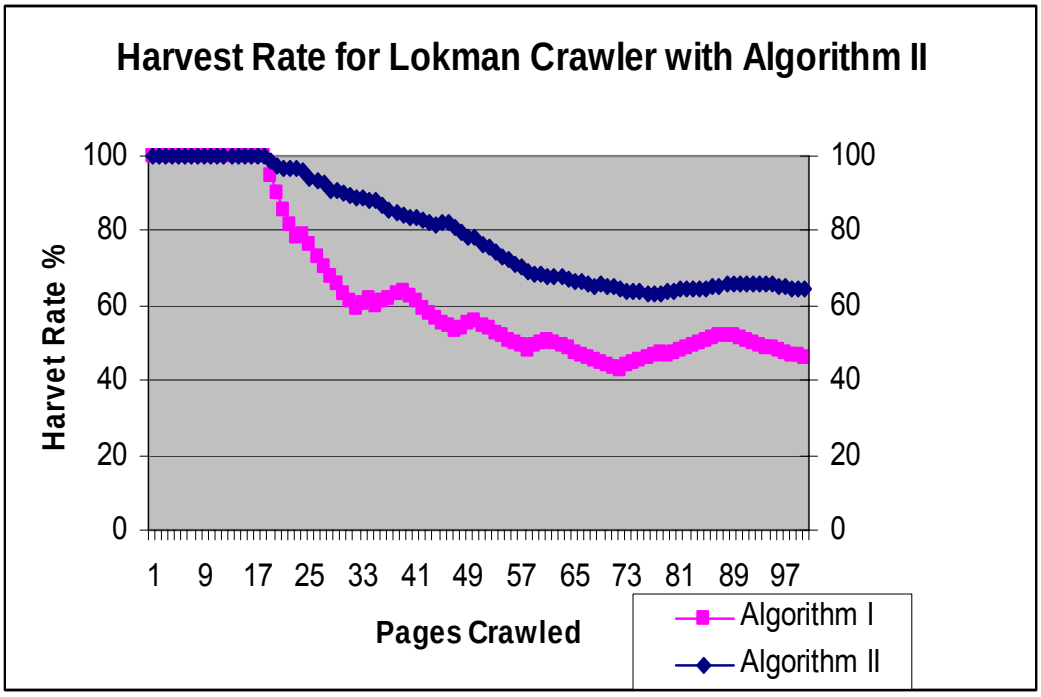


Figure 17: Average Harvest Rate for Lokman Crawler using GetGreater Including Direct Links out of Seed URL set.

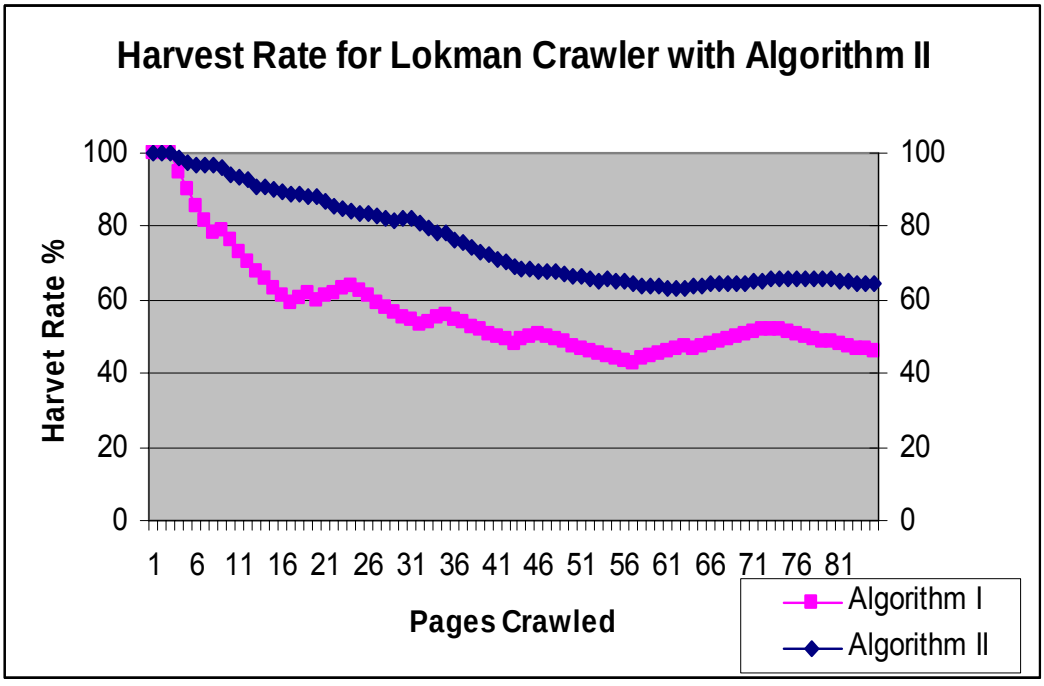


Figure 18: Average Harvest Rate for Lokman Crawler using GetGreater Excluding Direct Links out of Seed URL set.

pages by 9,811% if D = 1 pages are included in test results. Excluding D = 1 pages, a more dramatic improvement of 26,0562% is achieved by the crawler. Therefore, Figure 17 also resembles the average harvest rate for Lokman Crawler System.

To illustrate the efficiency of GetGreater algorithm, screenshots for 3 different URLs provided encountered for a crawl with "Breast Cancer" topic. Figure 19 is from http://patients.uptodate.com/topic.asp?file=cancer/5162, 17th visited page during crawl. Figure 20 is from http://www.cnn.com/HEALTH/library/HQ/00348.html, 57th visited page. Figure 21 is from http://medlineplus.nlm.nih.gov/medlineplus/breastcancer.html, 97th visited page.

Figure 19: "http://patients.uptodate.com/topic.asp?file=cancer/5162", 17th Visited Page by Lokman Crawler with topic "Breast Cancer" using GetGreater Algorithm

Figure 20: "http://www.cnn.com/HEALTH/library/HQ/00348.html", 57th Visited Page by Lokman Crawler with topic "Breast Cancer" using GetGreater Algorithm

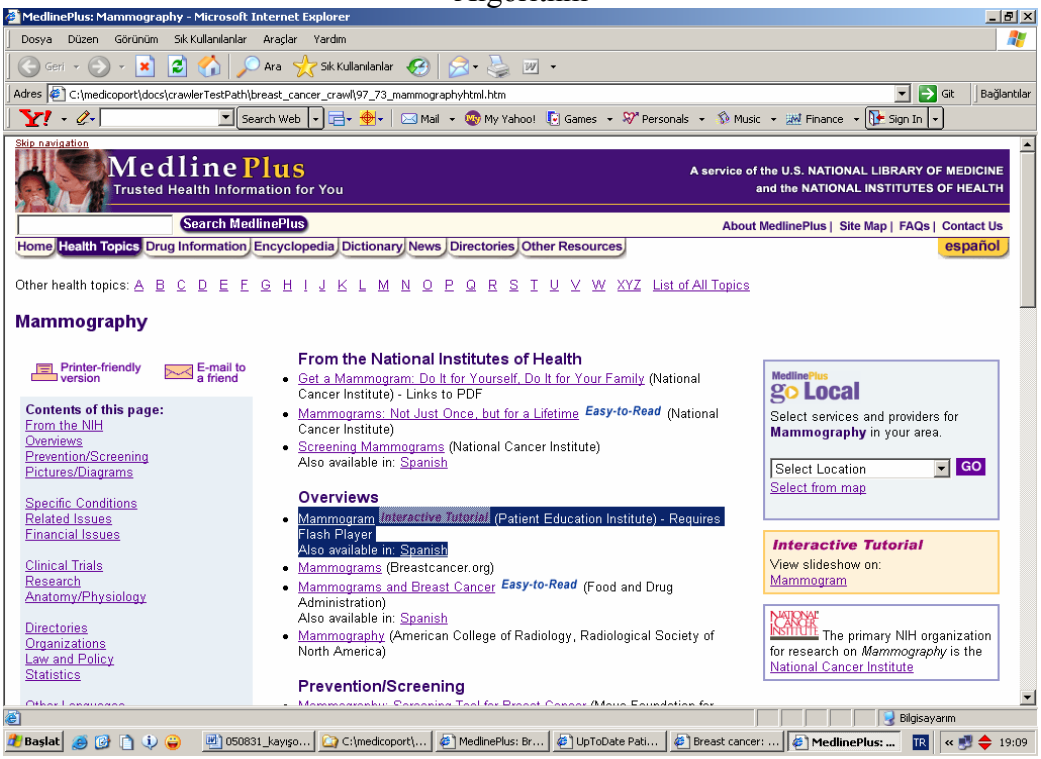


Figure 21: "http://medlineplus.nlm.nih.gov/medlineplus/breastcancer.html", 97th Visited Page by Lokman Crawler with topic "Breast Cancer" using GetGreater Algorithm

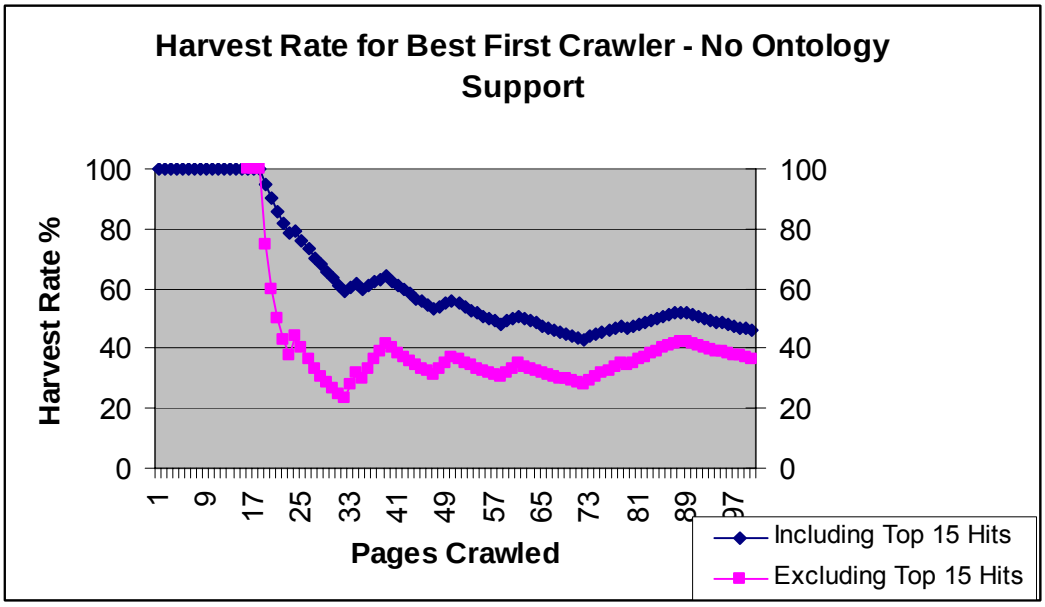


Figure 22: Harvest Rate for a Simple Best Search Crawler

To illustrate the positive effect of ontology use in crawling, an ordinary best first crawler with no ontology support is implemented as well. Harvest rates for same conditions are measured. This crawler scored a harvest rate of 63,3287% for first condition and 38,0434% for second (Figure 22).

Using the above values of the ontology based crawler and the crawler with no ontology support, it is proved that ontology increases the harvest rate of the crawler by 13,577% for first condition and by 29,2472% for second condition.

To compute the target recall value, 50 pages are selected randomly from different search engines for each topic. These pages are split to two sets, 10 for seed set and 40 for target set. Seed set and target set for "Bedwetting" is provided in Appendix S.

Target recall value increases proportionally to the number of pages crawled. During tests, crawls with length = 800 are performed and the crawler reached an average recall value of 21,25% at 800 pages (Figure 23). Case studies performed in [14] and [29] reveal non-ontology supported crawlers' target recall

performances. Shark Search crawler reaches a target recall value of 12% at same level. A best first crawler's performance varies from 9% to 30%. An InfoSpiders crawler reaches a target recall level close to 20%. Using this information, it can be claimed that ontology support increases the recall value considerably in comparison to crawlers with no ontology support.

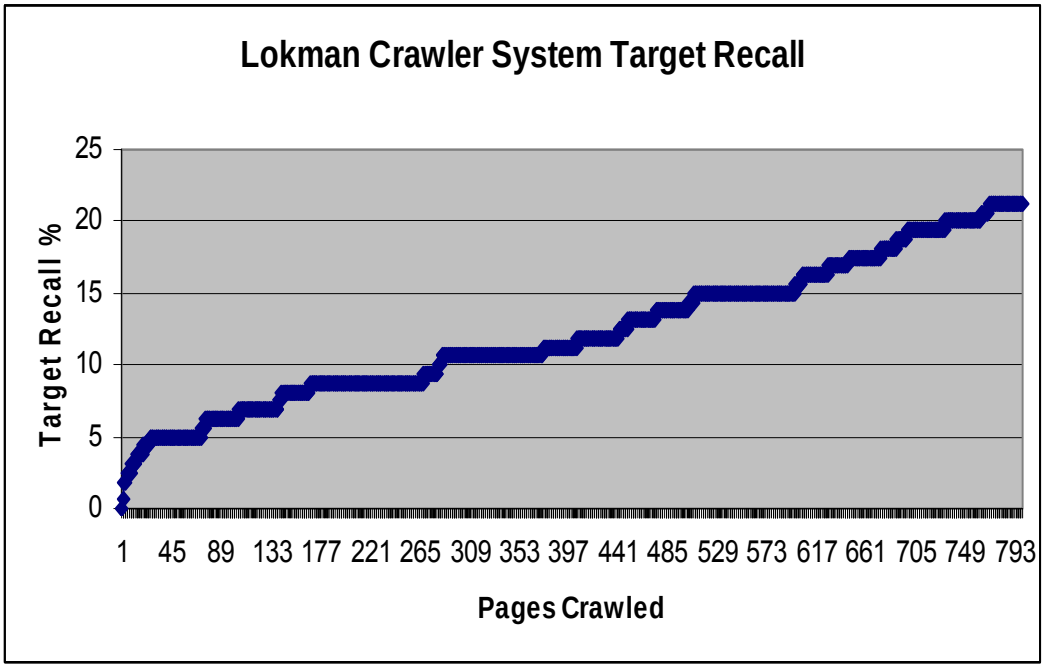


Figure 23: Target Recall Graph for Lokman Crawler System at Page Number = 800

CHAPTER 5

Conclusion

This chapter presents the concluding remarks of the study, future work about the system and contribution of the thesis study to MedicoPort Project.

5.1 Conclusion

As the information sharing medium of information age, web has a very dynamic nature. Some information on the web change, some disappear and some appear continuously. Dynamism of the web is accelerated by increasing number of users making the web the greatest structure humankind ever constructed.

On the other hand, it is obvious that handling the information on web is impossible without computer support. No matter what the focus of a search performed on the web is; reliable software is required to find out and filter excessive amount of information.

Crawler systems form the backbone of any web information retrieval systems. In this thesis study, use of ontological information as the knowledge base for a topical crawler, Lokman, is discussed. Lokman is designed to serve MedicoPort web search engine therefore, medicine is selected as domain and UMLS ontology is used as the domain knowledge base.

Since ontology provides larger information on the topic to be crawled, Lokman has the potential to inspect the downloaded pages and hyperlinks within in a more effective way discussed in previous chapters. While a simple topical

crawler performs crawling tasks with a phrase or a word, Lokman, as ontology based crawler, has information about the term, the synonyms and other terms from same context.

Such contextual information helps to prioritize URLs and visit the ones resembling more similarity to the topic before the ones with less similarity. In this thesis, two different versions of URL prioritization algorithms are discussed and evaluated. Although GetGreater algorithm is chosen to implement the system, it is proved that even worse performancing IncrementValues algorithm outperforms a standard crawling algorithm when ontology support is provided.

Ontology support is largely used during downloaded page analysis. It can be claimed that analyzing the pages and hyperlinks with a larger set of information makes the crawler system more robust against the hazards of natural language ambiguities. Such robustness allows crawler visit relevant pages and find out relevant hyperlinks faster.

Lokman Crawler System is an example of ontology enhanced web crawler. Enhancing a topical crawler's performance with domain knowledge representation is an effective way to improve the document collection's quality unless it is a general repository. Lokman's test results reveal that ontology increases the harvest rate for a topical crawler by 13,577% if D (depth) = 1 pages (pages which are directly referenced by the seed URL) are included to the visited pages and by 29,2472% if D=1 pages are excluded. Harvest rate values acquired by Lokman are 76,9057% with D = 1 pages and 67,2906% without them. Since seed URL selection is a very important factor affecting the overall performance of crawler systems, second harvest rate value is more significant.

This study also revealed that ontology support increases the performance of a topical crawler by 76,87% raising it from 38,04% to 67,29%. This significant conclusion proves that an ontology based crawler finds out 76,87% more relevant pages than a simple crawler when same number of pages are crawled.

Generic search systems addressing classic user behavior are getting away from satisfying user needs as information published on web grows. Therefore, more sophisticated, more focused and more personalized systems are required to answer the user needs. A sophisticated system needs a better understanding of the human need. Therefore, it requires more information than a simple search phrase. The better is the information granted by human to machine, it is obvious that the higher the quality of retrieval results. Concept of ontology, in this manner, provides a good framework for information to be fed to computers. Future's ontology based information retrieval systems should rely on ontology based crawlers like Lokman promising better performance consuming less resources.

5.2 Future Work

Although Lokman Crawler System forms a good example of embedding ontology in crawling process, there is still much to do to increase the performance of the system. As emphasized in previous section, ontology information is heavily used in page and link evaluation.

In this study, it is assumed that all HTML tags are of same importance except <a href> tags. Rest of the document excluding the lines between <a href> tags are treated the same. Assigning different weight values to other HTML tags appearing in the page has the potential to increase link and page quality. These tags are namely, bold tag (), header tags (<h1></h1>, (<h2></h2>, etc.) and font tags.

Another issue remains to be tested and discussed is, how crawler's performance would change if different weight factors were used for information obtained from a hyperlink and information carried by the page except hyperlinks while a page's value is computed. In other words, how would crawler behavior change needs to be tested if different weight schemes are used in computing hyperlink value as an information indicator and page's value as an information resource.

Although Lokman is implemented as a subsystem of MedicoPort project, its infrastructure also supports crawling as an independent search agent harvesting documents of interest from web as well. It also supports implementation of a stand alone application accepting input as search terms and weight factors from user instead of ontology.

Lokman is designed to run with UMLS ontology but it is capable of performing with any ontology provided. To run the system with other ontology, required modifications on XML Parser and UMLS Connector modules should be performed. Modifications include the interfaces used in connection and semantic relations within the ontology used in parsing. Such modular structure of the design allows the system act as an independent agent seeking information in the web for any purpose. In addition, considering the similarities between topical crawling on web and crawling on enterprise local area networks (LAN), it has the potential to serve for any LAN search engine with small modifications.

5.3 Contribution

Lokman Crawler is implemented as the crawler subsystem of MedicoPort Project. As part of a search engine, a crawler's function is increasing the quality of documents retrieved from web.

Search Engine performance is directly affected by the capability of its crawler. An index of documents with high quality means a better set of answers presented to the user. Therefore, especially for a domain specific search engine like MedicoPort, quality of pages is more important than the quantity. Lokman's effective URL ordering (page selection) algorithm and detailed page analysis algorithm addresses this fact.

Analyzing a downloaded document, even it is processed in RAM as an input stream, is a time consuming task. Lokman performs a crawl of 100 pages at an average time of 588,215 seconds. Considering that MedicoPort serves for

medical domain and promises to retrieve a set of answers with high relevance degree to given queries, Lokman has to download the whole HTML document instead of first n bytes of it from any source before processing. Main reason for longer processing time is this fact.

On the other hand, document collection formed by Lokman crawler is of pretty high quality. As test results indicated, Lokman reaches a harvest rate of 67,2906% excluding D = 1 pages thanks to the information obtained from UMLSKSS. With such a coherent document collection, Lokman eases the workload of Indexing subsystem, keeping the index terms (the ones other than UMLS SPECIALIST Lexicon terms) in a limit [3].

MedicoPort Project aimed to find out advantages of ontology use in information retrieval and prove the positive affect on overall performance of the system from crawling to indexing. This thesis study proved that ontology use almost doubled the crawler performance using the algorithm details of which are presented in section 3.4.3. Lokman Crawler System provided a framework for ontology based crawling strategy. The author of this thesis hopes that future studies on document processing issues will make the system more intensive about the content and faster to process.

REFERENCES

[1] Google Inc., Retrieved August 30, 2005, from http://www.google.com.

[2] Arasu, A., Cho, J., Garcia-Molina, H., Paepcke, A., Raghavan, S., Searching the Web, ACM Transactions on Internet Technology (TOIT), Volume 1, No. 1, August 2001.

[3] Kubilay, M., Special Index and Retrieval Mechanism for Ontology Based Medical Domain Search Engines, M.S. Thesis, METU, September 2005, Ankara, Türkiye.

[4] Lenci, A., Building an Ontology for the Lexicon: Semantic Types and Word Meaning, in: P.A. Jensen, P. Skadhauge (Eds.), Ontology-based Interpretation of Noun Phrases, Proc. of the 1st International OntoQuery Workshop, Kolding, 2001.

[5] Ding, Y., Ontology: The Enabler for the Semantic Web, Journal of Information Science, Volume 27 (6),, 2001.

[6] Kalem, G., Semantic Web Application: Ontology-Driven Recipe Querying, Atılım University, M.S. Thesis, June 2005, Ankara, Türkiye.

[7] Web Ontology Working Group (WebOnt), World Wide Web Consortium (W3C), Retrieved August 23, 2005, from http://www.w3.org/2001/sw/WebOnt.

[8] UMLS Introduction, National Library of Medicine, Retrieved August 21, 2005, from http://www.nlm.nih.gov/research/umls/umlsdoc_intro.html.

[9] UMLS Documentation, Unified Medical Language System (UMLS), Retrieved August 21, 2005, from http://www.nlm.nih.gov/research/umls/umlsdoc_intro.html#s1_0.

[10] Specialist Lexical Tools, Unified Medical Language System (UMLS), Retrieved August 16, 2005, from http://specialist.nlm.nih.gov/LexTools.html.

[11] UMLS Metathesaurus Introduction, Unified Medical Language System (UMLS), Retrieved July, 21, 2005, from http://www.nlm.nih.gov/research/umls/meta3.html#s3_0.

[12] UMLS Metathesaurus, Unified Medical Language System (UMLS), Retrieved August 20, 2005, from http://www.nlm.nih.gov/research/umls/meta2.html#s2_0.

[13] Dunne, D. (2001, April), What is a Bot?, Retrieved August 20, 2005, from http://www.darwinmag.com/learn/curve/column.html?ArticleID=95.

[14] Pant, G., Srinivasan, P., Menczer, F., Crawling the Web, in Web Dynamics Adapting to Change in Content, Size, Topology and Use, Levene, M.; Poulovassilis, A. (Eds.), Springer-Verlag, 2004.

[15] Cheong, F.C., Internet Agents: Spiders, Wanderers, Brokers, and Bots, New Riders Publishing, USA, 1996.

[16] Chau, M., Chen, H., Personalized and Focused Web Spiders, in Web Intelligence, Zhong N. et al. (eds), pp 197-216, Springer-Verlag, Berlin, 2003.

[17] Brin, S., Page, L., The Anatomy of a Large-Scale Hypertextual Search Engine, Proc. the 7th International World Wide Web Conference, Australia, 1998.

[18] Heydon, A., Najork, M., Mercator: A Scaleable, Extensible Web Crawler, World Wide Web Conference, 2(4) 219-229, April 1999.

[19] Koster, M., The Web Robots Pages, Retrieved December 13, 2004, from http://info.webcrawler.com/mak/projects/robots/robots.html.

[20] Baeza-Yates, R., Riberio-Neto, B., Modern Information Retrieval, Addison-Wesley, 1999.

[21] Brewington, B.E., Cybenko, G. (1999), How Dynamic is the Web?, Retrieved: August 27, 2005, from http://www9.org/w9cdrom/264/264.html.

[22] Lawrence, S., Giles, C.L., Accessibility of Information on the Web, Nature, 1999.

[23] Cho, J., Garcia-Molina, H., Estimating the Frequency of Change, ACM Transactions on Internet Technology (TOIT), Volume 3, Issue 3, August 2003.

[24] Cyveillance, Sizing the Internet, Retrieved September 1, 2004, from http://www.cyveillance.com.

[25] Breadth-first Search National Institute of Standards and Technology, Retrieved August 26, 2005, from http://www.nist.gov/dads/HTML/breadthfirst.html.

[26] Breadth-first Search, Brainy Encyclopedia, Retrieved August 15, 2005, from http://www.brainyencyclopedia.com/encyclopedia/b/br/breadth_first_search.html.

[27] Design and Analysis of Algorithms Lecture Notes, Retrieved April 11, 2005, from http://www.ics.uci.edu/~eppstein/ 161/960215.html.

[28] Horowitz, E., Sahni, S., Anderson-Freed, S., Fundamentals of Data Structures in C, Computer Science Press, 1997.

[29] Menczer, F., Pant, G., Srinivasan, P., Topical Web Crawlers: Evaluating Adaptive Algorithms, ACM Transactions on Internet Technology, Volume 4, November 2004.

[30] Depth-First Search, National Institute of Standards and Technology, Retrieved August 26, 2005, from http://www.nist.gov/dads/HTML/depthfirst.html.

[31] Depth-first Search, Brainy Encyclopedia, Retrieved August 15, 2005, from http://www.brainyencyclopedia.com/encyclopedia/d/de/depth_first_search.html.

[32] Tengli, A., Focused Crawling, Retrieved June 19, 2005, from http://www.cs.cmu.edu/~tengli/ clair/slides/Focused-Crawling.ppt.

[33] Best-first Search, National Institute of Standards and Technology, Retrieved August 26, 2005, from http://www.nist.gov/dads/HTML/bestfirst.html.

[34] Best-first Search, Brainy Encyclopedia, Retrieved August 15, 2005, Available: http://www.brainyencyclopedia.com/encyclopedia/b/be/best_first_search.html.

[35] Menczer, F., Pant, G., Srinivasan, P., Ruiz, M.E., Evaluating Topic Driven Web Crawlers, Proc. of the 24th Annual International ACM/SIGIR Conference, New Orleans, USA, 2001.

[36] Kleinberg, J., Authoritative Sources in a Hyperlinked Environment, Proc. of the 9th Ann. ACM-SIAM Symp. Discrete Algorithms, ACM Press, New York, 1998.

[37] Hersovici, M., Jacovi, M., Maarek, Y.S., Pelleg, D., Shaltaim, M., Ur, S., The Shark-Search Algorithm – An Application: Tailored Web Site Mapping, Proc. of the 7th International World Wide Web Conference, Australia, 1998.

[38] De Bra, P., Post, R., Information Retrieval in the World Wide Web: Making Client-based Searching Feasible, Proc. of the 1st International World Wide Web Conference, Geneva, 1994.

[39] IEEE Std 1016-1998, Recommended Practice for Software Design Descriptions, 1998

[40] Unified Medical Language System Knowledge Source Server, Retrieved: August 24, 2005, from http://umlsks.nlm.nih.gov/kss/servlet/Turbine/template/.

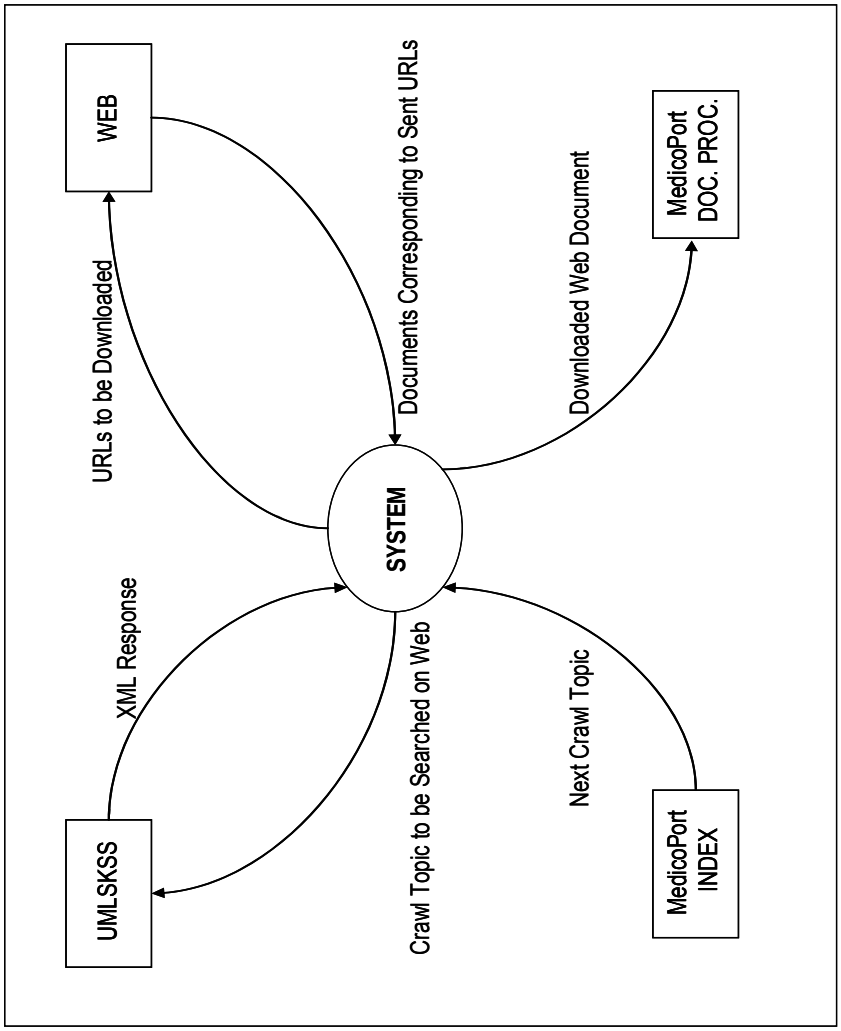
[41] Aggarwal, C. C., Al-Garawi, F., Yu, P.S. Intelligent Crawling on the World Wide Web with Arbitrary Predicates, In WWW10, Hong Kong, May 2001.

[42] Pant, G., Menczer, F., Topical Crawling for Business Intelligence, Retrieved: August 27, 2005, from http://dollar.biz.uiowa.edu/~pant/Papers/BizIntelECDL.pdf.

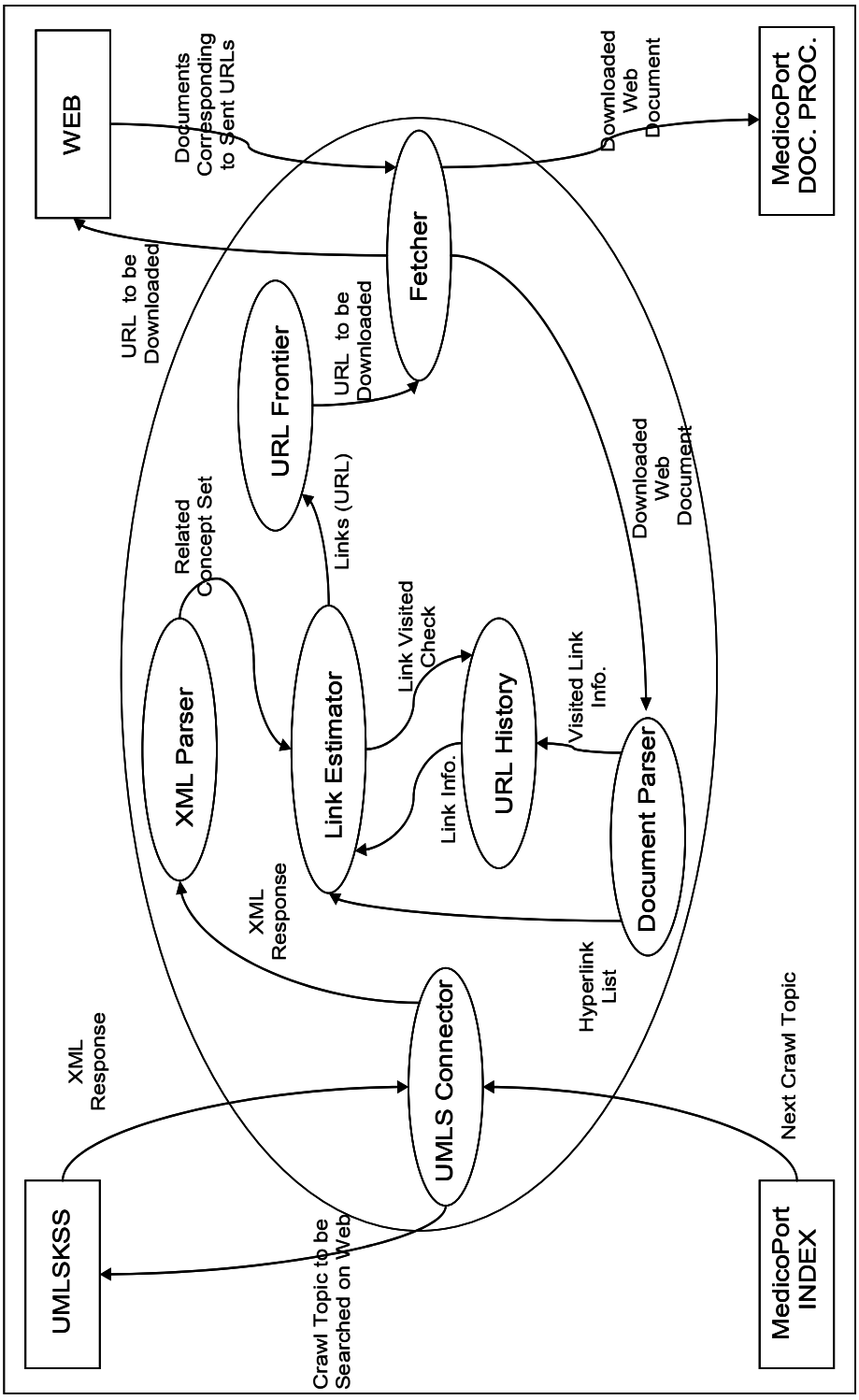
[43] MedLine All Health Topics Page, Retrieved August 31, 2005, from http://www.nlm.nih.gov/medlineplus/ all_healthtopics.html

APPENDICES

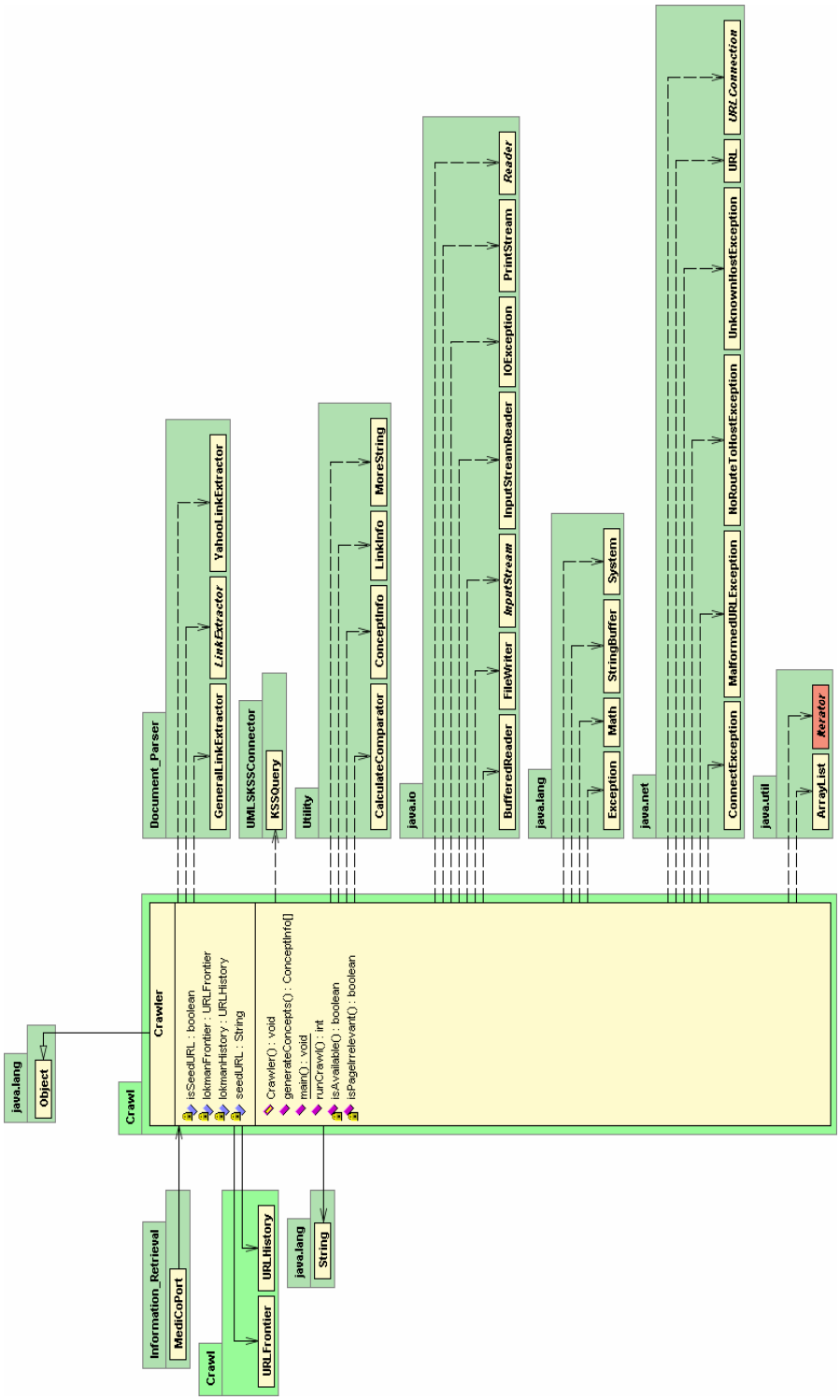
Appendix A: Lokman Crawler System Level 0 Dataflow Diagram



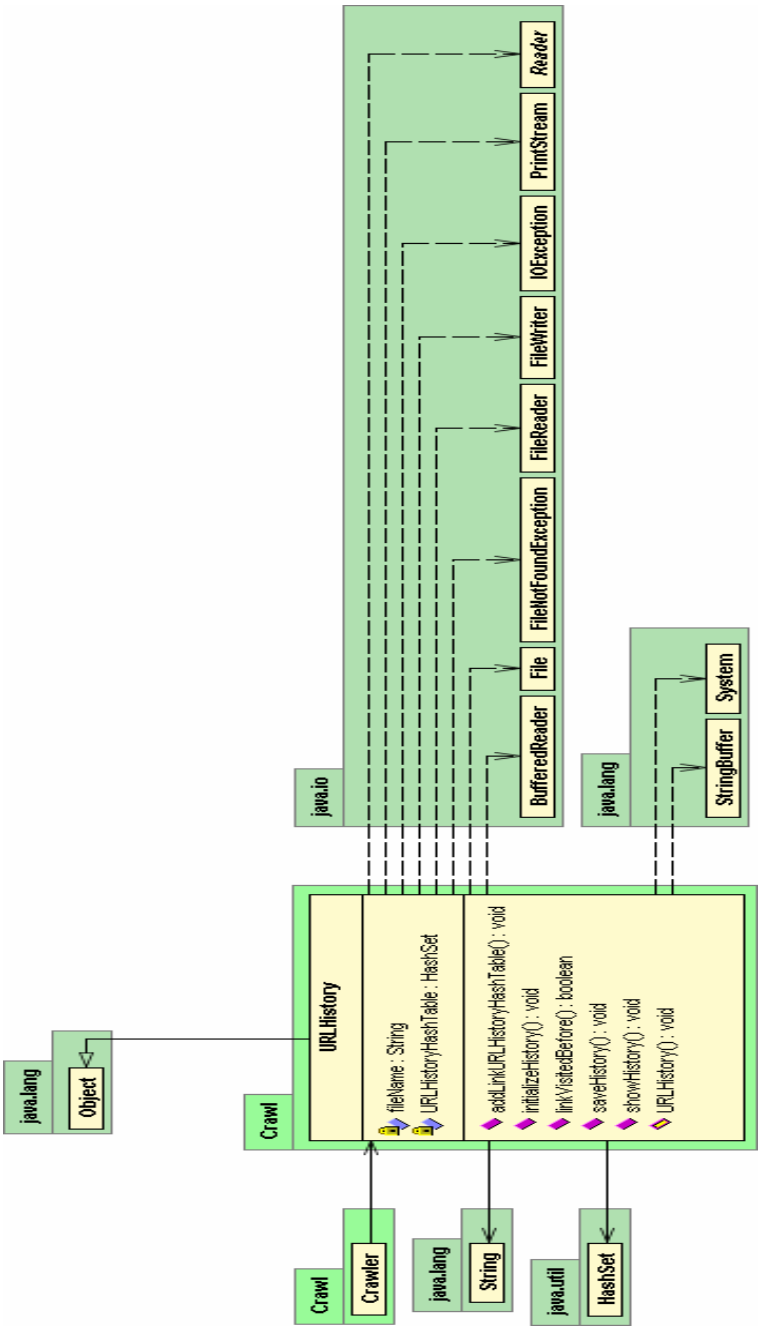
Appendix B: Lokman Crawler System Level 1 Dataflow Diagram



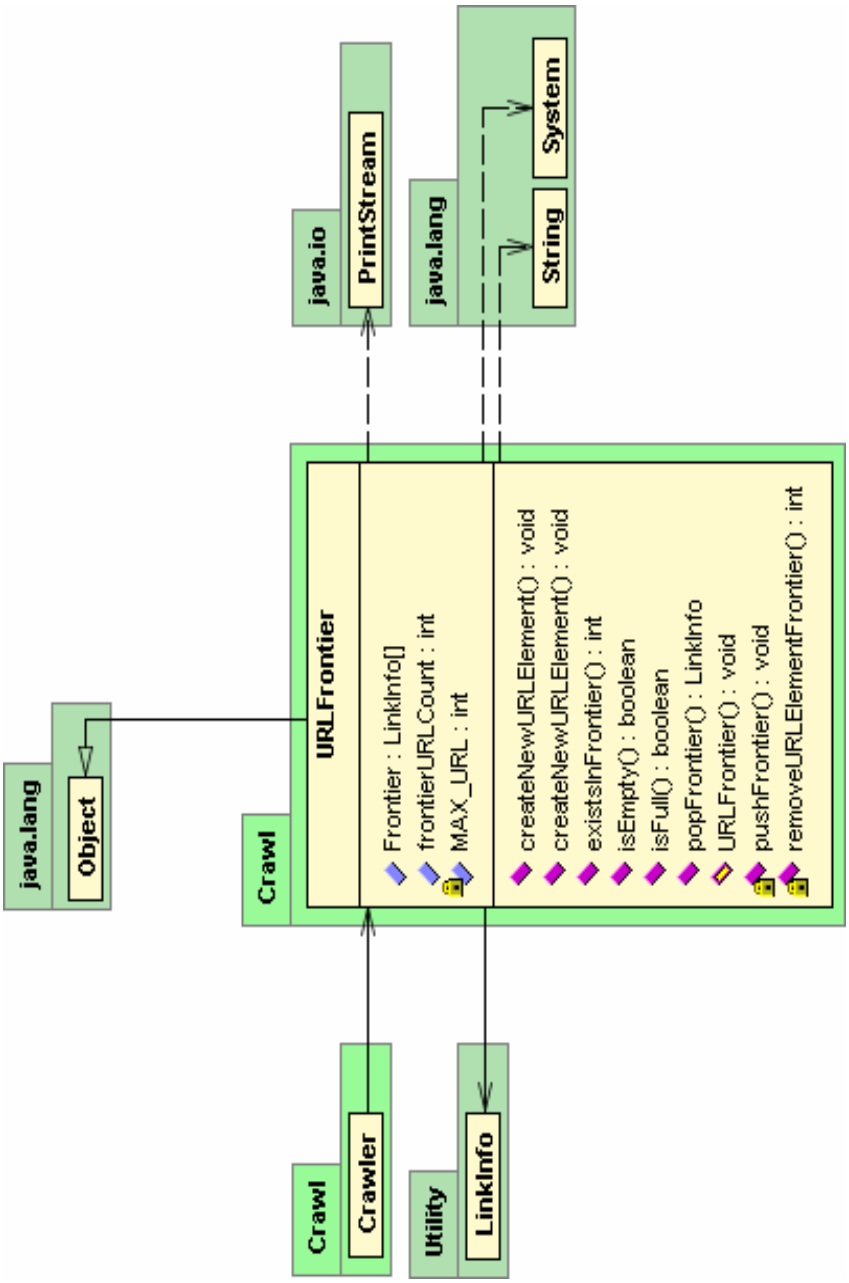
Appendix C : Package Crawl Class Crawler Class Diagram



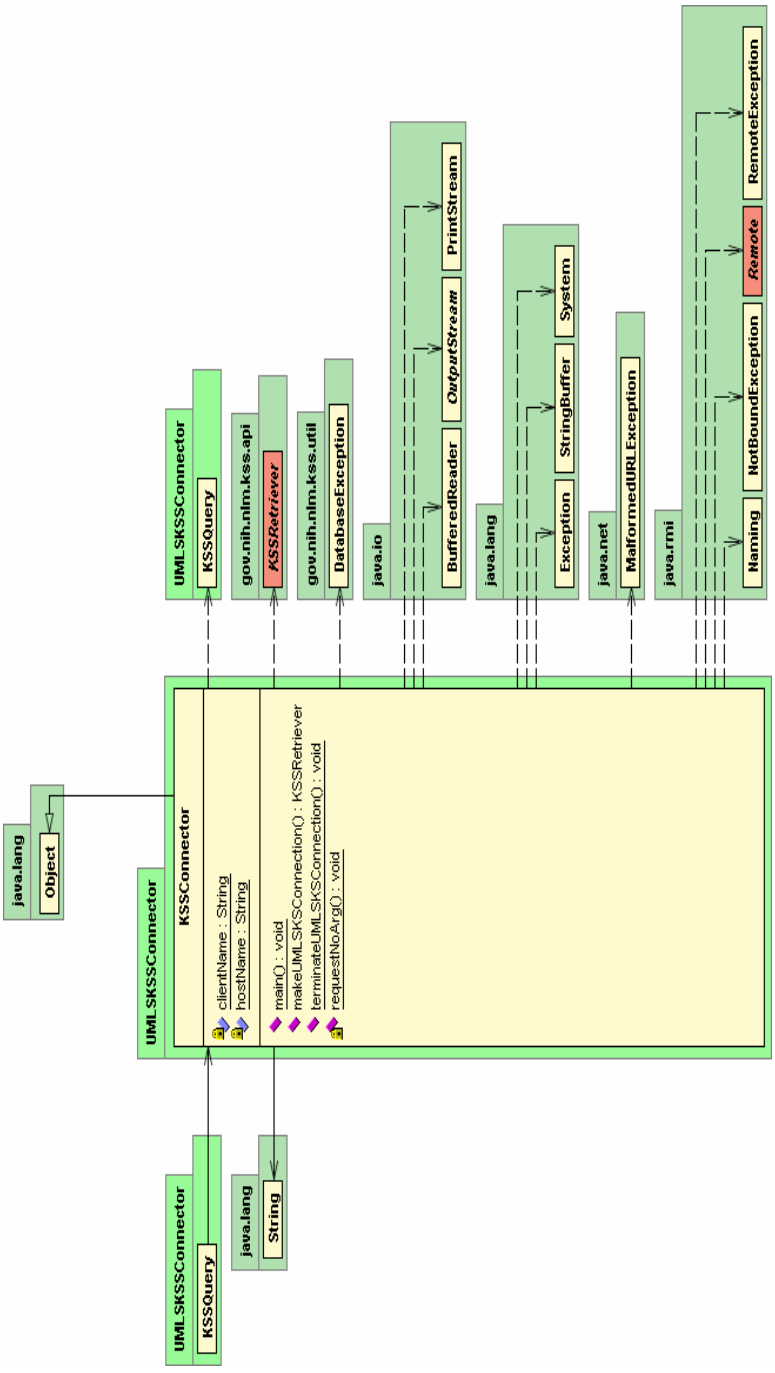
Appendix D: Package Crawl Class URLHistory Class Diagram



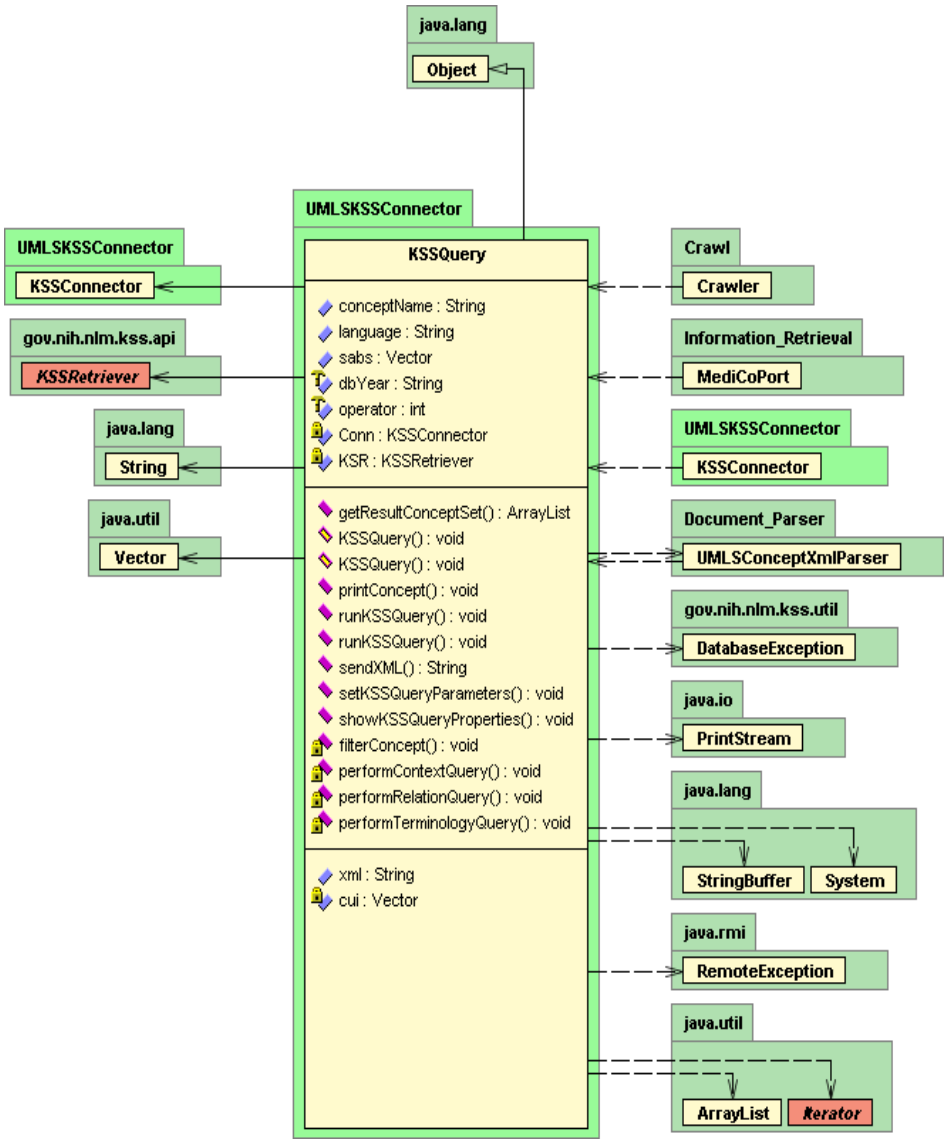
Appendix E: Package Crawl Class URLFrontier Class Diagram



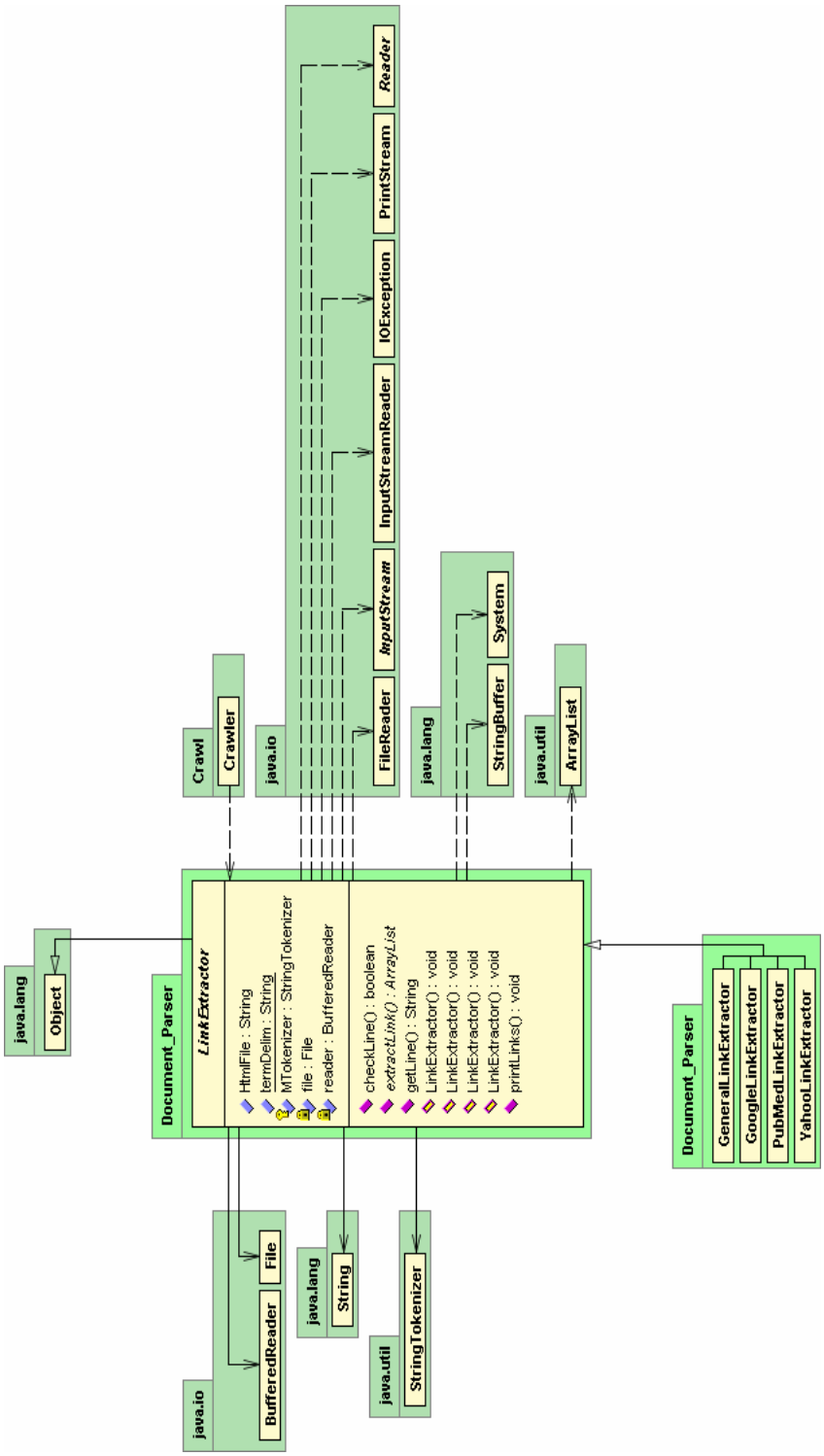
Appendix F: Package UMLSKSSConnector Class
KSSConnector Class Diagram



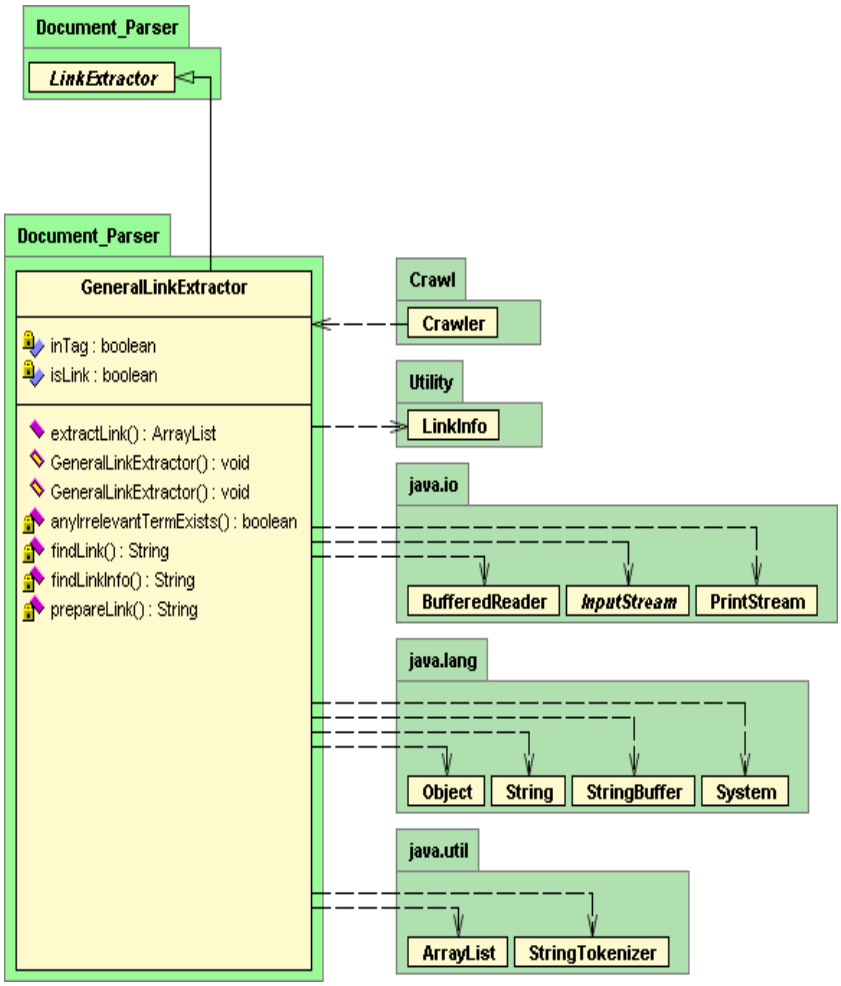
Appendix G: Package UMLSKSSConnector Class KSSQuery Class Diagram



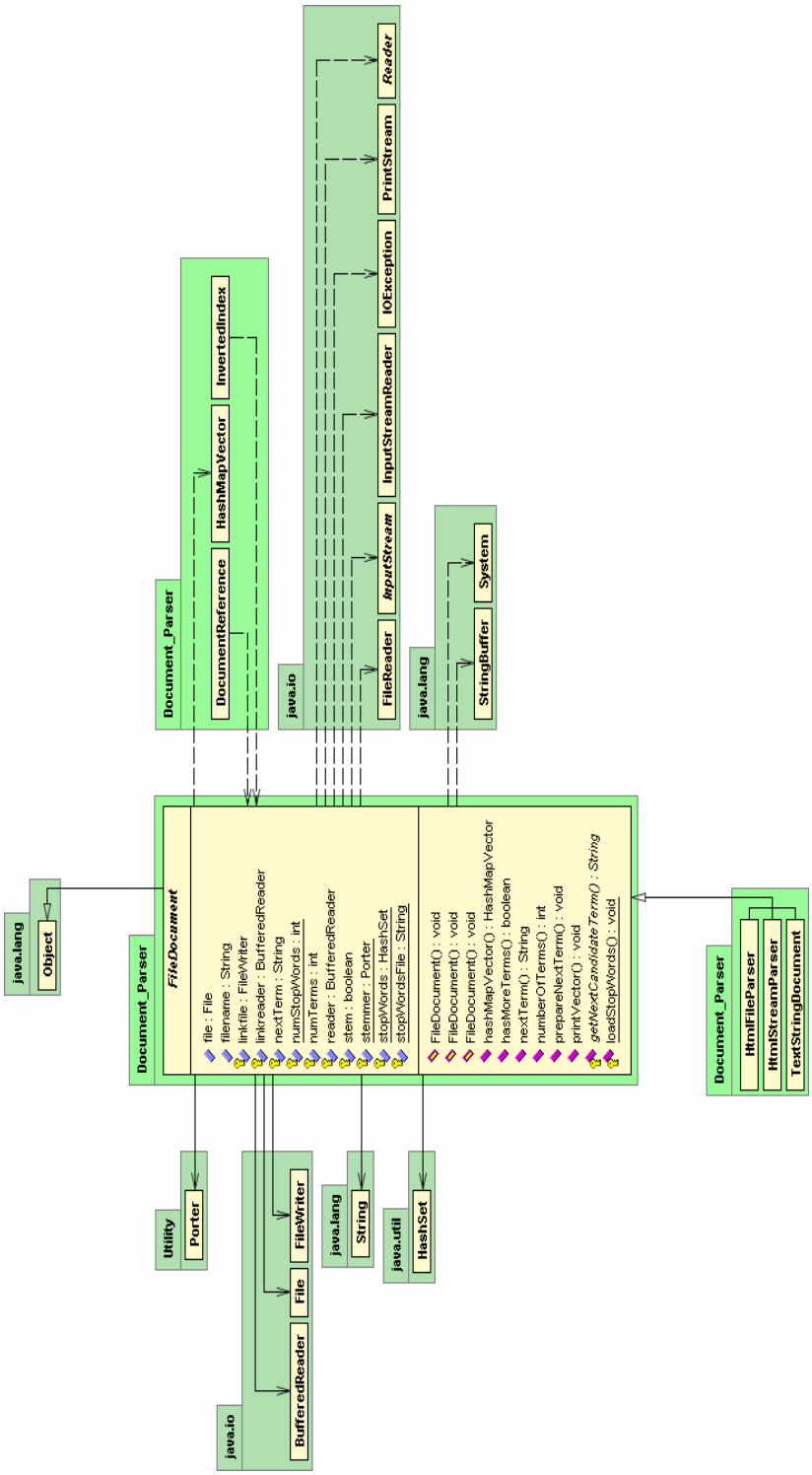
Appendix H: Package Document_Parser Class
LinkExtractor Class Diagram



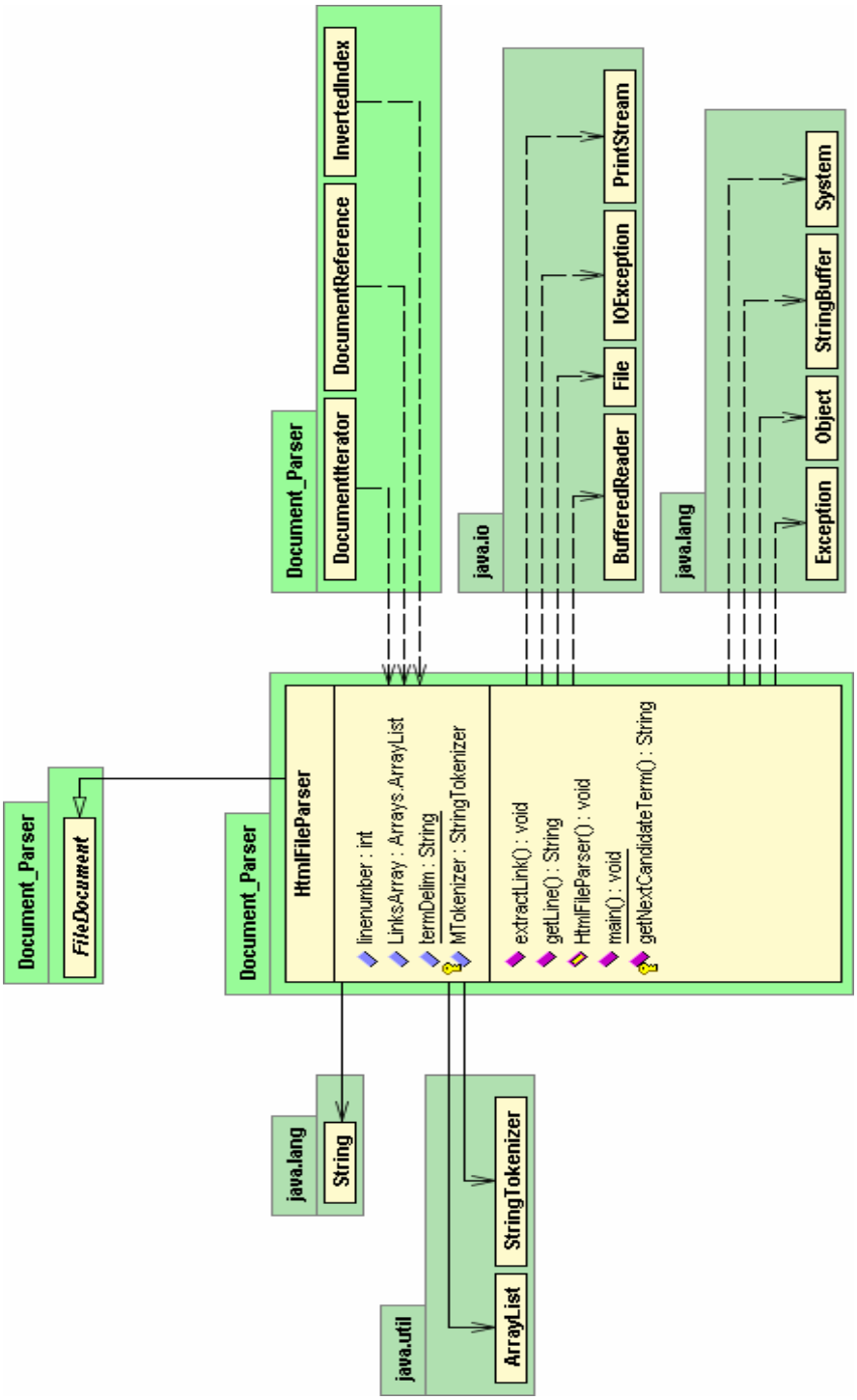
Appendix I: Package Document_Parser Class GeneralLinkExtractor Class Diagram



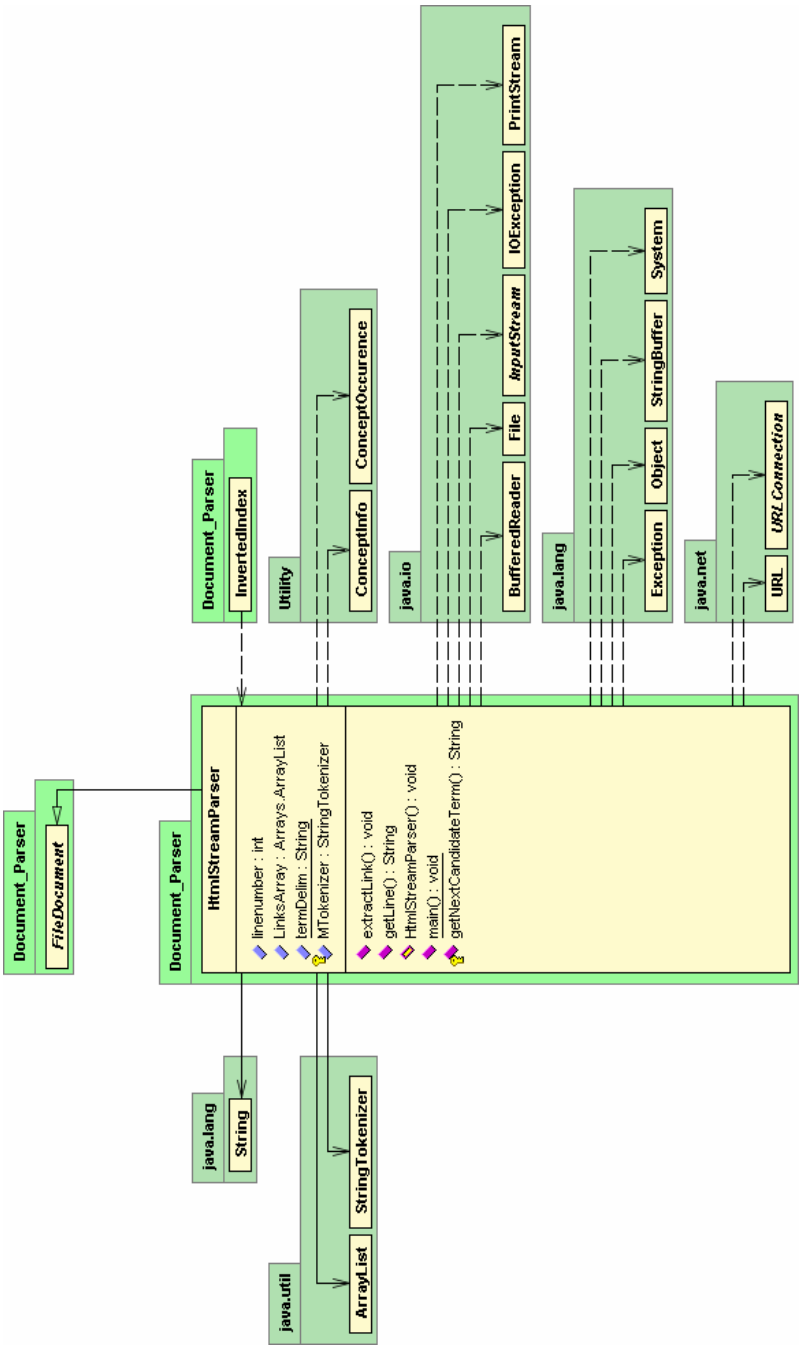
Appendix J: Package Document_Parser Class
FileDocument Class Diagram



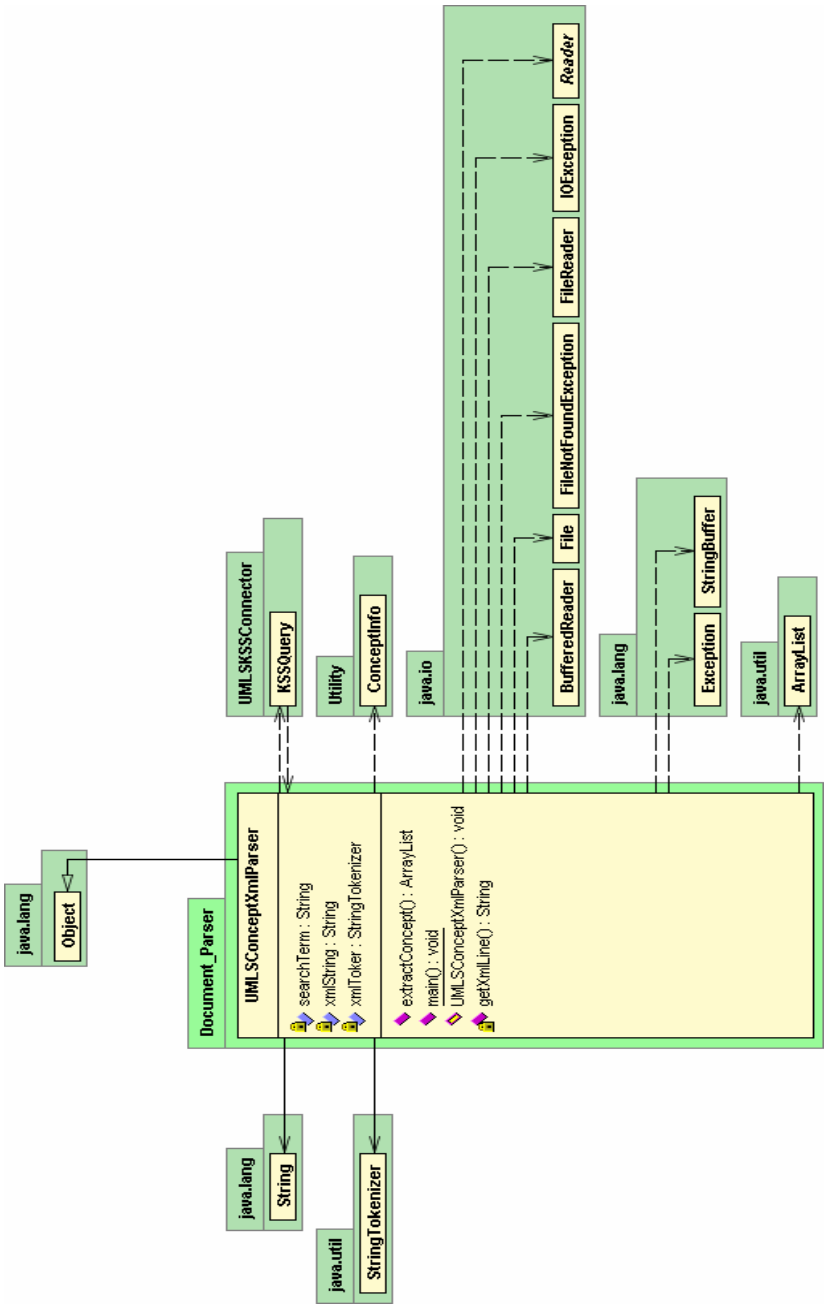
Appendix K: Package Document_Parser Class
HtmlFileParser Class Diagram



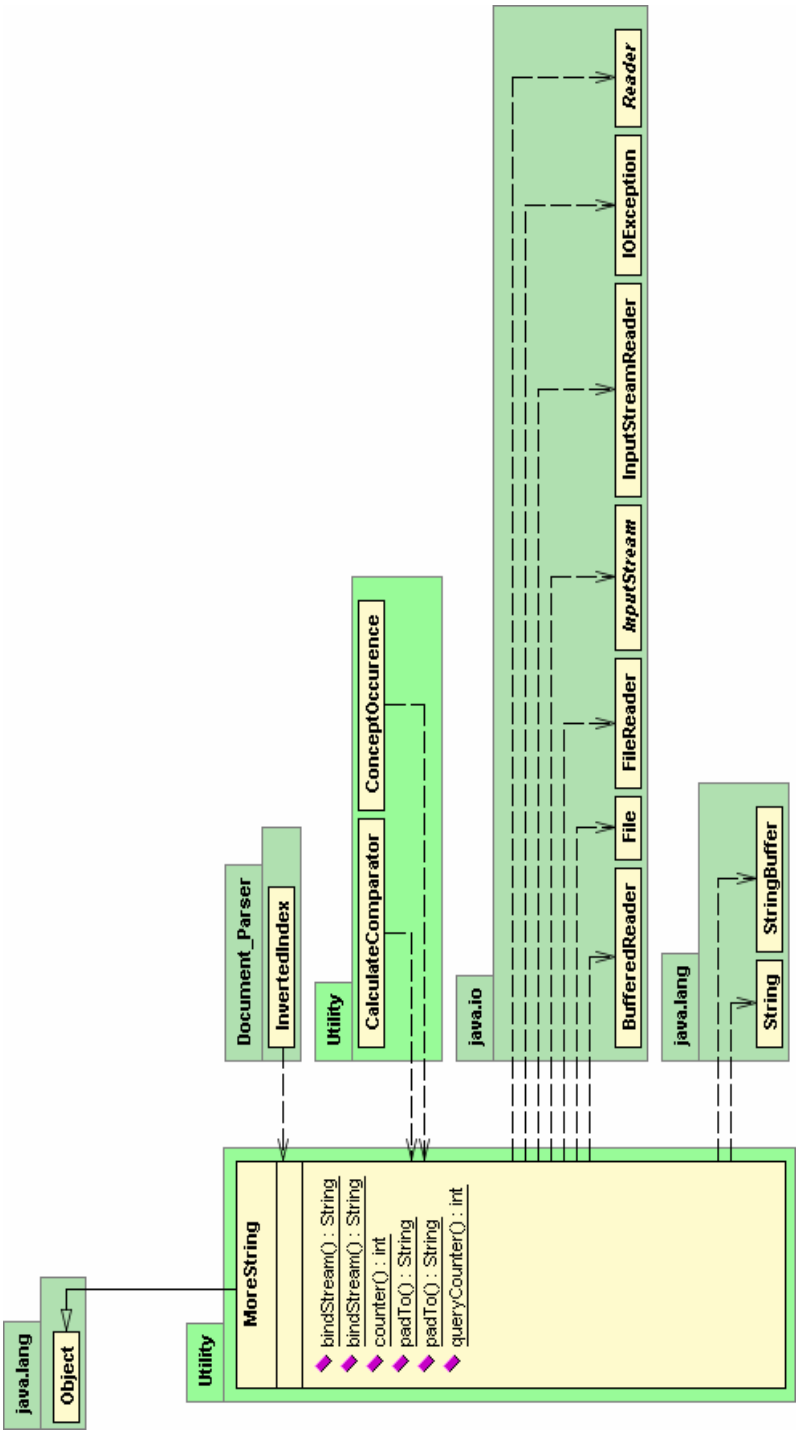
Appendix L: Package Document_Parser Class HtmlStreamParser Class Diagram



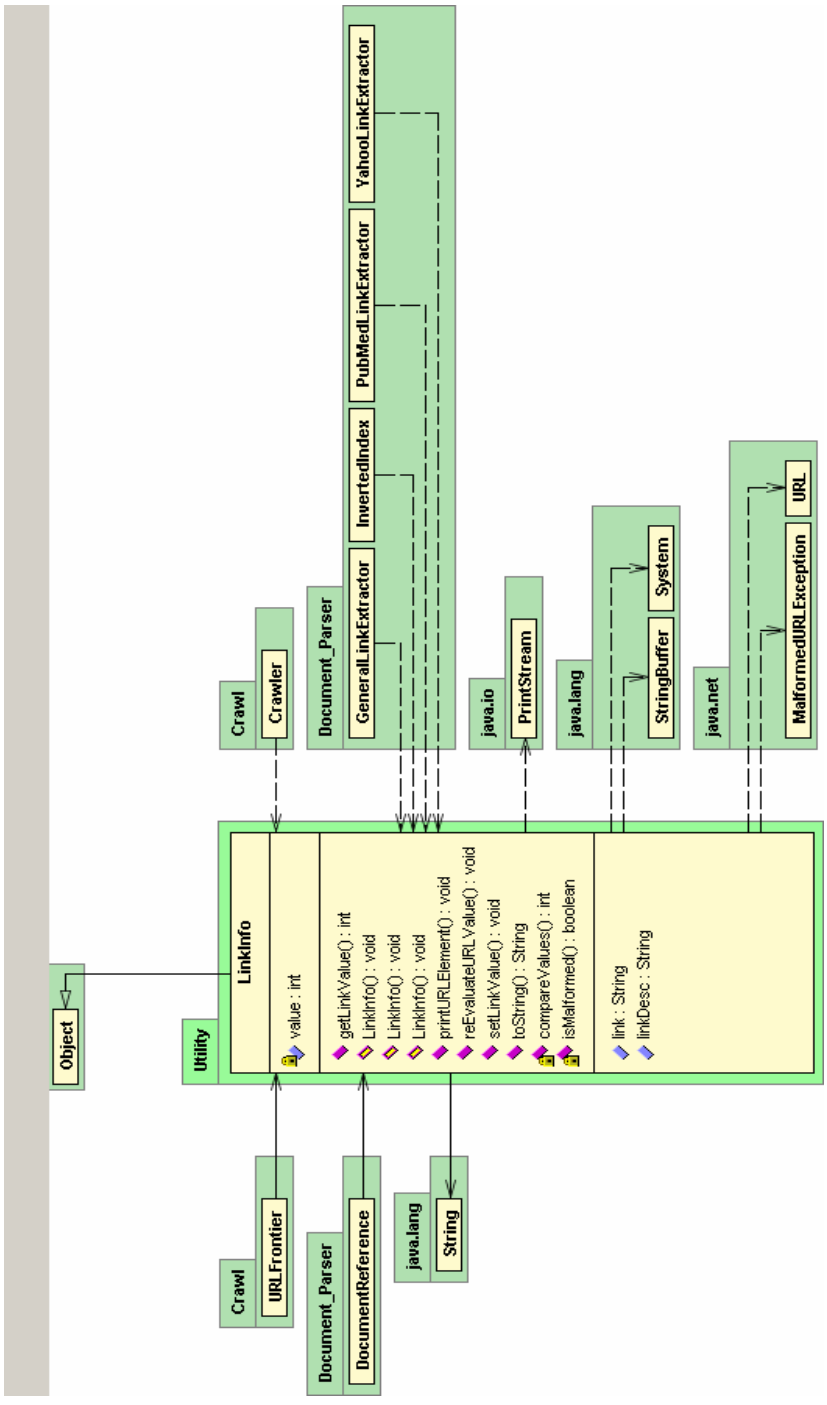
Appendix M: Package Document_Parser Class UMLSConceptXmlParser Class Diagram



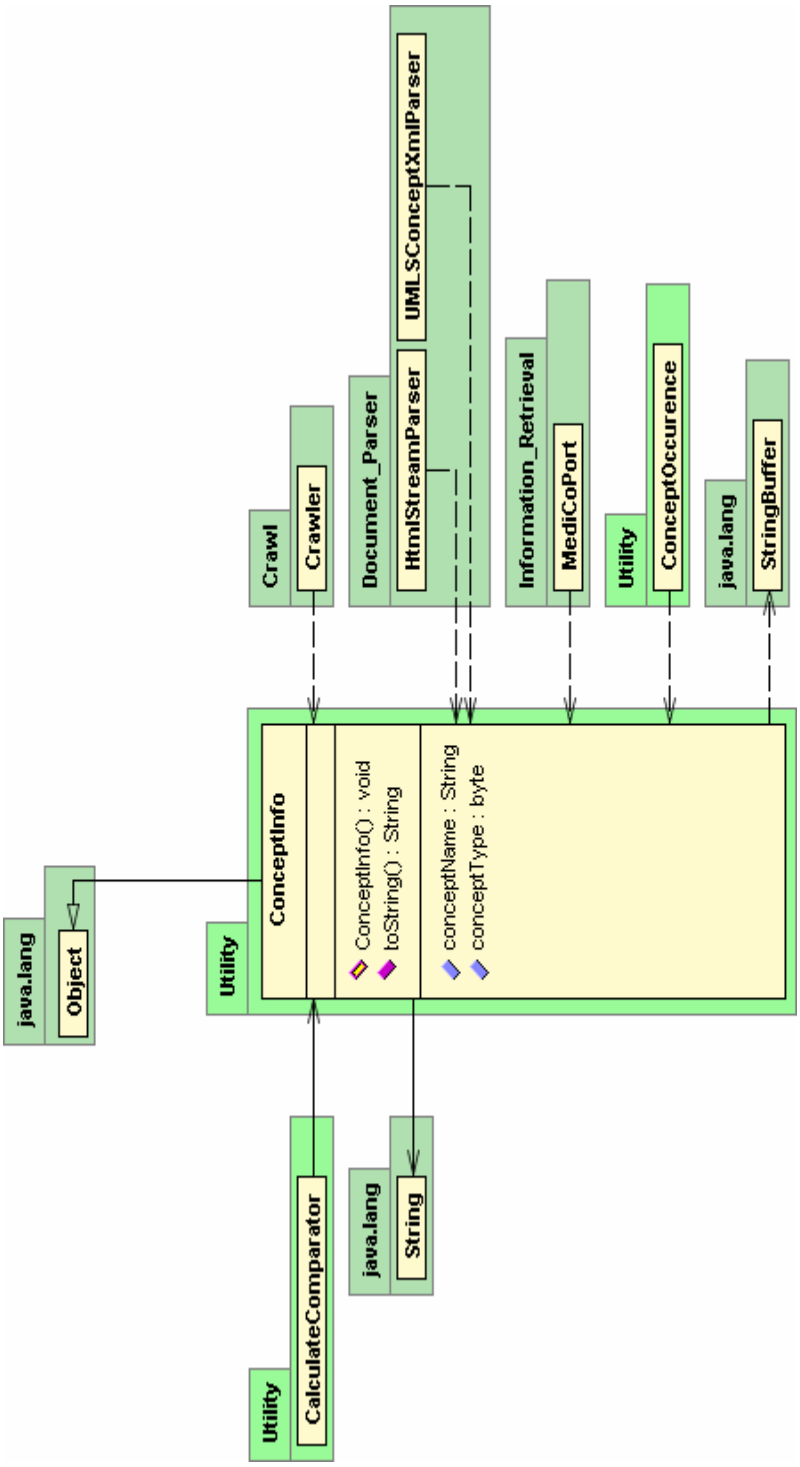
Appendix N: Package Utility Class MoreString Class Diagram



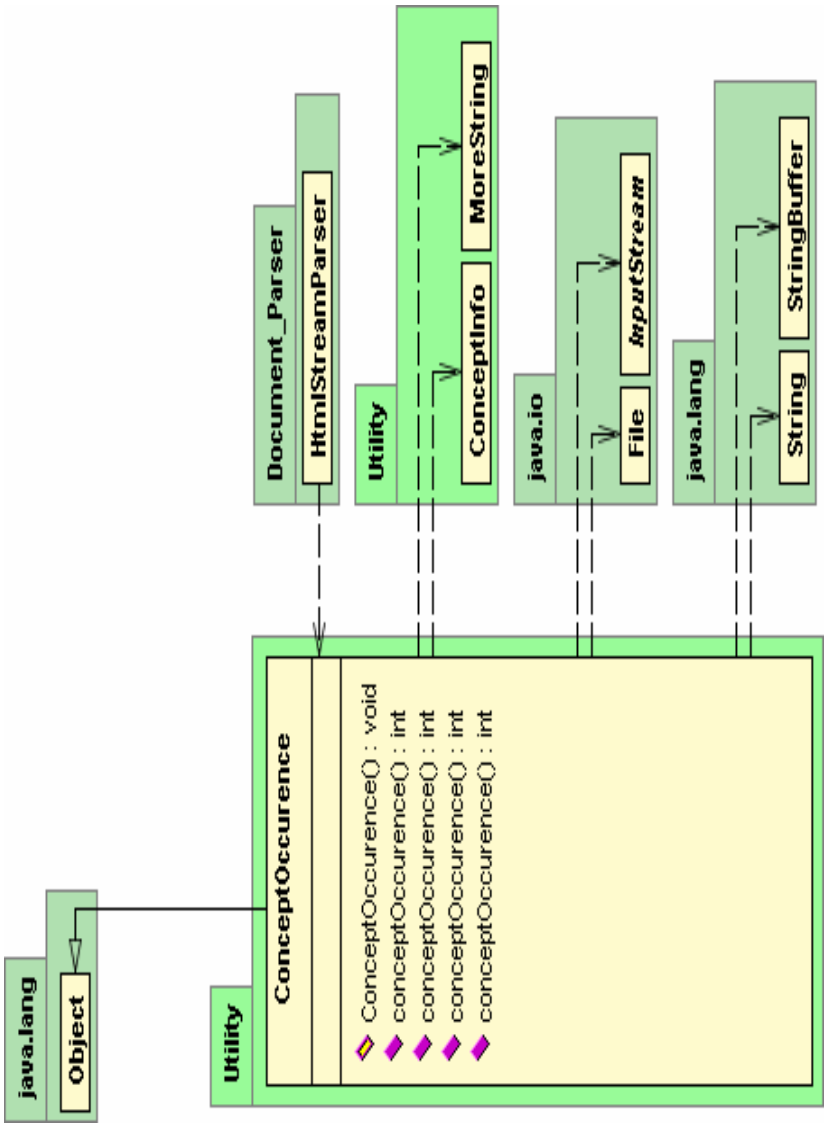
Appendix O: Package Utility Class LinkInfo Class Diagram



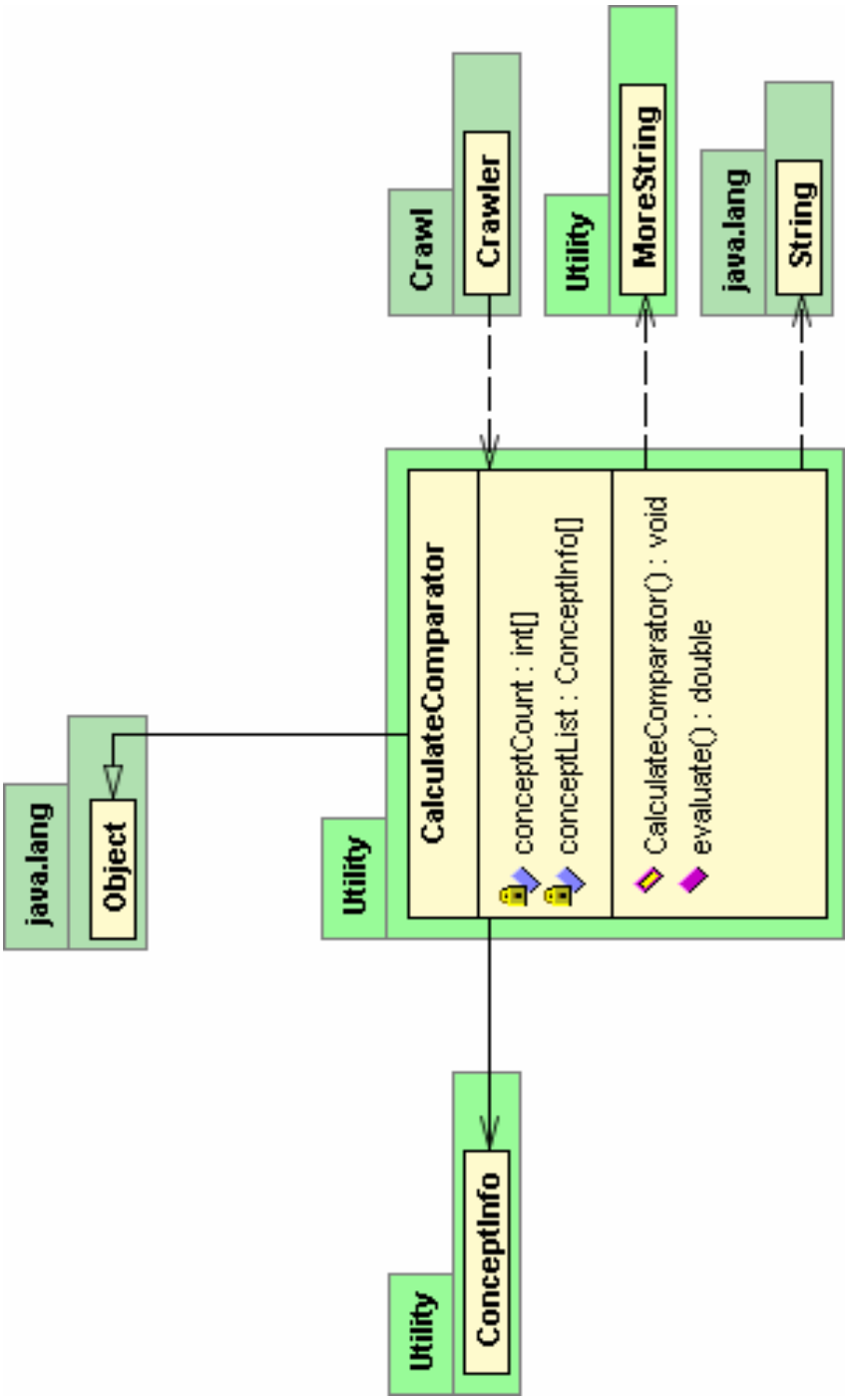
Appendix P: Package Utility Class ConceptInfo Class Diagram



Appendix Q: Package Utility Class ConceptOccurence Class Diagram



Appendix R: Package Utility Class CalculateComparator Class Diagram



Appendix S: Seed and Target URL Sets for Term "Bedwetting"

Seed URL Set

http://www.med.umich.edu/1libr/yourchild/topics.htm

http://about.com/health/

http://www.nlm.nih.gov/medlineplus/all_healthtopics.html

http://www.cincinnatichildrens.org/health/info/growth/diagnose

http://www.drgreene.com/54_28.html

http://www.noah-health.org/en/sleep/index.html

http://kidshealth.org/teen/diseases_conditions/

http://childparenting.about.com

http://familydoctor.org/alpha_results.xml?letter=B

http://www.medicinenet.com/site_map/article.htm

Target URL Set

http://pediatrics.about.com/od/bedwetting/

http://www.bedwettingstore.com/

http://www.medicinenet.com/bedwetting/article.htm

http://www.bedwettinghelp.com/

http://kidshealth.org/parent/general/sleep/enuresis.html

http://www.dryatnight.com/

http://www.bedwettinghelp.com/bedwetting_purchase.html

http://www.amazon.com/exec/obidos/redirect?tag=bedwettingsto-20&path=http%3A%2F%2Fwww.amazon.com%2Fgp%2Fbrowse.html%3F%255Fencoding%3DUTF8%26me%3DA26OASPQYU1JXQ

http://www.bedwettingstore.com

http://www.nlm.nih.gov/medlineplus/toilettrainingandbedwetting.html

http://www.medicinenet.com/script/main/art.asp?articlekey=47923

http://familydoctor.org/handouts/168.html

http://www.baltimorepsych.com/adhd_and_bedwetting.htm
http://www.wetbuster.com/
http://www.netdoctor.co.uk/diseases/facts/bedwetting.htm
http://www.pottytrainingsolutions.com/
http://www.drgreene.com/54_11.html
http://childdevelopmentinfo.com/disorders/bedwetting.shtml
http://www.keepkidshealthy.com/parenting_tips/bedwetting.html
http://www.daycare.com/fastfacts/bedwetting.html
http://www.cincinnatichildrens.org/health/info/growth/diagnose/enuresis.htm
http://hcd2.bupa.co.uk/fact_sheets/html/nocturnal_enuresis.htm
http://www.med.umich.edu/1libr/yourchild/enuresis.htm
http://sleepdisorders.about.com/b/a/190664.html
http://www.netdoctor.co.uk/diseases/facts/bedwetting.htm
http://www.bedwet.com/
http://www.herbalremedies.com/bedwetting.html
http://www.stopwetting.com/
http://www.health-nexus.com/enuresis-bed-wetting.htm
http://www.medical-library.org/journals2a/bed_wetting.htm
http://www.pediatricspec.com/Pediatrics/Articles/bedwetti.asp
http://www.surgerydoor.co.uk/medical_conditions/Indices/E/enuresis_and_bed wetting.htm
http://www.drgreene.com/21_1082.html
http://www.caringforkids.cps.ca/behaviour/Bedwetting.htm
http://www.nevdgp.org.au/ginf2/murtagh/Childrens/Bedwetting.htm
http://health.allrefer.com/health/enuresis-info.html
http://www.chiroweb.com/archives/17/03/34.html
http://health.allrefer.com/health/urination-bed-wetting-info.html
http://www.noah-health.org/en/sleep/specific/bedwetting.html