SPECIAL INDEX AND RETRIEVAL MECHANISM FOR ONTOLOGY BASED MEDICAL DOMAIN SEARCH ENGINES

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

BY

MUSTAFA KUBİLAY

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 DEMİRÖRS

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)	______________________

ABSTRACT

SPECIAL INDEX AND RETRIEVAL MECHANISM FOR ONTOLOGY BASED MEDICAL DOMAIN SEARCH ENGINES

KUBİLAY, Mustafa
M.S., Department of Information Systems
Supervisor: Assoc. Prof. Nazife BAYKAL

September 2005, 98 pages

This thesis focuses on index and retrieval mechanism of an ontology based medical domain search engine. First, indexing techniques and retrieval methods are reviewed. Then, a special indexing and retrieval mechanism are introduced. This thesis also specifies the functional requirements of these mechanisms. Finally, an evaluation is given by indicating the positive and negative aspects of mechanisms.

Keywords: Information Retrieval, Inverted Index, Ontology Based Search Engines, Search Engines, Vector Space Model.

ÖZ

ONTOLOJİ TABANLI BİR TIB ARAMA MOTORU İÇİN ÖZEL DİZİNLEME VE SONUC DÖNDÜRME YAPISI

KUBİLAY, Mustafa
Yüksek Lisans, Bilişim Sistemleri Bölümü
Tez Yöneticisi: Assoc. Prof. Nazife BAYKAL

Eylül 2005, 98 sayfa

Bu tez ontoloji tabanlı bir tıbbi alan internet arama motoru için dizin ve bilgi döndürme yapılarını konu alır. İlk olarak, dizinleme ve bilgi döndürme teknikleri ele alınmıştır. Daha sonra, gerekli olan dizin ve bilgi döndürme yapıları tartışılmıştır. İşlevsel gereksinimler belirlenmiş ve sistem gerçekleştirilmiştir. Son olarak, sistemin olumlu ve olumsuz tarafları test edilmiş ve sonuçlar değerlendirilmiştir.

Anahtar Kelimeler: Bilgi Döndürme, Devrik Dizinler, Ontoloji Tabanlı Arama Motorları, Arama Motorları, Vektör Boşluk Modeli.

To my mother and my sister,

ACKNOWLEDGEMENTS

I first thank my advisor Assoc Prof. Dr. Nazife BAYKAL for providing guidance, encouragement and patience, which promoted this study. I am grateful to my project mate Altuğ KAYIŞOĞLU for his inspiration and desire.

Finally, words cannot truly express my deepest gratitude to my mother and my sister who always gave me their love and emotional support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

API	: Application Programming Interface
CTX	: Contextual Relation
CUI	: Concept Unique Identifier
HITS	: Hyperlink-Induced Topic Search
HTTP	: HyperText Transfer Protocol
IP	: Internet Protocol
LSI	: Latent Semantic Indexing
MAC	: Media/Medium Access Control
NLM	: National Library of Medicine
PAR	: Partially Relevance.
RAM	: Random Access Memory
SYN	: Synonymy
TCP/IP	: Transmission Control Protocol/ Internet Protocol
UMLS	: Unified Medical Information System
UMLSKSS	: Unified Medical Information System Knowledge Source Server
URL	: Universal Resource Locator
tf/idf	: Term Frequency/Inverse Document Frequency
WebOnt	: Web Ontology Working Group
XML	: eXtended Markup Language

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

Due to the advances in computer technology and introduction of the Web to the community, traditional methods to access/publish information have changed fundamentally. This great impact allows information seekers to reach any information repository on the web eliminating geographical distance as well as information publishers introduce their resources rapidly and easily 0.

On the other hand, the motivation to use the web also involves difficulties for the community when the issue is finding the useful information. Since there is no control over the web, many types of documents exist within. In addition, there are no limitations or rules for representing information like the usage of formal language. Another important point, which makes the challenge harder, is the growth/change rate of web.

Considering the size of the web and the change rate of the documents, it is obvious that it is impossible for an individual to find out the documents of his/her interest without computer assistance [2]. Most common and well known software agents dealing with this problem are web search engines.

Web search engines are complex structures to develop, but as a principle, they provide a very simple user interface. An information seeker simply types down his/her need in the edit field and gets the corresponding hyperlinks according to the specification. Consequently, traversing the hyperspace through a web search engine's response page is more popular for most of the internet users than typing down the URL of each desired page in the web browser's address field.

To serve the users in such a manner, a web search engine has many tasks to complete. Simply, it uses the same strategy with any web surfer. A component named crawler visits the pages on web and downloads them. Then each downloaded web page is saved to a special data structure named index. When a user invokes the searching module, search is performed on the index and user is granted a large set of answers in a very short period time.

But search engines never promise the guarantee of satisfaction due to multiple reasons which stem from the difference between human cognition and machine processing. First of all, a user's specification of information need and machine's treatment to the specification might be different. User might receive an answer set related with other fields of interest which are indexed with same term or a set in another language with same term but also an irrelevant meaning. Another fact which makes the "search on the net" problem more challenging is, user has no chance to find a document related to his need but not indexed with any user query terms. Last, but not least, search engines generate a long list of query responses as answer set which sometimes reach to millions in number.

Such great number of hyperlinks is impossible for an individual to visit regarding the normal user behavior. Mostly first 20 links are visited and rest is omitted. Therefore, ranking the hyperlinks according to relevance to the user query is very important to construct a successful search engine.

Although search on the web bears all the above problems, a new approach, use of ontology in web search, promises better results than conventional search engine strategy. When a search engine is empowered by ontology, it becomes a domain web search engine and specializes in a single area of interest. This way, terms to be indexed are shrunk to a terminology of a domain from any possible term of a language. In addition, special relations defined in the ontology help developers implement special routines that allow the system to retrieve documents of interest indexed with other terms.

To find out the advantages of domain knowledge use in medical search on web, MedicoPort project was proposed in September 2004. It is a medical domain web search engine enhanced with medical ontology. 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 useful links related to medicine.

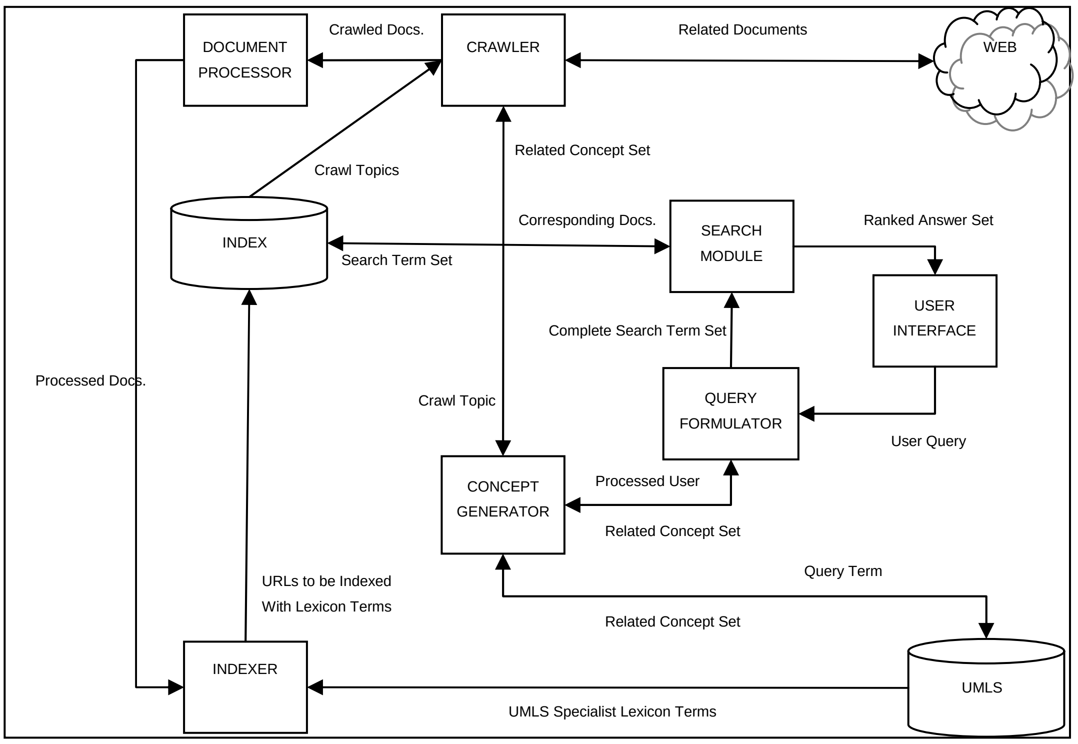


Figure 1: MedicoPort System Overview

MedicoPort binds the advantages of ontology use and old well-known search engine strategies. Obtained domain knowledge is used throughout every task during retrieval process. These tasks include crawling, URL prioritization, URL selection, URL indexing, user query formulation and result ranking.

MedicoPort is compound of three major subsystems. These include crawler module, indexing module and retrieval module (Figure 1). Other modules in the system exist to support these major subsystems. This study is performed in order to implement index and result ranking (retrieval) subsystems used in the project. Crawler subsystem, Lokman Crawler, is implemented by Altuğ Kayışoğlu [3].

1.2 Ontology

Ontology is a powerful way of information modeling. It categorizes every concept in a domain. Each concept in the domain can be organized in this structured categorization [4,5]. Ontology has a taxonomy which defines the concepts (classes) in a domain and a rule set describing the interrelations among these concepts 0. It is designed to define knowledge, reuse it and share it [6]. Ontology is data defining the relations and concepts in a domain.

Unified Medical Language System (UMLS) ontology is used in this thesis study. For further information on this issue see [3].

1.3 Thesis Structure

This text is organized as follows:

Section 2 provides the related research on indexing and result ranking strategies. Firstly, main indexing techniques, index compression algorithms and index distribution algorithms are presented. Secondly ranking algorithms are introduced.

Section 3 includes the analysis study, design principles and a detailed overview of the indexing and result ranking (retrieval) subsystems of MedicoPort.

Section 4 covers the evaluation of the implemented subsystems.

Section 5 provides the conclusions, possible future work directions for the subsystems and function of the study in means of contribution effort in the MedicoPort project.

CHAPTER 2

Literature Survey

This chapter introduces main indexing structures: inverted index (files), suffix trees, signature files for information retrieval systems. Also compression of indices and distribution of indices are explained. Finally, retrieval models (Set theoretic, Algebraic, Probabilistic) are presented.

2.1 Types of Index Structures for Information Retrieval Systems

To retrieve an answer of a query, first solution is scanning the text sequentially. Sequentially or online text searching involves finding the occurrences of a pattern in a text if the text is not preprocessed. This can be a choice only if text database is volatile or index space overhead cannot be afforded [1].

Second option is building a special data structure over the text to speed up the search. These data structures are called indices and they are suitable for semi-static (that it can be updated at reasonably regular intervals, e.g. daily but not in a second) and large databases [1]. Like other research area today, hybrid solutions that contain both online search and indexing search are very popular. This section covers three main indexing techniques: inverted files, suffix arrays and signature files.

Inverted files are currently the best choice for most applications. Inverted files are useful, because the information retrieval search strategy is based on the vocabulary which is usually much smaller than text. Suffix arrays are faster for phrase search and less common queries, but they are harder to build and maintain. Signature files are not useful relatively. Inverted files outperform signature files on search cost, space overhead building cost and also updating cost.

2.1.1 Inverted Files

Inverted file is a word-oriented mechanism. It has two components first of which is the vocabulary. Vocabulary is the set of all different words in the text. The second component is occurrences. Occurrences hold text position of a word. Word or character positions can also be used for occurrences. Word positions are practical for phrase and proximity queries and character positions are useful for direct access to the text position.

Vocabulary space requirement is O(n^b). Value of b is 0.4 to 0.6 [1]. Occurrences' space requirement depends on granularity of addressing. If a full inverted index is used, addressing words occurrences' space requirement 30-40% of text size without stop-words. If block addressing technique is used, then needed space requirement is decreased since pointers' size become smaller and some word pointers that indicate same blocks can be collapsed.

Blocking strategy can be applied in two ways; the blocks can be divided to fixed size or natural division can be used (e.g. document by document). Natural division is useful whenever exact position is not required (as in MedicoPort). But if many block retrieval units are packed in one block, it causes that block to be traversed to learn which term is retrieved [1].

Below searching procedures are performed in inverted files:

- Vocabulary search: finding a single word in a vocabulary,
- Retrieval of occurrences: list of the occurrences that are retrieved,
- Manipulation of occurrences: processing of occurrences lists.

Vocabulary structure can fit the main memory even for large text sets. So it should be a separate file. To speed up the searching, hash trees, B-trees or Trie structures can be used. Search cost of hash tree and trie structures are independent of text size, it depends on term size. Lexicographical order of vocabulary also provides improvement for searching speed.

In context queries, inverted indices are ineffective. For this type of queries, suffix arrays outperform inverted files.

The most time-demanding operation on inverted indices is to merge the occurrences list. This is because, usually occurrences lists can't be kept in main memory and some disk access operations are required. Compression algorithms and parallel (distributed) schemes are candidates for solution.

An inverted index of n characters can be built in O(n) time [1]. Maintenance is also inexpensive for inverted files. Cost of deleting a record is O(n). Replacement cost is O(n + n' log(n'/M)) (n' is size of new text and M is available capacity of main memory) if occurrences lists (posting file) are not fit to main memory. Algorithm of recording a new word is as follows:

```
While Eof (NewText) do
        Search(wi) in trie
        If it is not found
                Add(wi) to trie,
                Add an empty occurrence list to posting file
                Write position of (wi) in occurrence list
                Add pointer of new occurrence list' pointer to trie
        Else
                Add position of (wi) to related occurrence list
        Increment i
Write to final trie to disk with occurrence lists.
```

If index does not fit in memory, partial index is used. Finally, m (index size / memory size) partial indices are obtained. If partial indices are merged two by two, complexity of this operation is O(Xi + Xj) (X is the size of each partial index). An improvement can be provided if more than two indices are merged at once. Usually 20 or 30 % of total time is required for merging [1]. In merging process, to reduce space requirement merged new indices can be written to same disk address that belong to old ones.

As seen, inverted index is suitable for word search and its building and maintenance operation is also inexpensive. Improvement can be provided with compression and distribution approach, too. But queries like phrases queries are expensive to solve.

2.1.2 Suffix Tree and Arrays

Definition: **A suffix tree** is a trie data structure built over all the suffixes of the text.

In computer science, a trie is an ordered tree data structure that is used to store an associative array where the keys are strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree shows what key it is associated with. All the descendants of any one node have a common prefix of the string associated with that node, and the root is associated with the empty string. Values are normally not associated with every node, only with leaves and some inner nodes that happen to correspond to keys of interest. “A trie is a tree of degree m >= 2 in which the branching at any level is determined not by entire key value but only a portion of it” [7].

Suffix array is a space efficient implementation of suffix trees. Solving complex user queries with inverted files is inefficient but suffix trees and arrays are built for this purpose. This index structure assumes that the text as a long string. Each position is considered as a suffix of text. Not all the positions need to be indexed. May be word beginnings suitable in our situation. Main drawbacks of this structure are: its costly construction process, the text must be available at query time and the query results are not delivered in text position order additionally in this structure only indexed elements can be retrieved. Suffix tree pointers are stored in leaf nodes of tree structure. Depending on the implementation pointers, they can hold word beginnings or every character positions. Even if only word beginnings are indexed, needed space requirement of suffix trees can be 120 to 240 % over text size [1].

To reduce space overhead without reducing functionality suffix array can be used. Suffix array is simply an array containing all the pointers to the text suffixes in lexicographical order. Suffix array space requirement are almost the same as those for inverted file, which is close to 40% of text size [1].

Binary search is suitable for suffix array but if array length is too large because of disk operations it is inefficient. Solution of this problem is supra-indices. Supra-index is sampling of one out of b suffix array entries. It is used as a first step of the search to reduce external accesses. Supra-index seems like vocabulary of inverted index and it also needs extra space requirement. If we use vocabulary supra-index then space requirement will be same as for inverted index.

Search cost of suffix tree is O(m) [1](m is the length of query term) but it is not suitable for large text because of its space requirement. Suffix array search cost is O(log n) because of binary search. If disk operation cost is added then total cost is O(n log n) [1].

If supra-index is used in order to reduce search cost, time requirement decreases to 75% of original time. Nevertheless if search terms are simple words search cost can be worse than inverted index.

A suffix tree for a text of n characters can be built in O(n) time. Because of big space requirement of suffix tree it must be convert to a suffix array. This operation needs extra time and also it increases construction complexity. Construction of suffix arrays, for large text is required one more extra time for partial indices if it is not fit in main memory,

2.1.3 Signature Files

Signature files are another word-oriented index structures that are based on hashing algorithm. However its search complexity is suitable for large text, inverted files outperforms signature files for most application [1, 8]. A signature file uses a hash function to map words to bit masks. Firstly it divides text to blocks then to each block a bit mask is assigned. It means that signature file is a sequence of bit masks of all blocks and a pointer to each block. Each block bit mask contains its word's bit masks. Consequently a query term can be checked whether it is in a block or not easily.

Checking a word is a simple Boolean operation. When you have a query term: Find the bit mask of term (result of hash function that takes query term is bit mask) do a bitwise AND operation with bit mask of term and bit mask of block. If the result of bitwise operation same with bit mask of term current block contains this term.

There are some drawbacks of this scheme. It is possible that all the corresponding bits are set even though the word is not there. This is called "false drop". The false drop is a function of the space requirement. It means that a bigger index size, the smaller false drop probability. "For instance a 10% overhead implies a false drop probability close to 2% while a 20% overhead errs with probability 0.046%."[1] The other problem is the exact position of the query term can't be known after the block checking operation. For all candidate text blocks an online traversal must be performed to verify whether the query term is actually there and the exact position of the term.

This scheme is more efficient to search phrases and proximity queries. You can learn in one operation whether all the terms of phrases is in the block or not with a pre-bitwise OR operation of terms. There is one more problem for this operation. Because of block boundaries all phrases may not be caught. In order to catch all the phrases, consecutive blocks must overlap in n words. This n must be at least number of term of phrase.

Construction of signature files is easy: the texts are simply cut in blocks and for each block an entry of signature file is generated. This signature is the bitwise OR of the signatures of all the words in the block. Adding text is also easy, since it is only necessary to keep adding records to the signature files. Text deletion is carried out by deleting the appropriate bit masks.

2.1.4 Deep Web Index Structures

Some studies estimate that the size of the deep web is 400-500 times the size of the surface web [9]. According to [9], for each distinct value of the indexing attribute, the list of data sources can be stored instead documents. Data sources contain at least one tuple that has the value for the indexing attribute. This is the main idea of deep web indexing structure but it depends on just online searching and it retrieves only the data sources not documents. In [9] there are some extensions of this scheme. Next paragraph explains some of them.

Data-Source Clustered Index: DCI instead of storing data sources separately, a single cluster id is stored in the data source list. Cluster id refers clustered data sources. **Value-Data-source Clustered Index:** In VDCI, a cluster is a set of values V and a set of data sources D, and implies that the data sources D are related with respect to the values V (or equivalently, the values V are related with respect to the data sources D). Thus, VDCI generalizes both VCI and DCI. It is a hybrid models. **Histogram Based Index:** HBI is based on clustering adjacent values into the same cluster. This is similar to grouping attribute values into “buckets” in histograms of relational systems.

2.2 Compression of Indices

Today sizes of index structures of search engines are huge because of the size of the web [9] and they don’t fit in a memory usually. On the other hand, disk operations are slow for fast information retrieval. Solutions of these problems are hidden in compression algorithms.

Compression reduces both the size of indices and the time needed to evaluate queries. As it was mentioned in Section 2.1.1, uncompressed inverted index structure consumes more than 30% of the space required to store the original documents. But when the index is compressed, the index size is reduced to between 10-15% of original documents size. Index compression also provides speed-up up to twice. Basically there are two main approaches Bitwise schemes and Bytewise schemes [10]. There are hybrid methods for compression too.

Bitwise or Variable-bit schemes store integers in an integral number of bits [11]. Well-known types of bitwise schemes are Elias gamma and delta coding [12] and Golomb-Rice coding [13]. In the Elias gamma code, an integer x >= 1 is represented by 1 + [log2 x] stored as a unary code. Elias delta code stores the gamma code representations of 1 + [log2 x] and then the binary representation of x without most significant bit.

According to [14] gamma coding is relatively inefficient for storing integers larger than 15 and also delta codes are suited for large integer but are inefficient for small integers. Golomb-Rice bitwise coding offers more compact storage of integers and faster retrieval than Elias codes [11]. In the Golomb-Rice code, integer x >= 1 is represented relative to parameter b by sending [(x-1)/b] using a zero-origin unary code; and then x-b*[(x-1)/b] using a one-origin binary code of either [log2 b] or [log2 b] bits. [15] For compression of inverted list in Golomb-Rice code, a value b is required and value of b changes according to the average length of term occurring list if it is small, b should be small and if it is big, b should be big.

In bytewise coding, an integer is stored in an integral number of eight-bit blocks. Seven bits in each block are used to store a binary representation of integer x. The remaining bit is used to show whether an additional block follows or not. Bytewise compression schemes performance better than bitwise techniques. Bytewise compressed index's retrieval performance is better than uncompressed index's retrieval performance. That is true, even for a much smaller collection which uncompressed index is fits in memory [11]. The only disadvantage of bytewise compressed indices is that they are 30% larger than bitwise compressed indices.

In [11] there is hybrid of bitwise and bytewise compression techniques and evaluation and also some improvements on present techniques in here. All the techniques were explained related with inverted files because inverted files are selected for design.

2.3 Distributed Index Structures

The number and size of text document collections has grown at explosive rates in the Web. Simultaneously, there is also a rapid increase in the number of users and user queries submitted to information retrieval systems. As data volume and query processing loads increase, large information retrieval systems are turning to distributed and parallel storage and searching.

Basically there are two main approaches; first is building local index and second is building global index mechanism. Name of these techniques may differ in different sources. In [16] same techniques are named as term partitioning and document partitioning. In partitioned systems there is a broker processor which distributes query terms to appropriate index server and collects the results from index servers to combine and rank results for representation. The other members are index servers. They can keep a part of global index or they can keep full index for its local documents. It depends on strategy of index mechanism.

In [17] Tomasic et al. presents a work. They compare the four partitioning systems; disk index organization, host index organization, I/O bus index organization and System index organization. Their results show that host index organization is the most balanced use of resources. Since balance is a big problem in distributed index schemes host index system has better performance. In the host index organization, documents are partitioned into groups, one for each CPU. Within each partition they build inverted lists. The lists are then uniformly dispersed among the disk attached to the CPU.

In the [18] two basic index organizations (global vs local) are compared each other. The results indicate that a global index organization outperforms a local index organization. Similar work represents in [19] and also in that work global index structure outperforms local one because the global index allows the parallelization of the most time consuming phase of the algorithm - disk seeking. Further, the global index provides a high concurrent query service, which is particularly evidenced when the number of processors exceeds the average number of terms in query.

The work in [16] uses similar approaches: document partitioning index, term partitioning index. In document partitioning index, documents are distributed among index servers and each server generates its own inverted files for its local documents. In term partitioning index an inverted file is generated for all documents and inverted list (vocabulary and occurrence list) are distributed among index servers.

The document partitioning index affords easy parallelization of ranking. The search performed locally at each index server but system produces globally consistent document scores that can be merged directly at the broker using a relatively inexpensive multi-way merge algorithm. In the term partitioning local rankings are loose, because index servers have no information on the term weights which reside on the other index servers. In this way, a large number of documents from the top of the local ranking have to be sent to the broker that computes globally consistent document scores. This increases considerably network traffic and CPU workload at the broker. The detailed result of comparing these techniques can be found in [9].In this paper a hybrid partitioning index is proposed too. There are two types for this hybrid system; hybrid-or, hybrid-and. In hybrid-or partitioning system can operate either using a document or term partitioning but in hybrid-and system operate using both approach at the same time.

In [20] an algorithm is introduced according to both inverted files and suffix array and they used BSP (bulk synchronous parallel) model of computing algorithm. They used combination of two strategies to achieve balance firstly the term of the vocabulary are distributed uniformly at random onto the processor. For every query the broker machine chooses a server processor performing the ranking of documents.

This work also tries to solve problem with same two approaches local index and global index approach. In local index documents are assumed to be uniformly distributed onto the processors and for each processor local indices are generated. In the second approach, a single inverted file is generated by using all of the documents as a global index. Later on, index terms are uniformly distributed onto the processors with related inverted list. Uniformly distribution mechanism is a hashing algorithm that is performed by the broker machine. In global index approach there is an efficiency reduction problem: What is to be done when most frequently asked terms tend to be located in the same processor.

It is mentioned that solution of this problem can be static mapping or dynamic re-distribution. They tested both approaches and their results indicate that local inverted indices have better performance but also have high network traffic. Then they propose a new approach “composite inverted lists”. This approach depends on terms’ inverted list size. As a result of this new approach test results come near in computational speed to local index, and in communicational load to global index approach. In other words, hybrid methods outperform others in this area too.

2.4 Retrieval Modeling

One of main problems of information retrieval systems is to respond the user request properly. Information retrieval systems convert the semantic texts to index terms. This is because a lot of the semantics in a document or user request are lost. Thus it is no surprise that the documents that are retrieved as a response for a user request that is expressed as a set of keywords are frequently information retrieval relevant. There is one more reason that decreases degree of relevance. Most of the users have no training in properly forming their information retrieval queries.

Actually main issues of information retrieval systems are predicting which documents are relevant which are not. Such a decision usually depends on a ranking algorithm which attempts to establish a simple ordering of the documents that are retrieved. It means that ranking algorithms are at the core of the information retrieval systems. Ranking algorithms are changed with information retrieval models. There are tree basic retrieval model types and each of them originates from one classic model.

- Set Theoretic: Boolean Model;
- Algebraic: Vector Space Model;
- Probabilistic: Probabilistic Model;

In a conventional information retrieval system, the documents in the database remain relatively static while new queries are submitted to the system and it is called as “ad hoc retrieval.” There is one similar but different task: “filtering operation". In this operation user queries are relatively static while new documents come into the system. Filtering operation can be used while constructing a user profile. In this section firstly classic models will be examined and then alternative modeling paradigms for each type of model.

The classic models of information retrieval consider that each document consists of index terms. In general, index terms are mainly nouns because nouns have meaning by themselves and their semantic is easier to identify. [1] But full text search systems like web search engines consider all the distinct words in a document as an index terms. For document ranking process, describing terms importance is very significant and difficult task. Despite this difficulty, some properties of an index term which are easily measured and which are useful for evaluating the potential of term can be discover. Frequency of term is one of these properties. If there is a word which appears in just ten documents that are collected in the information retrieval system database, this word is valuable as an index term and value of this word is called as weight of index term.

Information retrieval systems usually assume that there is no correlation between index-terms, but this is not true. For example, the terms "Hardware" and "Computer" are used to index a given document which covers the area of "Computer hardware". Frequently, in this document, the occurrence of one of two words attracts the appearance of the other. Consequently their weights could reflect their correlation. However up to now it is not proved that this is advantage or disadvantage for retrieval raking performance.

2.4.1 Boolean Model

The Boolean model is a simple retrieval model, based on set theory and Boolean algebra. Boolean model provides a framework which is easy to understand by a common user of an information retrieval system. So the Boolean model received great attention in past years and was adopted by many of the early commercial bibliographic systems. [1] But Boolean model suffers from some big drawbacks.

Firstly it is not simple to translate an information need into Boolean expression. Secondly since its retrieval strategy depends on a binary decision criterion essentially this model is data retrieval model. It means that a document is relevant or non-relevant with query there is no any probability. Despite these problems, Boolean model provides a good starting point for those new to the field. [1]

The main advantages of the Boolean model are the clean formalism behind the model and its simplicity. The main disadvantage is that exact matching may lead to retrieval of too few or too many documents. The Boolean model is considered as mother of Set Theoretic Models. There are alternative set theoretic models and now two of them will be described:

2.4.1.1 Fuzzy Set Model

Fuzzy Set Theory [21] deals with the representation of classes whose boundaries are not well defined. The key idea is to associate a membership function with the elements of the class. This function takes values in the interval {0, 1} with 0 corresponding to no membership in the class and 1 corresponding to full membership. Membership values between 0 - 1 indicate marginal elements of the class. Thus, membership in a fuzzy set is a notion intrinsically gradual instead of abrupt as in conventional Boolean logic.

The procedure, to compute the documents relevant to a query is analogous to the procedure adopted by conventional Boolean model. The difference is that here we deal with fuzzy sets. "Fuzzy set models for information retrieval are not popular among the information retrieval community. "[1]

2.4.1.2 Extended Boolean Model

The extended Boolean model was introduced in 1983 by [22]. It is based on a critique of a basic assumption in Boolean logic. Actually this strategy allows one to combine Boolean query formulation with characteristic of vector model.

The extended Boolean model relaxes Boolean algebra interpreting Boolean operation in terms of algebraic distances. In this sense, it is really a hybrid model which includes properties of both the set theoretic models and algebraic models. Nevertheless because of its simplicity, we can classify as a set theoretic model. "The extended Boolean model has not been used extensively however; the model provides a neat framework and might reveal it useful in the future." [1]

2.4.2 Vector Model

As it is mentioned before, in Boolean model, result ranking is not possible since term weights are 0 or 1. Ranking documents is accomplished by assigning non-binary weights to index terms in queries and in documents. These term weights are ultimately used to compute the degree of similarity between each document stored in the system and the user query. By sorting the retrieved documents, we can reach results that better matches the user information need.

For vector model the term weight is positive and non-binary and not only document index terms but also query index terms are weighted. The main idea behind the vector model is evaluating the degree of similarity between documents and query text by using correlation.

This correlation can be quantified by the cosine of the angle between document vector and query text vector. In the ranking phase query vector has no effect because it is same for all documents. Results can be also limited by using a threshold. This operation solves the huge answer set size problem.

Vector model assumes that the retrieving problem as a clustering problem. The goal of the simple clustering problem algorithm might be to separate a collection of objects into two set. First set which is composed of objects related to the set A and the other which is composed of objects not related to the set A. In a clustering problem, two main issues have to be resolved. First, one needs to determine, the features which better describe the objects in the set A. Second, one needs to determine, the features which better distinguish the objects in the set A from remaining objects in our collection. Consequently the first features provide us intra-cluster similarity and second features provide inter-cluster dissimilarity.

Vector model uses these two set. The raw frequency of a term inside a document is used for intra-clustering similarity and it is called as (term frequency) **tf** factor. Furthermore inter-cluster dissimilarity is quantified by measuring the inverse of the frequency of a term among the documents in the collection. This is called as (inverse document frequency) **idf** factor. These definitions can be formulated as follows:

$f_{i,j} = fr_{i,j} / \max(fr_{l,j})$ (Equation 1)

$idf_i = \log (N/n_i)$

$f_{i,j}$: the normalized frequency of term k_i in document d_j

$fr_{i,j}$: raw frequency of term k_i in document d_j

$\max(fr_{l,j})$: is computed over all terms which are mentioned in the text of the document d_j

N: total number of documents in the system

n_i: number of documents in which the index term k_i appears

So we can show the term weighting algorithm as follows:

$w_{i,j} = f_{i,j} * \log (N/n_i)$ (Equation 2)

This term weighting strategies are called "tf-idf" schemes and variation of this formula is possible for example in 0 next formula is suggested for query term weighting.

$w_{i,q} = (0.5 + 0.5\, fr_{i,q} / \max(fr_{l,q})) * \log (N/n_i)$ (Equation 3)

The main advantages of the vector model are:

(a) Retrieval performance better than Boolean model because of term-weighting scheme

(b) Its partial matching strategy allows retrieval of documents that approximate the query condition

(c) Its cosine ranking formula sorts the document according to Their degree of similarity to the query

In vector model index term independency assumes as if disadvantage but in [1] it is reported that sometimes term dependency can decrease the overall performance.

A large variety of alternative ranking methods have been compared to the vector model but, the vector model is either superior or almost as good as the known alternatives. Since its simplicity, vector model is a popular retrieval model nowadays.

2.4.2.1 Generalized Vector Space Model

As specified before index terms are assumed independent. In the generalized vector space model, to index term vectors might be non-orthogonal. This means that index term vectors are not seen as orthogonal vectors which compose the basis of the space.

The main idea in the generalized vector space model is to introduce a set of pairwise orthogonal vectors associated with the set of "minterms" and to adopt this set of vectors as the basis for the subspace of interest. Minterm is a special term that is a set of binaries. Minterms show that pattern of term co-occurrences. For example the minterm m1 points to the documents are containing none of the index terms and m2 points to documents containing solely the first index term and the last minterm points to documents containing all index terms in the query. If a minterm has a document pointer it means that, this minterm is active. This implies that no more than N minterms can be active, where N is the number of documents in the collection.

Different from original vector model, in the generalized vector space model representation of documents and queries are expressed by minterm vectors. So the ranking results combine the standard term-document weights with correlation factors. But it is not proved that correlation factors effect on retrieval performance. Furthermore the cost of computing the ranking in the generalized model can be fairly high, with large collections because of the number of the active minterms. [1] As a result it is not clear that the framework of the generalized vector model provides a clear advantage in practical situations and also this model is more complex and computationally more expensive than the classic vector model. Since it introduces new ideas this scheme is still important.

2.4.2.2 Latent Semantic Indexing (LSI) Model

As is discussed Section 2.4.2, summarizing the contents of documents and queries through a set of index terms can lead to poor retrieval performance due to two factors. First many unrelated document might be included in the answer set. Second relevant documents which are not indexed by any of the query keywords are not retrieved. The main reason for two effects is the inherent vagueness associated with a retrieval process which is based on keywords sets.

The ideas of a text are more related to the concepts that are described in it than to the index terms that are used in its description. Thus the process of matching documents to a given query could be based on concept matching instead term matching. This would allow the retrieval of documents even when they are not indexed by query index terms. In other words, a document could be retrieved because it shares concepts with another document which is relevant to the given query. LSI is an approach introduced which addresses these topics.

The main idea in latent semantic indexing model is to map each document and query vector into a lower dimensional space which is associated with concepts. This is accomplished by mapping the index term vectors into this lower dimensional space. “The claim is that retrieval in the reduced space may be superior retrieval in the space of index terms.” [23] The latent semantic indexing model introduces an interesting conceptualization of the information retrieval problem based on the theory of singular value decomposition. [23] Thus it has its value as a new theoretical framework. Whether it is superior in practical situation with general collections remains to be verified.

2.4.2.3 Neural Network Model

According to neural network model, first query term nodes send a signal to the document term nodes and then the document term node generates a signal to the document nodes and this is the first phase. The signals which reach a document node are summed up. After a first phase, the activation level of the document node is associated to the document. In here signals that are summed by document nodes are normalized query-terms and normalized document-terms. Activation levels provide us ranking order like classic vector model. [24]

To improve the retrieval performance the network continues with the spreading activation process after the first phase. This modifies the initial vector ranking. [24] However there is no conclusive evidence that a neural network provides superior retrieval performance with general collection. A neural network presents an alternative modeling paradigm and also it allows retrieving documents which are not initially related to the query terms.

2.4.3 Probabilistic Model

Basics of probabilistic model depend on finding ideal answer set of documents. If the property of ideal answer set is known there is no problem but at the start we know only index terms whose semantics should be used to characterize these properties. So an effort has to be made at initially guessing what the properties are.

As a concept of this model an interaction is required with user for improving the probabilistic description of the ideal answer set. Firstly the user takes a look at the retrieved document and decides which ones are relevant and which ones are not. Then the system uses this information to refine the description of ideal answer set. According to this model a query is a subset of index terms. Similarity between a document dj and query q can be expressed as follows:
sim(dj , q) =P(R\dj)/P(R'\dj) (Equation 4)

R is the set of documents known to be relevant
R' is the complement of R
After some simplifying and transformation the expression becomes:

$\sum^{t} w_{iq} * w_{ij} * (\log(P(k_i\backslash R)/1-P(k_i\backslash R)) + \log(1-P(k_i\backslash R')/P(k_i\backslash R')))$ (Equation 5)

But the problem in here is at the beginning R is unknown. So a simplifying assumption has to be made.

$P(k_i\backslash R) = 0.5$ (Equation 6)
$P(k_i\backslash R') = n_i/N$

ni: is the number of the documents which contain the index term k_i

N: is the total number of document in the collection.

These are general assumptions [1]. With this initial effort we can retrieve the documents that contain query terms and provide an initial probabilistic ranking for them. After this phase we can improve the answer set by using the result. If we call X the retrieved and ranked document or subset of them and X_i is the subset of X that contain the index term. So the assumption is refined to:

$P(k_i \backslash R) = X_i/X$ (Equation 7)

$P(k_i \backslash R') = (n_i - X_i)/(N-X)$

This process can be repeated recursively. This process does not require human interaction. The main advantage of the probabilistic model is that documents are ranked in decreasing order of their probability of being relevant. Drawbacks of this model are:

(a) The need to guess the initial separation of documents into relevant or non-relevant sets.

(b) The fact that the method does not take into account the frequency with which index term occurs inside a document.

(c) The adoption of the independence assumption for index term.

2.4.3.1 Bayesian Networks

Bayesian networks are useful for information retrieval because they provide a clean formalism for combining distinct sources of evidence in support of the rank for a given document. This combination of distinct evidential sources can be used to improve retrieval performance. In [25] this claim has been demonstrated, and details can be found there.

There are various Bayesian network models. Inference network model and Belief network model are good examples for information retrieval systems, for more information about these models please look at [25-27].

CHAPTER 3

Design Principles of Medicoport Index and Retrieval Mechanism

In Chapter 2 types of index structures and retrieving models are introduced. Chapter 3 clarifies the scheme chosen for implementation and the rationale behind this decision. Next, functional requirements of MedicoPort's index and retrieval mechanism are listed. Finally, implementation details and system operation principles are explained.

3.1 Purpose and Scope

Search engine structure is separated into three main parts. First of them is the crawler module that brings the text database and it works on web side. Second is representation part that contains user interface, query engine etc. and works on user side. Last main part is indexing module that is the base module of a search engine on which the other parts are built. Therefore, performance of a search engine is directly related with index. Retrieving and ranking mechanism can be included in index structures. For these reasons, selection, implementation and construction of index structure is a significant task.

As mentioned in Section 2.1, some examples of such indices are suffix arrays, inverted files, and signature files. Each of them has pros and cons. Inverted files have been traditionally the most popular indexing technique used along the years. Although it is expected that MedicoPort users will run phrasal queries, an inverted file structure is decided to be constructed. Actually suffix arrays have better performance on phrase search, but its building cost is a compelling factor for choosing inverted indices. Because periodically the system index structure must be reconstructed and total cost of inverted index (construction + maintenance + search) is lower than suffix array total cost.

An index structure alone is not enough to answer user's queries. A result ranking subsystem is required to find the most relevant documents. Vector space model is used for this purpose. In the vector space model, documents and user queries are represented as a vector of term weights [28-31]. Simply term weights depend on frequency of that term in the text collection. If a term is observed very often in the text collection, the term weight is low. This strategy called tf/idf. The standard algorithm for ranking documents uses a set of accumulators. There is one accumulator for each document in the collection. And there is a set of inverted list, one for each term in the collection. Complexity of alternative ranking algorithms, whose performance can compete with vector space model, is a compelling factor for choosing this scheme in this thesis.

If it is considered that a query is a document, in other words as a vector, by finding similar vectors, system can retrieve most relevant documents to the user. One important drawback of vector model is that relevant documents which are not indexed by any of the query keywords are not retrieved.

Retrieval subsystem of MedicoPort solves this problem with a special module that is concept generate module. This module adds a new capability to vector space module. This capability would allow the retrieval of documents even when they are not indexed by query index terms. This is type of a concept search strategy.

Latent Semantic Indexing Model (section 2.4.2.2) is an alternative for this solution but this model is not verified now in practice and vector space model provides less complex solution to be implemented [1],[23]. Another improvement is primary index approach. Volume of the traditional index structure gradually grows and requires disk operations even if it is type inverted index. This reduces the performance of the search engine systems. Compression algorithm may solve this type of problems. But MedicoPort proposes a new solution which is primary index approach. MedicoPort keeps two types of indices one of them is traditional index, the other one is primary index. Type of primary index is an inverted index but it contains just special terms that are taken from UMLS Specialist Lexicon source.

Since MedicoPort performs on medical domain, search terms can be limited but limitation must be done professionally. For this reason, as a primary index terms the Specialist Lexicon is selected and description for UMLS Specialist Lexicon is given below:

Words and terms are selected for lexical coding from a variety of sources. Approximately 20,000 words from the UMLS Test Collection of MEDLINE abstracts together with words which appear both in the UMLS Metathesaurus and Dorland's Illustrated Medical Dictionary form the core of the words entered. In addition, an effort has been made to include words from the general English vocabulary. The 10,000 most frequent words listed in The American Heritage Word Frequency Book and the list of 2,000 words used in definitions in Longman's Dictionary of Contemporary English have also been coded.

Since the majority of the words selected for coding are nouns, an effort has been made to include verbs and adjectives by identifying verbs in current MEDLINE citation records, by using the Computer Usable Oxford Advanced Learner's Dictionary, and by identifying potential adjectives from Dorland's Illustrated Medical Dictionary using heuristics developed by McCray and Srinivasan 0.

A lexicon, recording information specific to individual lexical items, is necessarily a core component of any natural language processing system. The SPECIALIST lexicon has been developed to provide the lexical information needed for the SPECIALIST Natural Language Processing System. It is intended to be a general English lexicon that includes many biomedical terms.

Coverage includes both commonly occurring English words and biomedical vocabulary discovered in the NLM Test Collection and the UMLS Metathesaurus.

The lexicon entry for each word or term records the syntactic and morphological information. Syntactic information includes syntactic category (part of speech), and complementation patterns for verbs, adjectives and nouns, as well as positional and modification types for adjectives and adverbs. Inflectional morphology is indicated for those syntactic categories which inflect, and spelling variation is recorded for each lexical item known to exhibit such variation.

The lexicon consists of a set of lexical entries one entry for each spelling or set of spelling variants in a particular part of speech. Lexical items may be "multi-word" terms made up of other words if the multi-word term is determined to be a lexical item by its presence as a term in general English or medical dictionaries, or in medical thesauri such as MeSH. Expansions of generally used acronyms and abbreviations are also allowed as multi-word terms.

The lexical entry is a frame structure consisting of slots and fillers. Each entry is enclosed in braces ({...}) and identified by a unique entry number (EUI) recorded as the filler of the **entry=** slot. The EUI is a seven digit number preceded by the letter "E". The **cat=** slot indicates the part of speech of the entry and the **base=** slot indicates the base form of the entry. The base form is the uninflected citation form of the lexical item; the infinitive in the case of a verb; the singular in the case of a noun; and the positive in the case of an inflecting adjective or adverb. Optionally a **spelling_variants=** slot records spelling variants of the base form.The lexical entries for *anaesthetic* given below illustrate some of the features of a SPECIALIST lexical entries:

```
{base=anaesthetic
spelling_variant=anesthetic
entry=E0008769
cat=noun
variants=reg
}
{base=anaesthetic
spelling_variant=anesthetic
entry=E0008770
cat=adj
variants=inv
position=attrib
}
```

There are two entries for the base form *anaesthetic,* a noun entry and an adjective entry. The **variants=** slot contains a code indicating the inflectional morphology of each entry; the filler **reg** in the noun entry indicates that the noun *anaesthetic* is a count noun which undergoes regular English plural formation (*anaesthetics*); **inv** in the **variants=** slot of the adjective entry indicates that the adjective *anesthetic* does not form a comparative or superlative. The **position=** slot indicates that the adjective *anaesthetic* is attributive and appears after color adjectives in the normal adjective order. 0

The lexicon structure is not suitable with this form to construct primary index structure. This index will be static or semi static (if the UMLSKS is updated then primary index structure will be rebuilt). In the UMLS lexicon there are nearly 470.000 terms with duplications. After filtering operation we hold nearly 221.000 different lexicon terms in our primary index with its empty posting file. Posting file includes the lists of occurrences which are stored contiguously and it is an important part of inverted index structure. Since at the very beginning the system does not have any file, posting file is empty.

3.2 Assumptions and Dependencies

3.2.1 Assumptions

It is assumed that in the test phase index structure of MedicoPort has enough terms.

It is assumed that for the first release of the system index structure fits 512 MB of memory

It is assumed that system works on Pentium 4 1.8 GHz processor.

It is assumed that in the setup phase index must be initialized.

It is assumed that every system user is interested in medical information.

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

It is assumed that all the search operations shall be performed on main memory

It is assumed that retrieved documents are not hold in the local area.

3.2.2 Dependencies

MedicoPort's retrieving performance is affected by any possible UMLSKSS failures and any possible change in terms of use of UMLSKSS resources.

MedicoPort shall be run on a computer system which is registered to UMLSKSS with its IP and MAC addresses.

MedicoPort 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 of Indexing and Retrieving Subsystems

1. The system shall establish a connection to UMLSKSS.
2. 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.
3. If connection establishment is successful the system shall generate a notification message.
4. The query formulator module shall accept a new search topic from the user interface module.
5. The query formulator module shall convert query string to query terms.
6. The query formulator module shall send to primary index query terms to concept generator module.
7. The concept generator module shall obtain related concepts for each query term.
8. The concept generator module shall send retrieved related concept and query terms to the query formulator.
9. The query formulator shall send whole query terms to the search module
10. The search module shall convert the query term to query vector.
11. The search module shall find similar document vectors for answer set.
12. The search module shall prepare answer set.
13. The search module shall send answer set to user interface module.
14. The user interface module shall present answer set to the user.
15. Document processor shall acquire retrieved page from crawler module.
16. The document processor shall convert document string to index terms.
17. The system shall insert index terms to the index.
18. The system shall keep two index structures.

19. First structure shall keep UMLS Specialist Lexicon terms.
20. Second structure shall keep other retrieved terms.
21. First index structure shall keep phrase and single terms.
22. Index structure shall be updated monthly
23. System shall hold occurrence list for each index term.
24. Each occurrence list's item shall hold a document reference.
25. Each document reference shall hold link information to this document.
26. Each term shall have an idf value.
27. Each document reference shall have a vector length
28. The system shall compute vector length for each indexed document.

3.4 System Design and Implementation:

3.4.1 Decomposition Description

Index and Retrieving subsystem is developed with Document_Parser, Utility and User_Interface packages:

Document_Parser: This package covers the document processing features embedded in the system. Features include document parser, indexer and document vector computer.

Identification	:	Document_Parser
Type	:	Package
Purpose	:	Framework for document parsing, indexing and searching activities.
Function	:	Performs document processing functions, constructing inverted index and searching.

Utility: This package includes the utility features used by other packages.

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

User Interface: This package includes user interface module.

Identification	:	User Interface
Type	:	Package
Purpose	:	Framework that keeps user interface module.
Function	:	Interaction with system users. Takes user queries and represent the answer set to the users.

3.4.2 Dependency Description

Dependency description involves classes as design entities, and provides details related to classes of packages for Indexing and Retrieving subsystems by addressing identification, type, purpose, function, subordinates, dependencies, and resources design entity attributes 0.

Dependencies among classes and detailed description of these classes are given in Appendix C to Appendix R in form of class diagrams.

3.4.3 Detailed Description

MedicoPort Indexing and Retrieving subsystems details are presented in Figure 2. Crawler subsystem of MedicoPort, UMLSKSS and system users provide input to this system and system returns output to the system users. Subsystem modules will be described in the next subtitles.

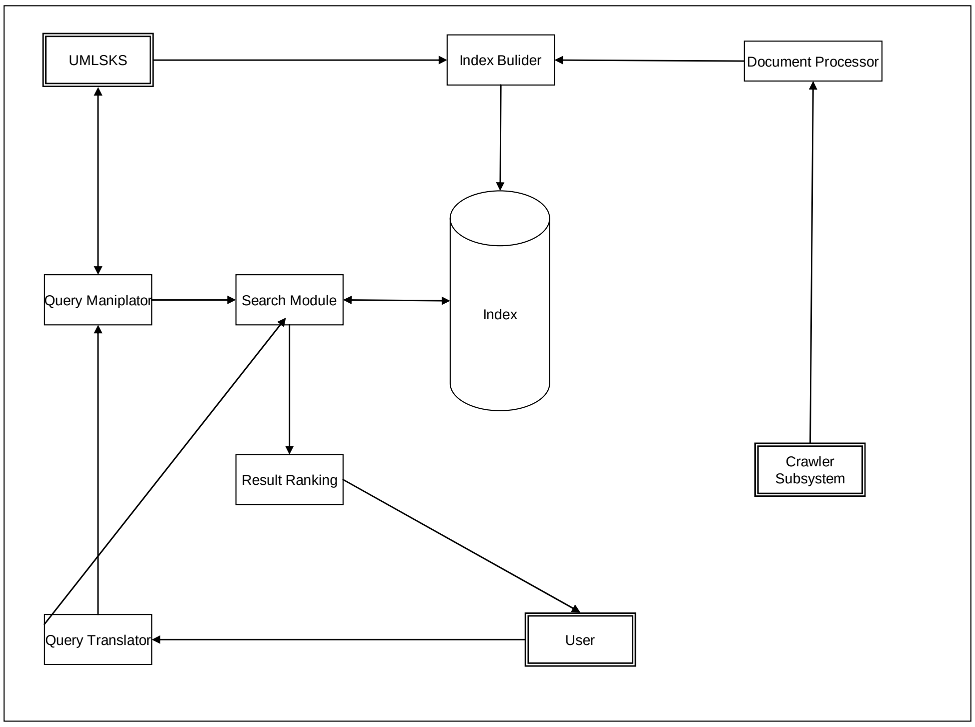


Figure 2: MedicoPort Indexing and Retrieving Subsystems

3.4.3.1 Query Translator Module

Query Translator module interacts with users. This module takes query string via user interface. System assumes that user queries are unformatted, so user query string must be converted to query terms. There are three main operations for translation. Suppose that system is given a query string of “institutions for breast cancer treatment”;

- All search phrases specified by the user are skimmed in order to find out if any UMLS SPECIALIST Lexicon terms or phrases exists, (‘breast’, ‘cancer’, ‘breast cancer’ are encountered)
- Rest of the query is filtered off stop words (‘for’ is omitted),
- Rest of the terms are stemmed (institutions => institute , treatment => treat),

After translation operation, one of the two output lists that includes lexicon terms is sent to Query Manipulator module. The other output list that includes query terms is sent to Searching module.

This operation is performed by PorterStemmer (Appendix L), TextStringDocument (Appendix I), QueryScreen (Appendix R)classes.

3.4.3.2 Query Manipulator Module

Query manipulator module takes a list that includes lexicon terms. For manipulation, this module uses UMLSKSS. Firstly this module establishes a connection to UMLSKSS.

UMLSKSS is free of charge and open to remote users. But registration is required. UMLSKS allows remote to users run queries on its resources via its web interface. For applications, UMLSKS provides Developer's API. It consists of a collection of interfaces for *JAVA* and some example applications. API makes use of Remote Method Invocation (java.rmi package) and allows remote applications to run queries on its resources. Responses to queries are returned in XML format. For further information on terms of use of UMLSKS, see [35].

Actually MedicoPort Crawler subsystem Lokman has a module for connection and retrieving information; UMLS Connector. So Query Manipulator also uses the same module for connection and retrieving related concepts for querying primary lexicon terms. For detailed description about UMLS Connector module see [3].

After the manipulation task, retrieved concepts are added to the list that is taken from Query Translator module. The result list is sent to the Searching module to prepare the answer set. Actually, result list is an array of ConceptInfo data structure. Each ConceptInfo entry of this list has one original query lexicon term and related terms with their weights.

This module uses class TextStringDocument(Appendix I), class InvertedIndex (Appendix K), LinkInfo(Appendix O) and UMLS Connector module [3].

3.4.3.3 Searching Module

Searching module prepares the answer set and sends the set to result raking module. Answer set consists of index terms and related document lists. To find the related document, Search module calculates vector length of query string. Vector length calculation can be defined as follows:

- Firstly all acquired terms are added as uniquely as to the new list with the number of times it occurs in the query.
- Each term is searched in primary index.
    - If term is not in the primary index, this term is searched in secondary index.
- Multiplication of “idf” value that was calculated before each term and “count” that is the number of times it occurs in the query, gives weight of the term.
- The weight of terms that are retrieved from UMLSKSS is multiplied with 1, 0.8, 0.5 or 0.3 according to their concept type.
- Total value of square of term’s weights gives vector length of query.

Then module searches the primary index to find the similar vectors. The most similar vector means the most related document.

This module uses classes, InvertedIndex (Appendix K), TextStringDocument (Appendix I), HashMapVector (Appendix F)

3.4.3.4 Result Ranking Module

Result Ranking module takes the answer set and reorganizes it. The answer set contains document references and document vector lengths. Answer set items’ is sorted in ascending order.

This module uses InvertedIndex (Appendix K), Result (Appendix P) and UserInterface (Appendix R)

3.4.3.5 Document Processor Module

Document Processor is fed by the Crawler with the fetched web documents. It performs several document processing operations on these which include stemming and parsing. After completion of these tasks, processed document is sent to Index Builder module.

This module uses classes HashMapVector (Appendix F), HtmlStreamParser (Appendix H), InvertedIndex (Appendix K).

3.4.3.6 Index Builder Module

At initialization of the system, Indexer retrieves UMLS Specialist Lexicon in order to construct a primary index. The primary index is updated via the same channel after new releases of UMLS are announced.

Second responsibility of this component is indexing the documents transmitted by Document Processor according to terms within them.

This module uses classes TermInfo (Appendix M), TermOccurence (Appendix N), and DocumentReference (Appendix D).

3.4.3.7 Index

Index covers two sub-indices. First of these is constructed using UMLS SPECIALIST Lexicon and called primary index. The latter structure, which keeps all terms appearing in document collection except the ones in primary index, is called secondary index.

It should be emphasized that Index keeps URL of the retrieved documents. Index also interacts with Crawler and Search Module. It serves as a topic name generator for the first and sends query results to the second.

Index uses classes InvertedIndex(Appendix K), LinkInfo(Appendix O), Weight(Appendix J), TermInfo (Appendix M), TermOccurence (Appendix N), and DocumentReference (Appendix D).

CHAPTER 4

Evaluation and Discussion of Indexing and Retrieval Subsystems

As declared in Section 1.1, this thesis focuses on two subsystems of MedicoPort search engine system; Indexing subsystem and Retrieval (Result Ranking) subsystem. Henceforth term System refers to these two subsystems. In previous section, detailed description of the system is depicted. In this section, to give better understanding, process details of system are explained.

4.1 Discussion of System

System index is assumed to be empty at initialization. As explained in Section 3.4.3.7 the system has two index structures. These indices are physically in the same format but content of indexes are different. Normally an index is filled by a system whenever a document is retrieved from the source (web in our case).

Since primary index contains medical terms, it must be initialized even the system has no documents. Initialization of primary index is performed by using medical terms obtained from UMLS Specialist Lexicon. Primary index has two main tasks:

- It is used as a database during search process like in other information retrieval systems. When a user specifies a query to system, system processes this query and converts it to search terms. Every search term is sought in the index by system. Result set for query is prepared and presented the user.

- It is used in document retrieval process by Lokman Crawler subsystem [3]. When invoked, Lokman takes a primary index term to retrieve related documents from web. For detailed information see [3].

Thanks to primary index, every search operation can be performed in main memory. This fact relatively increases system performance because index term number does not increase until a new version of UMLS Specialist Lexicon is released. For system index implementation, inverted index approach is chosen. Characteristic of inverted index is explained in Section 2.1.1. Construction cost of an inverted index is O(n) (n is the number of characters of the text).

System index must be reconstructed for every new version of UMLS Specialist Lexicon. This is another reason to select inverted index structure. Other index types that are described in Section 2.1 have the same or greater construction costs than inverted index.

Considering the facts given in Table 1, it can be claimed that Inverted index is suitable for word search and inexpensive to build and maintain. But Suffix array outperforms Inverted index on phrase search case. Medical domain requires more phrase search than word search. In this case, an information retrieval specialist needs such a model that is as effective as suffix array on phrase search and also as simple as inverted index. This can be possible if he/she has a set that includes all the possible search phrases. Without domain knowledge, it is not possible to obtain such a set because required phrase number is not limited.

UMLS Specialist Lexicon includes more than 220.000 unique terms [35]. This term set includes many medical phrases beside single medical words. This set (UMLS Specialist Lexicon) is reliable because it is constructed by medical specialists and using information from more than 14 medical sources 0,0. Therefore, the system uses this medical term set as primary index terms.

Next operation of system is to index retrieved documents from web. If the document includes a primary index term (it has to have at least one because it is retrieved by using this terms), this document is indexed in by the primary index but for sure this document also includes many other terms which are not members of primary index. Therefore secondary index structure is used. In other words, secondary index keeps an index using non-primary terms existing in retrieved documents.

Table 1: Comparison of Indexing Models.
(Req. = Requirement, n is character size of text, n' is size of new text, m is the length of query term, T is required time convert suffix tree to suffix array, N is the block size and M is available capacity of main memory)

Index Types / Costs & Req.	Inverted Files (index)	Signature Files	Suffix Tree	Suffix Array
Building	O(n)	O(n/N)	O(n)	O(n) + T
Insertion	O(n' log(n'/M))	O(log(n/N))	O(m)	O(log n)
Deletion	O(n)	O(log(n/N))		O(log n)
Space Req.	O(n^b) or 30-40% of text size	20% of text size	140-240% of text size	40-45% of text size
Disadvantages	Bad performance on phrase search	False Drop	Large Space Req.	Complex Structure
Advantages	Good performance on word search	Good performance for small sized DB	Good performance on phrase search	Good performance on phrase search

In this way, without a performance reduction, system can keep every retrieved term and gains the ability to respond to general queries of users. Actually this is not a main requirement but system must respond non-medical queries. Even though a query is non-medical, answer set of this query has medical documents. For example, if a user enters “computer” as a query term he/she will see probably documents that are related with not only term “computer” but also medical terms; in other words medical informatics related documents.

System database, because of the reasons that are mentioned in previous paragraphs, is a collection of high quality documents. It means that every retrieved document is related with medical domain. It is provided by Primary index and ontology based crawler approach.

System also takes care of user interaction. If a user is not satisfied with his/her search result, he/she can inform system via system user interface. If system takes such a feedback from user, it invokes Lokman Crawler with this query terms to retrieve the related documents. This operation degrades the quality of document database, but it increases the system response quality.

As a conclusion inverted index approach with specialist lexicon outperforms all the other indexing approaches. Therefore this scheme is chosen for implementation of system.

System's other task is to retrieve related documents and reorder them according to ascending relevance degrees. In a word, it is result ranking. In the Section 2.4 most popular retrieval model types are introduced. There are three classic retrieval models: Boolean, Probabilistic, Vector model. Boolean model is considered to be the weakest classic method. Main problem is the inability to recognize partial matches which frequently lead to poor performance.

There is a claim that the probabilistic model outperforms the vector model. Croft performed some experiments and suggested that the probabilistic method provides a better retrieval performance 0. However, experiments done afterwards by Salton and Buckley 0 indicated that the vector model is expected to outperform the probabilistic model with general collection. "This also seems to be the dominant thought researchers, practitioners, and the Web community, where the popularity of the vector model runs high." 0. The vector model is maybe the most popular model among the research community in information retrieval. "Most of this research revolved around the SMART retrieval system [38] developed at Cornell University" 0.

Probabilistic models are potentially open for further development but their complex structures make them difficult to construct. An experimental study shows that 0 probabilistic models outperform vector model. But derivation of vector model suggests more retrieval performance like Latent Semantic Indexing Model (Section 2.4.2.2).

The retrieval approach based on vector-space similarities can reach satisfactory recall rates only if the terms in the query are actually present in the relevant documents. For example, in English, the word ‘tank’ refers to both a large receptacle in which liquids or gases are stored (as in water tank) and also to a heavily armed and armored combat vehicle that moves on two endless metal chains called tracks. Polysemy negatively affects precision. Overall, synonymy and polysemy lead to a complex relation between terms and concepts that cannot be captured through simple matching. If a user enter “tank” as a query term he/she can meet undesirable result. To solve this problem, information retrieval systems must know the terms relation type.

Another main retrieving problem is: although a query may conceptually be very close to a given set of documents, its associated vector may be orthogonal or nearly orthogonal to those document vectors, simply because the authors of the document and the user have a different usage of language. Also last problem type can be solved by classification of terms. But linearly it seems to be too difficult

Latent semantic indexing is a statistical technique that attempts to estimate the hidden structure that generates terms given concepts. It uses a linear algebra technique known as singular value decomposition to discover the most important associative patterns between words and concepts.

This model theoretically is a solution to problems that are mentioned above. But implementation of this model seems to be too complex. For a specific domain, a model that is as effectives as latent semantic indexing model and also as simple as vector space model can be proposed. But it is only possible if an ontology based concept set is at disposal. UMLSKS provides such a concept set for medical domain. By using this database with vector space model a simple and efficient retrieval model can be constructed. The System (Indexing and Retrieval Subsystems) uses this idea.

After query processing operation query string turns to query term set. Some of the query terms belong to primary index, some of them do not. System assumes that every user is interested in medical information, so terms that are not included in primary index have secondary importance. The System takes into consideration term importance while answer set is prepared.

To prepare an answer set, The System calculates a vector length of query by using every query term and System compares this vector with document vector and finds the relevant documents.

Table 2: Term Relation Types and Weights

Relation	**Factor**
Exact Search Term	1
SYN (Synonymy)	0.8
PAR (Partial Relevance)	0.5
CXT (Contextual Relation)	0.3

To increase the relevance degree of answer set, System uses concept generation approach. Although UMLS defines various relations among concepts for a medical term, System interests three of them: synonymy (SYN), partially relevance (PAR) and contextual relation (CTX). As it is explained in Section 2.4.2 vector space model depends on term weights. Both query and document vectors are calculated according to their term weights. The System uses this scheme with a single difference. The related concepts of terms that are included in primary index are retrieved from UMLS and they are changed with related terms that are in the query and query vector is recalculated by System, but this time new terms' weight is reevaluated according to relation type. Table 2 reveals those reevaluation factors. Table 3 indicates a result of concept generation operation that is performed by System.

Table 3: Concept Generation Result for Term "Breast Cancer"

Query Term	**Concept List Obtained from UMLS**	**Relation**
Breast Cancer	Breast Carcinoma	SYN
	Cancer of the breast	SYN
	Mammary Carcinoma	SYN
	Carcinoma of Breast	SYN
	Malignant Tumor of Breast	PAR
	Malignant Neoplasm of Breast	PAR

To calculate vectors, Vector Space Model uses term frequency/inverse document frequency (tf/idf) formula. This formula gives term weight value. An experimental study shows that 0 variation of formula has different efficiency and most effective type of formula is selected by the System. This formula can be seen from (Equation 8).

$$W_{ij} = \log(t_{ij} + 1) * \log(N/n_j + 1) \qquad \text{(Equation 8)}$$

As a conclusion, System proposes new solutions to problems that are mentioned above. By using primary and secondary index structure System not only decreases the space requirement for main index but also holds high quality document database.

By using concept generation approach, retrieval performance of Vector Space Model is increased. Concept generation enables System to retrieve any document of interest even it is not indexed with the specified query term. In addition, this way, System gains the potential to overcome possible polysemy problems by assigning more points to a document with a search term and its ontological relatives than any document with only search term.

4.2 Information Retrieval System Evaluation Metrics

Information Systems uses two main techniques to evaluate their retrieval performance: recall and precision. To evaluate index performance system response time is another measurement. Since this is not an objective quantity, according to the information retrieval system purpose, whether this measurement might be a requirement or not. For implemented system response time is not a important requirement if this response time is smaller than human perception.

4.2.1 Recall

Recall is the retrieval proportion of a given relevant document (Nrs) to all relevant documents set (Nr)

$P = N_{rs} / N_r$ (Equation 9)

4.2.2 Precision

Precision (P) is defined as the proportion of a selected relevant documents (N_{rs}) to all retrieved document collection (N_s) (answer set generated for a given query)

$P = N_{rs} / N_s$ (Equation 10)

To give better understanding, recall and precision values are explained with an example in next paragraphs.

Assume that a set $R_{\mathbf{q}}$ containing the relevant documents for query q has been defined. Assume that the set $R_{\mathbf{q}}$ is composed of following documents:

$R_{\mathbf{q}} = \{D_3, D_5, D_8, D_{11}, D_{12}, D_{15}, D_{21}, D_{22}, D_{33}, D_{54}\}$

Thus, according to a group of specialist, there are ten documents which are relevant to query q. Consider now a new retrieval algorithm (system) which has just been designed. Assume that this algorithm returns, for query q, a ranking of the documents in the answer set as follows:

Table 4: Answer set of query q

No	Doc. Name	No	Doc. Name	No	Doc. Name
1.	D_{3} *	6.	D_{168}	11.	D_{8} *
2.	D_{31}	7.	D_{10}	12.	D_{66}
3.	D_{33} *	8.	D_{10}	13.	D_{30}
4.	D_{17}	9.	D_{15} *	14.	D_{84}
5.	D_{43}	10.	D_{23}	15.	D_{21} *

The documents that are relevant to the query q are marked with a star. If we examine this ranking, starting from the top document, the following points are observed. First, the document D_3 which is ranked as number 1 is relevant. Further, this document corresponds to 10% of the all the relevant documents in the set R_q. Thus system has a precision 100% and 10% recall. Second, the document D_{33} which is ranked as number 3 is the next relevant document. At is point, the system has precision 66% (two documents out of three are relevant) at 20% recall (two of the ten relevant document have been seen). After 15 result the system has precision 33% at 50% recall [1].

Usually retrieval algorithms are evaluated by running them for several distinct queries. Table 4 presents these distinct query topics for evaluation of the implemented system.

Table 5: Search Topics for System Retrieval Test

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

4.3 Evaluation of System

4.3.1 Test Bed

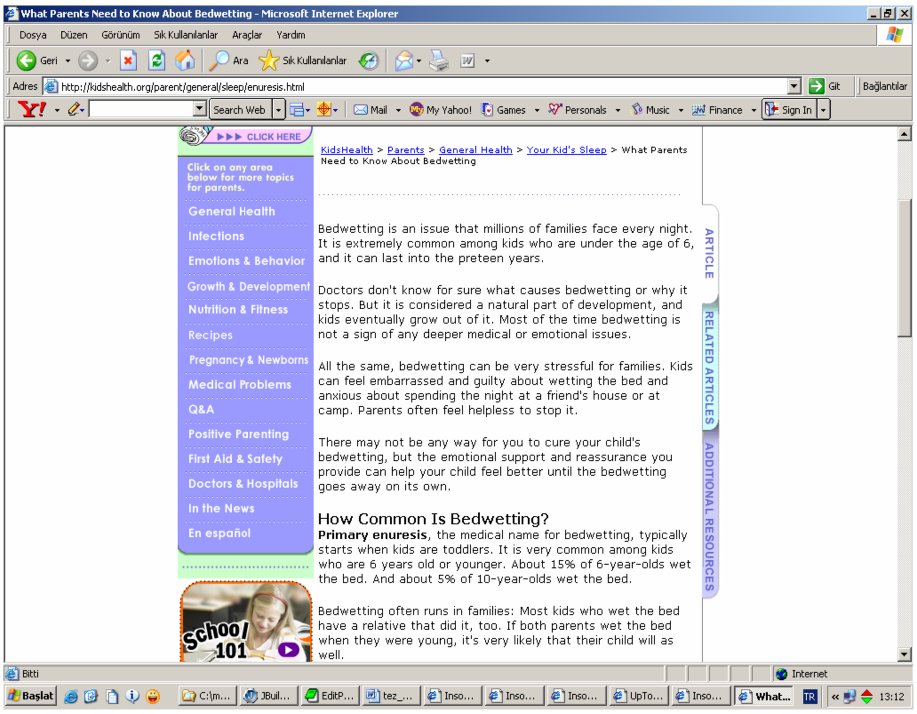
Tests for System are performed on a Pentium 4 2.8 GHz Processor PC with 768 MB of RAM. System needs internet connection and 256 Kb/s bandwidth is used test situation. 10 Topics are selected for test randomly from MedLine Plus web page 0(Table 4).

4.3.2 Empirical Results

Needed documents that are related with selected query terms for test were retrieved by Lokman Crawler [3]. Retrieved 1011 documents were indexed for this aim. System found 41705 unique terms from these documents. Normally System does not keep retrieved document in the disk but these 1011 documents were saved in order to shorten test time. Total size of documents was 53.1 MB. This document collection was indexed in 670 seconds. If document number is reduced to 513 documents with 31983 unique terms, required time for indexing was lessened 121 seconds.

Documents' relevance degrees (weight factors computed by tf/idf scheme) to queries were normalized to (0, 1) understand test results. If System finds a document that has the same content with query its relevance degree is equaled to 1. Irrelevant document's relevance is specified to 0 by System. This is the normalization mechanism.

To test the effect of concept generation on result set, queries were run on the System without using UMLSKS information. After the test results it is found out that from the same document collection, System can find 34 % more related document. For example, answer set of “breast cancer” without concept generation has 99 related results but with concept generation 133 related documents are found in same collection. An experimental threshold relevance degree “0.01” was chosen during tests. This value is the experimental result for answer sets which have more than 100 items. 133rd document has not included “breast cancer” but “breast carcinoma”. One of the aims of this study is to achieve similar results.


What Parents Need to Know About Bedwetting - Microsoft Internet Explorer
Dosya Düzen Görünüm Sık Kullanılanlar Araçlar Yardım
Geri Ara Sık Kullanılanlar
Adres http://kidshealth.org/parent/general/sleep/enuresis.html
Git Bağlantılar
Search Web Mail My Yahoo! Games Personals Music Finance Sign In
CLICK HERE
Click on any area below for more topics for parents.
General Health
Infections
Emotions & Behavior
Growth & Development
Nutrition & Fitness
Recipes
Pregnancy & Newborns
Medical Problems
Q&A
Positive Parenting
First Aid & Safety
Doctors & Hospitals
In the News
En español
School 101
ARTICLE
RELATED ARTICLES
ADDITIONAL RESOURCES
Bitti
Internet
Başlat C:\m... JBuil... EditP... tez_... Inso... Inso... Inso... UpTo... Inso... What...
13:12

KidsHealth > Parents > General Health > Your Kid's Sleep > What Parents Need to Know About Bedwetting

Bedwetting is an issue that millions of families face every night. It is extremely common among kids who are under the age of 6, and it can last into the preteen years.

Doctors don't know for sure what causes bedwetting or why it stops. But it is considered a natural part of development, and kids eventually grow out of it. Most of the time bedwetting is not a sign of any deeper medical or emotional issues.

All the same, bedwetting can be very stressful for families. Kids can feel embarrassed and guilty about wetting the bed and anxious about spending the night at a friend's house or at camp. Parents often feel helpless to stop it.

There may not be any way for you to cure your child's bedwetting, but the emotional support and reassurance you provide can help your child feel better until the bedwetting goes away on its own.

How Common Is Bedwetting?

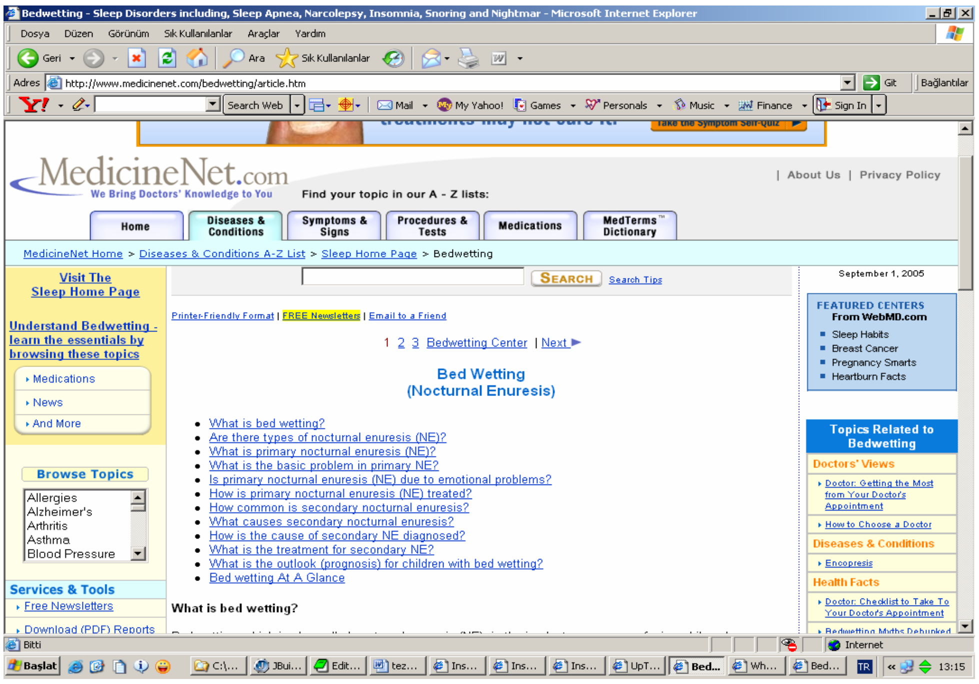
Primary enuresis, the medical name for bedwetting, typically starts when kids are toddlers. It is very common among kids who are 6 years old or younger. About 15% of 6-year-olds wet the bed. And about 5% of 10-year-olds wet the bed.

Bedwetting often runs in families: Most kids who wet the bed have a relative that did it, too. If both parents wet the bed when they were young, it's very likely that their child will as well.

Figure 3: “http://kidshealth.org/parent/general/sleep/enuresis.html” Number 4 hit from MedicoPort answer set generated for query “bedwetting”

To illustrate the positive effect of ontological information on retrieval results, screen shots from the answer set of query of “bedwetting” are presented in Figure 3 and Figure 4. In this example, it is clearly noticed that System assigns a higher relevance degree to documents which include more terms from ontology. Therefore a page with a variant of exact search phrase has the chance to get a place previous than a page with exact search phrase in the answer set.

System orders the result set ascending according to relevance degree. To depict a set of relevant documents, a limit relevance degree must be defined. As a minimum relevance degree "0.01" was selected for answer sets that have more result than 100. For small answer sets that have fewer entries than 100, relevance degree limit must be augmented. For instance term "insomnia" System finds 43 related documents and last result's relevance degree is 0.023.


Bedwetting - Sleep Disorders including, Sleep Apnea, Narcolepsy, Insomnia, Snoring and Nightmar - Microsoft Internet Explorer

Dosya Düzen Görünüm Sık Kullanılanlar Araçlar Yardım

Geri Ara Sık Kullanılanlar

Adres http://www.medicinenet.com/bedwetting/article.htm Git Bağlantılar

Search Web Mail My Yahoo! Games Personals Music Finance Sign In

MedicineNet.com We Bring Doctors' Knowledge to You

Find your topic in our A - Z lists:

| About Us | Privacy Policy

Home | Diseases & Conditions | Symptoms & Signs | Procedures & Tests | Medications | MedTerms™ Dictionary

MedicineNet Home > Diseases & Conditions A-Z List > Sleep Home Page > Bedwetting

Visit The Sleep Home Page

Understand Bedwetting - learn the essentials by browsing these topics

- Medications
- News
- And More

Browse Topics

Allergies
Alzheimer's
Arthritis
Asthma
Blood Pressure

Services & Tools

- Free Newsletters
- Download (PDF) Reports

SEARCH Search Tips

Printer-Friendly Format | FREE Newsletters | Email to a Friend

1 2 3 Bedwetting Center | Next

Bed Wetting
(Nocturnal Enuresis)

- What is bed wetting?
- Are there types of nocturnal enuresis (NE)?
- What is primary nocturnal enuresis (NE)?
- What is the basic problem in primary NE?
- Is primary nocturnal enuresis (NE) due to emotional problems?
- How is primary nocturnal enuresis (NE) treated?
- How common is secondary nocturnal enuresis?
- What causes secondary nocturnal enuresis?
- How is the cause of secondary NE diagnosed?
- What is the treatment for secondary NE?
- What is the outlook (prognosis) for children with bed wetting?
- Bed wetting At A Glance

What is bed wetting?

September 1, 2005

FEATURED CENTERS
From WebMD.com

- Sleep Habits
- Breast Cancer
- Pregnancy Smarts
- Heartburn Facts

Topics Related to Bedwetting

Doctors' Views

- Doctor: Getting the Most from Your Doctor's Appointment
- How to Choose a Doctor

Diseases & Conditions

- Encopresis

Health Facts

- Doctor: Checklist to Take To Your Doctor's Appointment
- Bedwetting Myths Debunked

Bitti Internet

Başlat C:\... JBui... Edit... tez... Ins... Ins... Ins... UpT... Bed... Wh... Bed... TR 13:15

Figure 4: "http://www.medicinenet.com/bedwetting/article.html" Number 17 hit from MedicoPort answer set generated for query "bedwetting"

System response time was measured smaller than 20 milliseconds without using concept generation function. But if system uses concept generation, system response time reaches nearly 6 second. The cause of this delay is connection to UMLSKSS. When concept generation was accomplished in local area, system response time was measured at maximum157 milliseconds.

System index space requirement was found 100-120 % of text size. Normally inverted index space requirement is 30-40 % of text size. Using JAVA as programming language and constructing two index structures are the reasons for the increase in memory requirement.

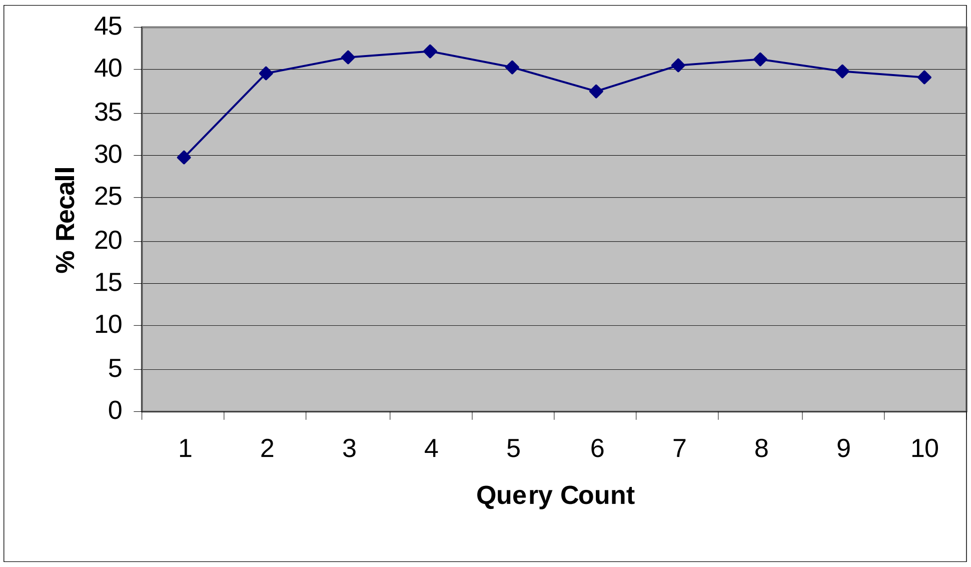


Figure 5: System Retrieval Recall after 100th Result

To evaluate the system retrieval performance, Recall and Precision measures are used. These measurements are inversely proportional. High precision is expected for a successful information retrieval system. System precision and recall results are as in Figure 5, Figure 6, Figure 7. Figure 5 indicates system recall value of ten query results. In Figure 6 system precision without concept generation is presented and Figure 7 shows system precision with concept generation.

4.3.3 Evaluation of Results

As seen in Figure 6, System finds 93 similar documents in first retrieved 100 documents without concept generation. This precision value is very high. This is the expected result because document collection that is used in tests is retrieved by a topical crawler. This means that test collection which is formed of 1011 documents is related with the search terms (Table 4). Precision value of Lokman Crawler is 67 % and more than 500 documents are related with search terms (Table 4) [3].

If System uses concept generation, system precision value increases to 96.3 % (Figure 7). Because, concept generation enables System to retrieve any document of interest even it is not indexed with the specified query term. For example, if a user enters “breast cancer” as search term, System can retrieve a document that does not contain term “breast cancer” but contains “breast carcinoma”. This outcome shows that concept generation solves one of the information retrieval problems and this is one of the aims of this thesis study.

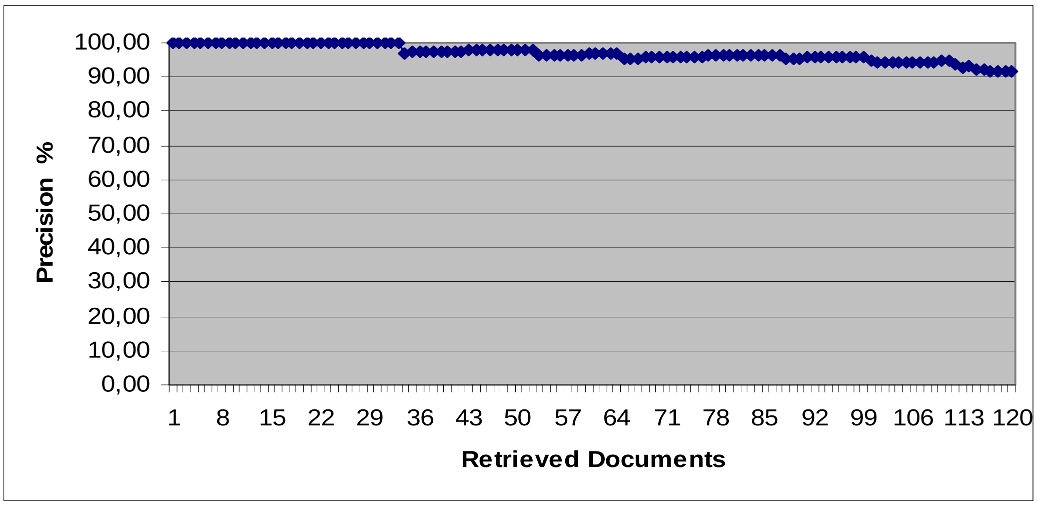


Figure 6: System Average Precision without Concept Generation

Average recall value of the implemented system is 39 % (Figure 5). The recall value of information retrieval systems is expected to be low with high precision. If system recall value is high, generally it brings about low precision value. Document collection size and quality of collection affects recall and precision values. System recall and precision values (Figure 5, Figure 6 and Figure 7) indicate that quality of our document collection is high. Probably if a larger test collection is used, this value decreases because of reduction of high quality document base of system.

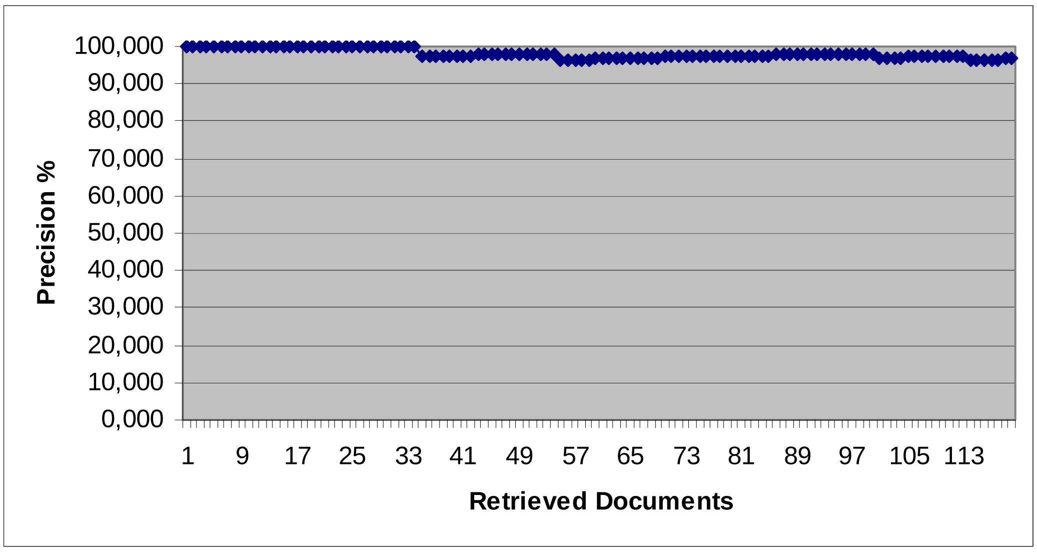


Figure 7: System Average Precision with Concept Generation

Space requirement of the system was found as 100-120 % of the text size. This ratio is too high for a system that uses inverted index. One reason of large space requirement is the programming language used. The System is implemented with JAVA programming language. If another low level programming language like C was preferred, this value would probably decrease. Another reason for high space requirement is the use of two index structures; primary index and secondary index. This scheme, in the beginning, causes large space requirement. But main operations of the system are performed on the primary index, so size of the index does not affect system performance.

System response time with concept generation is nearly 6 seconds and it seems to be high value for a web retrieval system. Reason of this delay is caused by UMLS connection. For instance, response time of Google search engine is approximately 0.013 second. If UMLSKS was installed in local area, response time would reduce to 157 milliseconds. This is an acceptable response time for an informational retrieval system. System response time is also directly related with bandwidth. Additionally, for the implemented system, quality of presented answer set bears more importance than rapid response time.

CHAPTER 5

Conclusion

Accessing the information seems easy thanks to recent technological improvements. Especially, internet provides any kind of information in a few seconds. But main problem of today is to find really needed information among this pile. Because of this problem, human needs machine help. But making machines understand human needs is another problem. Information retrieval systems try to find solutions to above problems. Although there are many improvements in this field, lots of problems are still waiting to be solved.

Main idea of an information retrieval system is to provide required information with less trouble. For this reason, search engines are the most popular information retrieval systems. Search engines, basically, are composed of three main parts: crawler that searches and retrieves required information; index that keeps the retrieved information in a suitable form; and retrieval module that prepares required information and presents to user.

Information needs of people change according to their interests. The need of domain specific search engine emerges to satisfy information requirements of people who interest in a specific area. This thesis suggests solutions for the indexing and result ranking problems of domain specific search engines. Problem of index that is built for domain specific search engine is constructing a quality document database. To solve this problem, use of ontological information as the knowledge base is proposed by this thesis study. Medicine is selected as domain and UMLS ontology is used as the knowledge base.

Since index structure is the basis for search engines, it serves to both crawler and retrieval module. It gives search terms to the crawler to search and retrieve related documents and also keeps those related documents for retrieval module. Existing index of search engines are not responsible to serve to crawler this way. Because of this duty, index in suggested system bears another responsibility.

System divides index structure to primary and secondary parts. The primary index contains medical terms and phrases that are acquired from UMLS. Thanks to primary index, size of index structure is limited and system operations can be performed in memory. Primary index structure can be considered as an index of book. In this index only required term and phrases are kept.

Secondary index structure prevents system to loose information for retrieved documents. To use the implemented system as a general search engine, this index keeps other terms that are not indexed by primary index. Although this scheme increases space requirement, it offers relative performance by allowing system keep secondary index in disk instead of memory.

Retrieval module of implemented system uses information obtained from UMLSKSS. System finds primary index terms and phrases in user queries and generates the related concept set for them from UMLSKSS. Search operation is performed with both original user query and its alternatives formed of generated concept set. With such a novel approach, system gains ability to retrieve any document of interest even it is not indexed with the specified query term. Performed test results also validate this fact. In addition, it is discovered that concept generation increased precision value of system from 93% to 96.3% for the test collection and test queries.

System is implemented with JAVA. Use of JAVA in system implementation and communication overhead caused by remote UMLS connection increase system response time. But for the implemented system, quality of presented answer set bears more importance than rapid response time.

5.1 Future Work

If UMLSKSS was installed in local area network, system response time decreases from 6000 milliseconds to 157 milliseconds (Section 4.3.3). UMLS allows installation of knowledge server in local area network. System overall performance needs to be tested if local installation of ontology is provided.

This system can be used as the backbone of a knowledge retrieval system. If a profiling database is constituted, system precision value will increase. System with profiling database can retrieve results which are more personalized, more refined and more qualified.

Data structures of JAVA are used during implementation. JAVA has no trie data structure that is more suitable for index mechanisms. If system index structure was implemented with trie data structure, space requirement of the system would decrease.

5.2 Contribution

Suggested indexing and retrieval subsystems are implemented as the subsystem of MedicoPort Project. As part of a search engine, function of index and retrieval module is increasing the quality of search engine retrieval performance. The other part of the system is Lokman Crawler.

As test results indicated, Lokman reaches a harvest rate of 67 % excluding D = 1 [3] pages. 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.

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 increased system performance from 93% to 96.3% (Section 4.3.2). This result shows that project has reached its goal.

REFERENCES

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

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

[3] Kayışoğlu A., Lokman: A Medical Ontology Based Topical Web Crawler, METU, M.S. Thesis, September 2005, Ankara, Turkey

[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, M.S. Thesis, Atılım University, June 2005, Ankara, Türkiye.

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

[8] Rmıt, J. Z., Moffat, A., Ramamohanarao, K., Inverted Files Versus Signature Files for Text Indexing, 1998, ACM Press New York, NY, USA.

[9] Qui, J., Shao, F., Zatsman M., Shanmugasundara, J., Index Structure for Querying Deep Web, Cornell University, 2003

[10] Witten, I., H., Moffat, A., Bell, T. C. Managing Gigabytes: Compressing and Indexing Documents and Images, Morgan Kaufmann, San Francisco, Second edition, 1999.

[11] Scholer, F., Williams, H. E., Yiannis, J., and Zobel, J., Compression of inverted indices for fast query evaluation, In Beaulieu M., Baeza- Yates R., S. Myaeng H., and Jarvelin K. (eds), Proc. 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Tampere, Finland, Aug. 2002. ACM Press, New York.

[12] Elias, P., Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory, Volume IT-21(2):194–203, Mar. 1975.

[13] Golomb, S. W., Run-length encodings, IEEE Transactions on Information Theory, Volume IT–12(3):399–401, July 1966.

[14] Ziviani, N., de Moura, E., Navarro, G., Baeza-Yates, R., Compression A key for next-generation text retrieval systems, IEEE Computer, Volume 33(11):37–44, Nov. 2000.

[15] Ngoc Anh, V., Moffat, A.: Index Compression using Fixed Binary Codewords, Proceedings of the Fifteenth Australasian Database Conference, (ADC2004), Dunedin.

[16] Zomaya, A.Y., Parallel and Distributed Computing Handbook, McGraw Hill,.New York, 1996.

[17] Tomasic, A. and Garcia-Molina, H., Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In Proceedings of the Second International Conference on Parallel and Distributed Information Systems, pages 8–17, 1993, San Diego, California, U.S.A.

[18] Ribeiro-Neto, B.,. and Barbosa R. A., Query performance for tightly coupled distributed digital libraries, In Proceedings of the third ACM Conference on Digital Libraries, pages 182–190, 1998.

[19] Badue, C., Baeza-Yates, R., Riberio-Neto, B., Ziviani, N, Distributed query processing using partitioned inverted files. In 8th Symposium on String Processing and Information Retrieval (SPIRE'01), Nov. 2001

[20] Laguna de, S.R., Using partitioned inverted files, In Proceedings of the Eighth String Processing and Information Retrieval Symposium, pages 10–20, 2001. IEEE Computer Society.

[21] Zadeh, L. A., Fuzzy sets, Information and Control, vol. 8, pp. 338--353, 1965.

[22] Salton, G., Fox, E A., Wu, H., Extended Boolean Information Retrieval, Communication of the ACM, Volume 26(11) pp: 1022-1036, Nov. 1983.

[23] Furnas, G. W., Deerwester, S., Dumais, S. T., Landauer, T.K., Harhman, R.A., Streeter, L.A: and Loachbaum, K.E., Information retrieval using a singular value decomposition model of latent semantic structure, ACM SIGIR Conference on Research and Development in Information Retrieval 1988

[24] Wilkinson, R., and Hingston, P., Using the cosine measure in a neural network for document retrieval, ACM SIGIR Conference on Research and Development in Information Retrieval, 1991

[25] Hurtle, H. and Croft, W.B., Evaluation of Inference network-based retrieval model, ACM Transaction on Information System, Volume. 9(3) pp. 187-222, July 1991,

[26] Hurtle, H. and Croft, W.B., Inference network for document retrieval, ACM SIGIR Conference on Research and Development in Information Retrieval, 1990

[27] Riberio-Neto, B., and Muntz, R., A Belief network model for information retrieval, ACM SIGIR Conference on Research and Development in Information Retrieval, 1996

[28] Luk, R.W.P., Kwok, K.L., Comparison of Chinese Document Indexing Strategies and Retrieval Models, ACM Transactions on Asian Language Information Processing (TALIP) Volume 1(3) pp: 225 – 268, September 2002

[29] Park, A. F., Spectral Based Information Retrieval, Ph.D. Thesis, University of Melbourne, Australia, December 2003

[30] Wainwrigth, J., Text Mining Lecture Notes, March 2003

[31] Rijsbergen C J van: Information Retrieval, Butterworth, London, 1979.

[32] Browne, A. C., McCray, A. T., Lister, S. S., The Specialist Lexicon Technical Report, Hill National Center for Biomedical Communications National Library of Medicine Bethesda, Maryland REVISED: JUNE 2000

[33] Tilley C. B., Willis Jan, UMLS Basics Presentation (May 2004)

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

[35] UMLS Knowledge Source Server, Retrieved: August 24, 2005, from http://umlsks.nlm.nih.gov/kss/servlet/Turbine/template/

[36] Croft, W. B. and Harper, D. J., Using probabilistic models of document retrieval without relevance information. Journal of Documentation, Volume 35 pp:285-295, 1979.

[37] Salton, G. and Buckley, C., Term weighting approaches in automatic retrieval, Information Processing & Management, Volume 24(5), 1988

[38] Salton G., The SMART Retrieval System--Experiments in Automatic Document Processing, NJ: Prentice Hall, 1971.

[39] Ribeiro-Neto, B., Ziviani, N., Moura, E., and Neuber, M.. Efficient Distributed Algorithms to Build Inverted Files. Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR Forum, pp. 105-112, August 1999.

[40] Marin M.; Index Structures for Distributed Text Databases; JCS&T, Vol. 4(1), April 2004

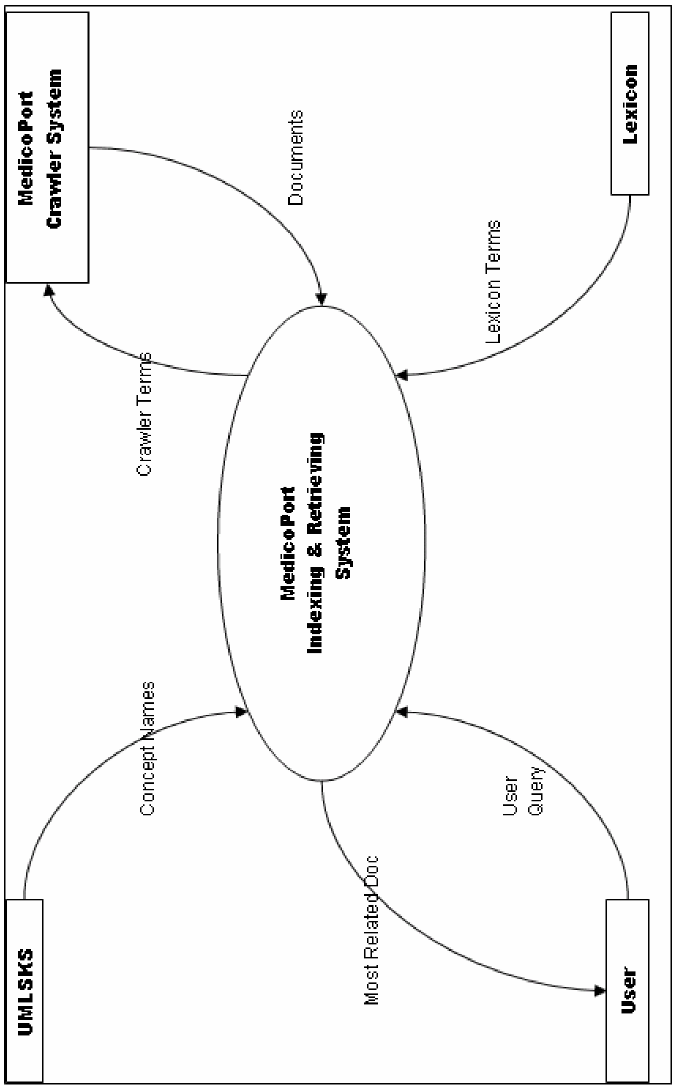
[41] Salton G., Lesk M. E.,Computer evaluation of indexing and text processing, Journal of the ACM, Volume: 15(1), January, 1968.

[42] Salton G. and McGill M. J. Introduction to Modern Information Retrieval, McGraw Hill, 1983

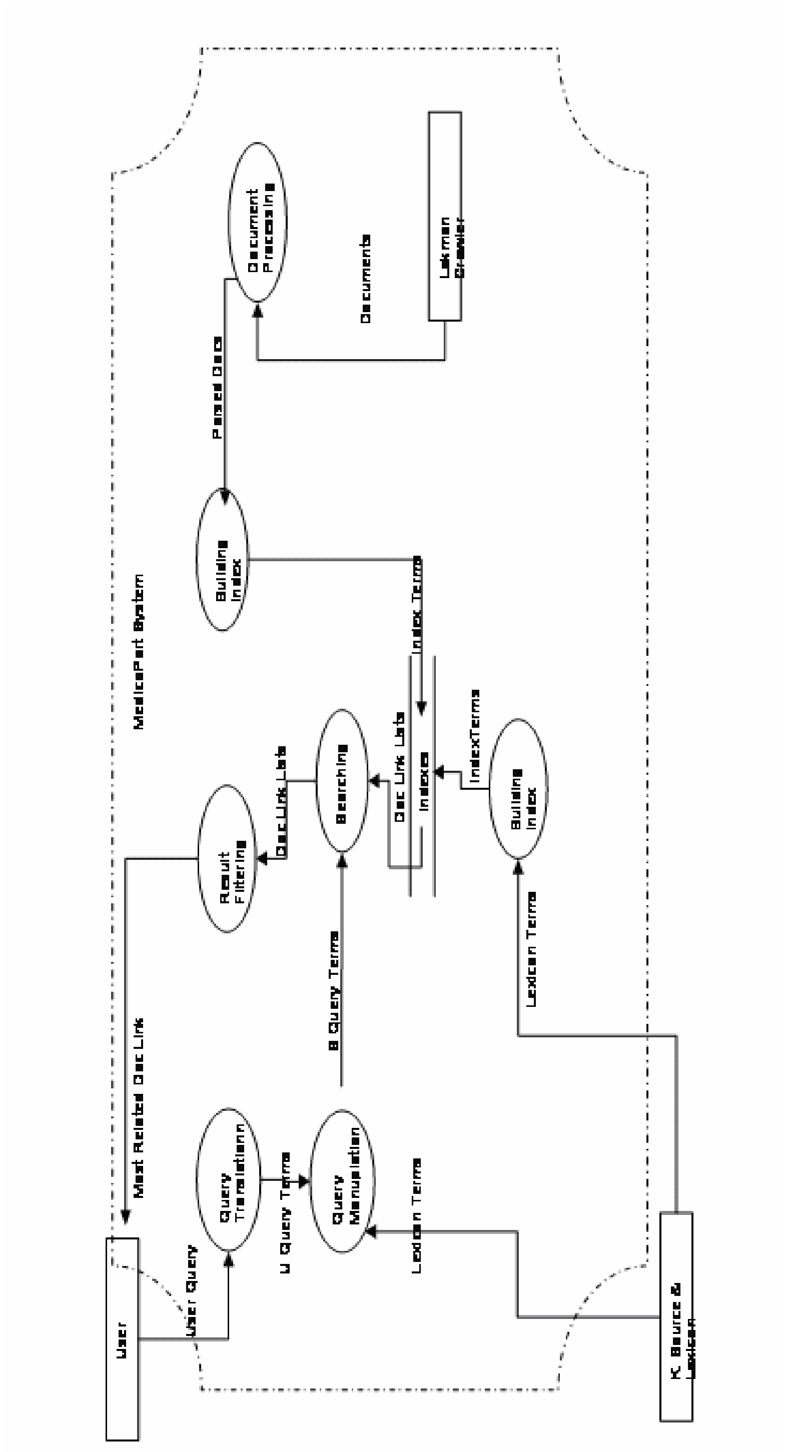
[43] MedLine All Health Topics Page, Retrieved: 25th August 2005, from http://www.nlm.nih.gov/medlineplus/ all_healthtopics.html,

APPENDICES

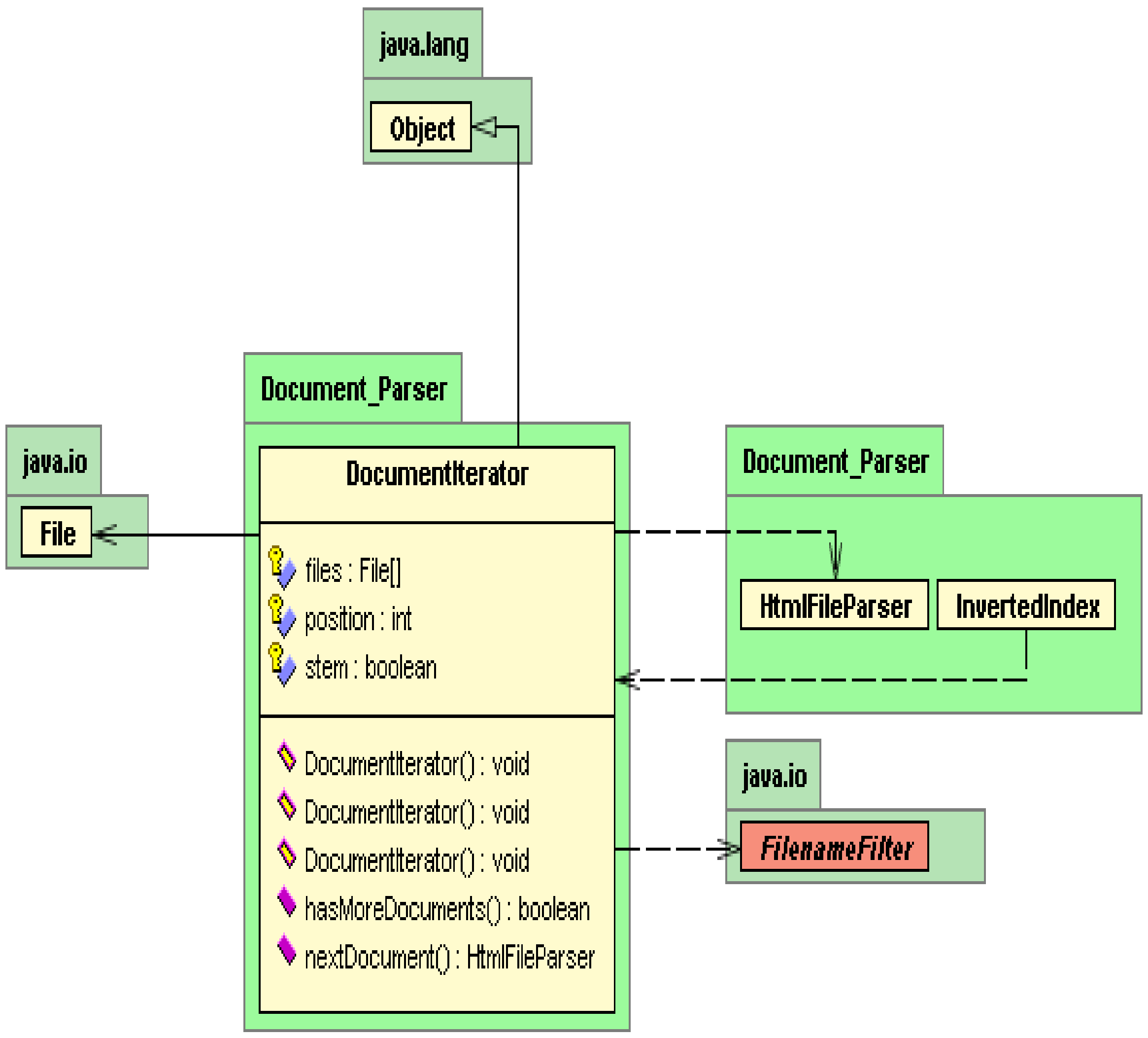
Appendix A: MedicoPort Index and Retrieving System Level 0 Dataflow Diagram



Appendix B: MedicoPort Index and Retrieving System Level 1 Dataflow Diagram



Appendix C: Package: Document_Parser; Class: DocumentIterator Class Diagram



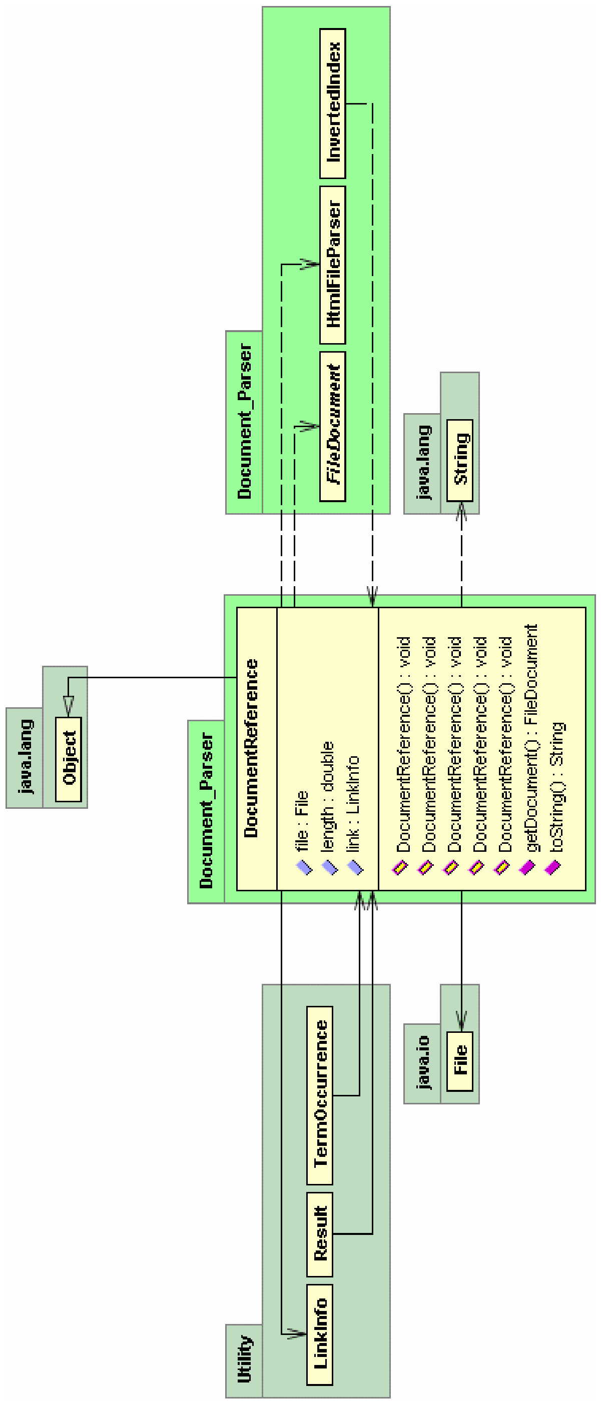
Detailed Description of Class DocumentIterator:

Identification	:	DocımentIterator
Type	:	Class
Superclass	:	Object
Purpose	:	An object for iterating over a set of documents in a directory.
Function	:	Iterates next document for indexing procedure

Subordinates	:	nextDocument()
		hasMoreDocuments()

Dependencies	:	Document_Parser.HtmlFileParser Document_Parser.InvertedIndex
Resources	:	java.io java.lang java.util;

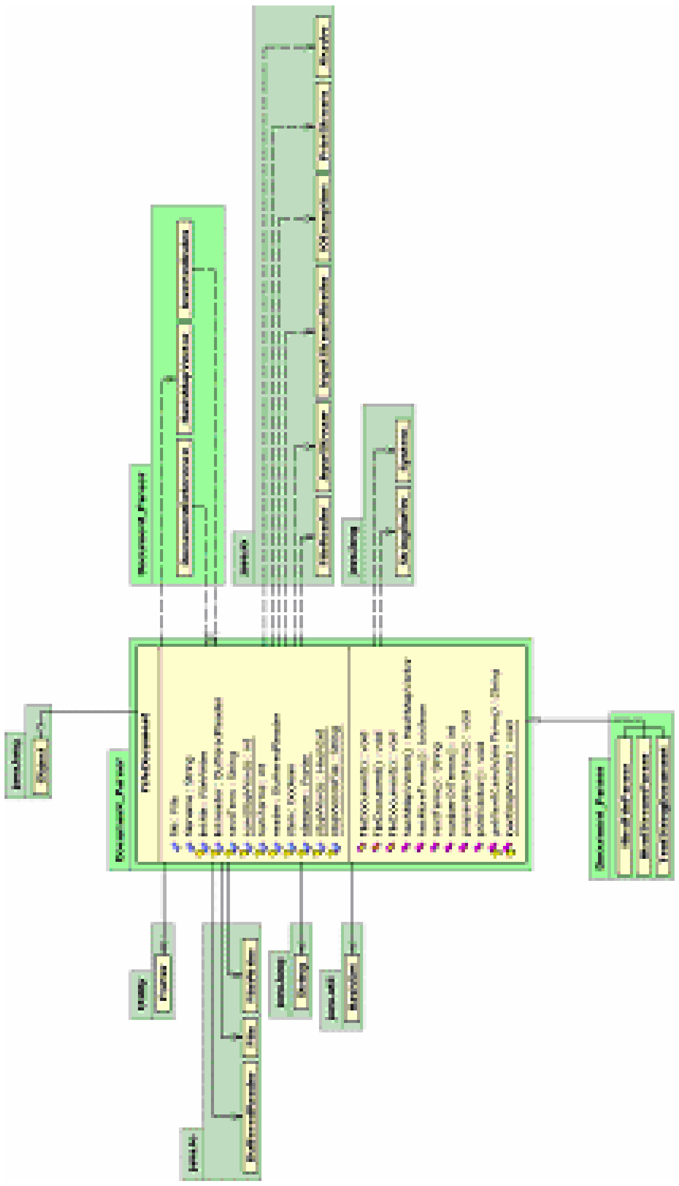
Appendix D: Package: Document_Parser; Class: DocumentReference Class Diagram



Detailed Description of Class DocumentReference:

Identification	:	DocımentReference
Type	:	Class
Superclass	:	Object
Purpose	:	This class hold document information
Function	:	Get the full Document for this Document reference by recreating it with the given docType and stemming
Subordinates	:	getDocument(short docType, boolean stem)
Dependencies	:	Document_Parser.FileDocument Document_Parser.HtmlFileParser Document_Parser.InvertedIndex Utility.LinkInfo Utility.Result Utility.TermOccurence
Resources	:	java.io Utility

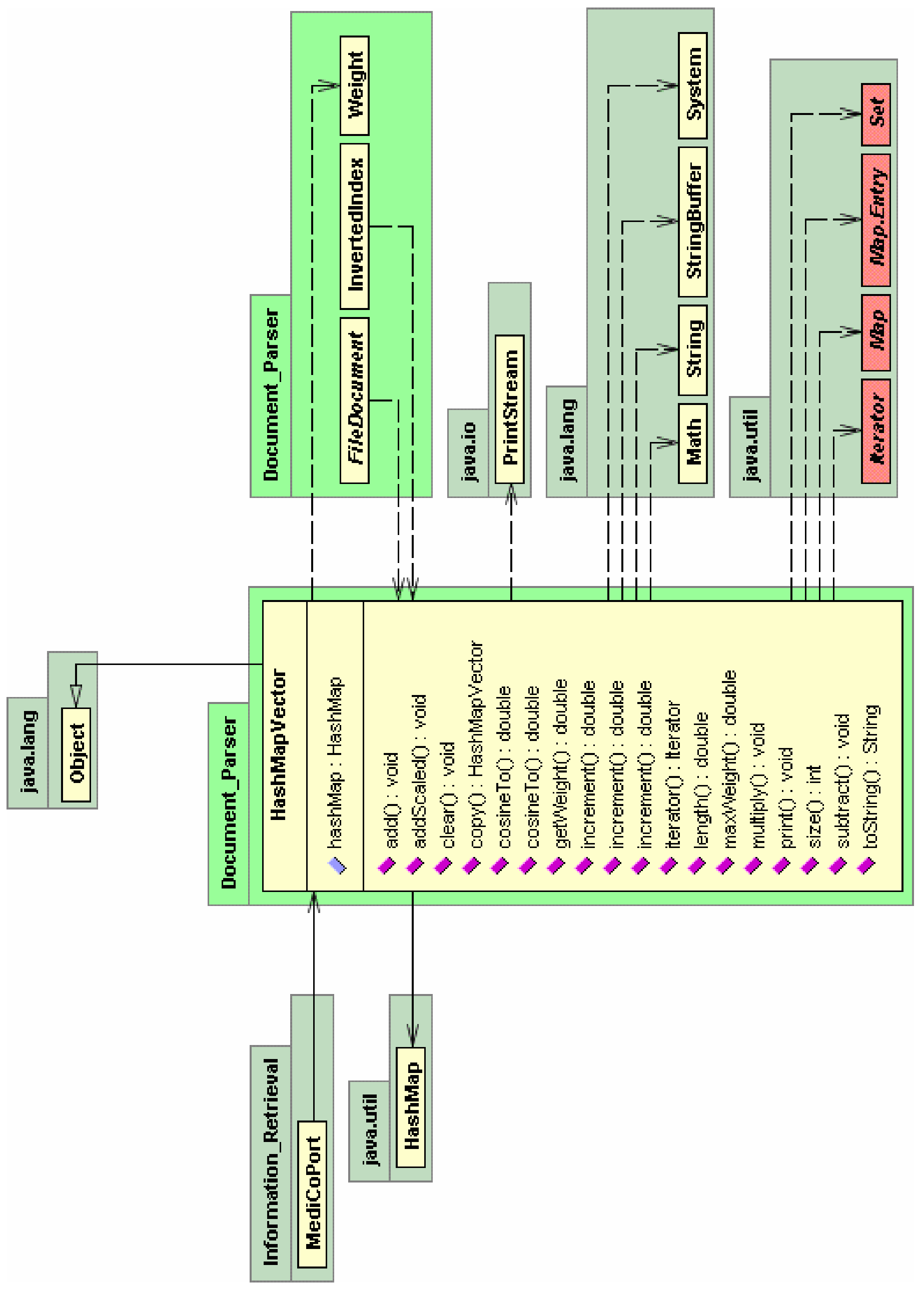
Appendix E: Package: Document_Parser; Class: FileDocument Class Diagram



Detailed Description of Class FileDocument:

Identification	:	FileDocument
Type	:	Class
Superclass	:	Object
Purpose	:	Base class for for HtmlFileParser and HtmlStreamParse
Function	:	Convert the text files to the a term list
Subordinates	:	int numberOfTerms()
		String nextTerm()
		boolean hasMoreTerms()
		void prepareNextTerm()
		void printVector()
		abstract String getNextCandidateTerm()
		static void loadStopWords()
Dependencies	:	Document_Parser.DocumentReference
		Document_Parser.HashMapVector
		Document_Parser.InvertedIndex
		Utility.Porter
Resources	:	java.io
		Utility.Porter
		java.util

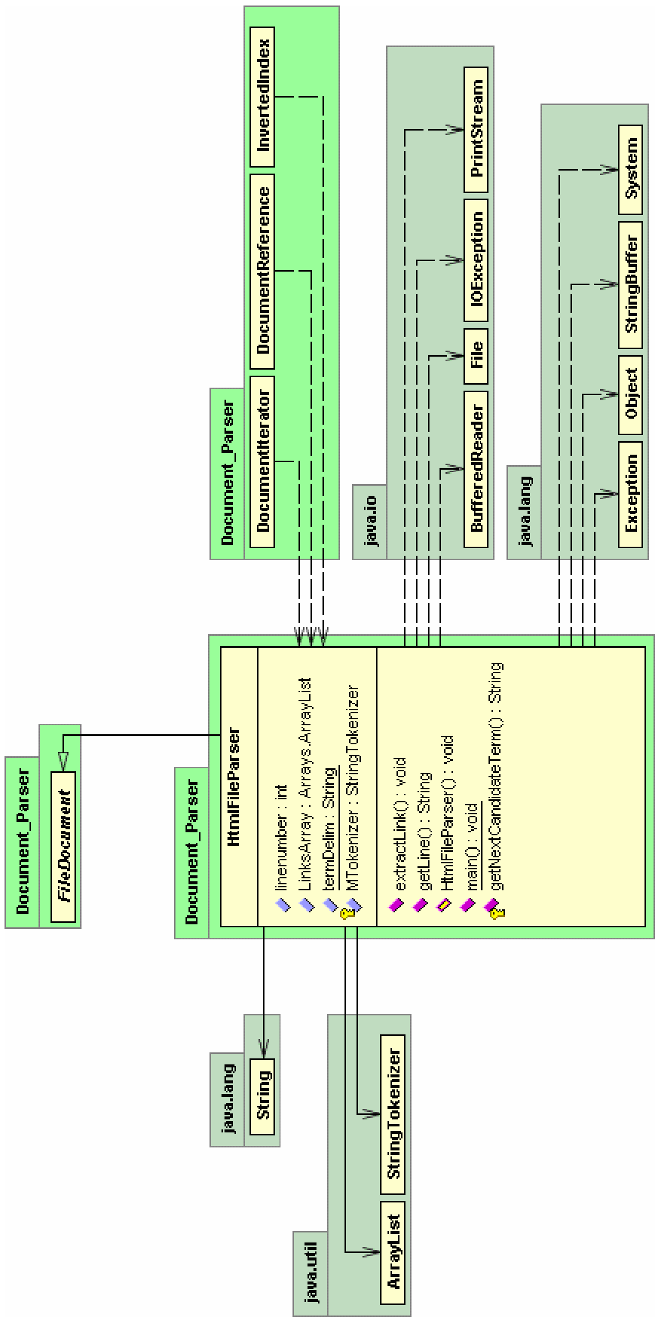
Appendix F: Package: Document_Parser; Class: HashMapVector Class Diagram



Detailed Description of Class HashMapVector:

Identification	:	HashMapVector
Type	:	Class
Superclass	:	Object
Purpose	:	A data structure for a term vector for a document stored as a HashMap
Function	:	This class maps terms to Weight's that store the weight of that term in the document.
Subordinates	:	Iterator iterator() int size() void clear() double increment(String term,double amount) double getWeight(java.lang.String term) void add(HashMapVector vector) void addScaled(HashMapVector vector,double scalingFactor) void subtract(Document_Parser.HashMapVector vector) void multiply(double factor) Document_Parser.HashMapVector copy() double maxWeight() void print() double cosineTo(Document_Parser.HashMapVector otherVector) double length()
Dependencies	:	Document_Parser.InvertedIndex Document_Parser.Weigth
Resources	:	java.util

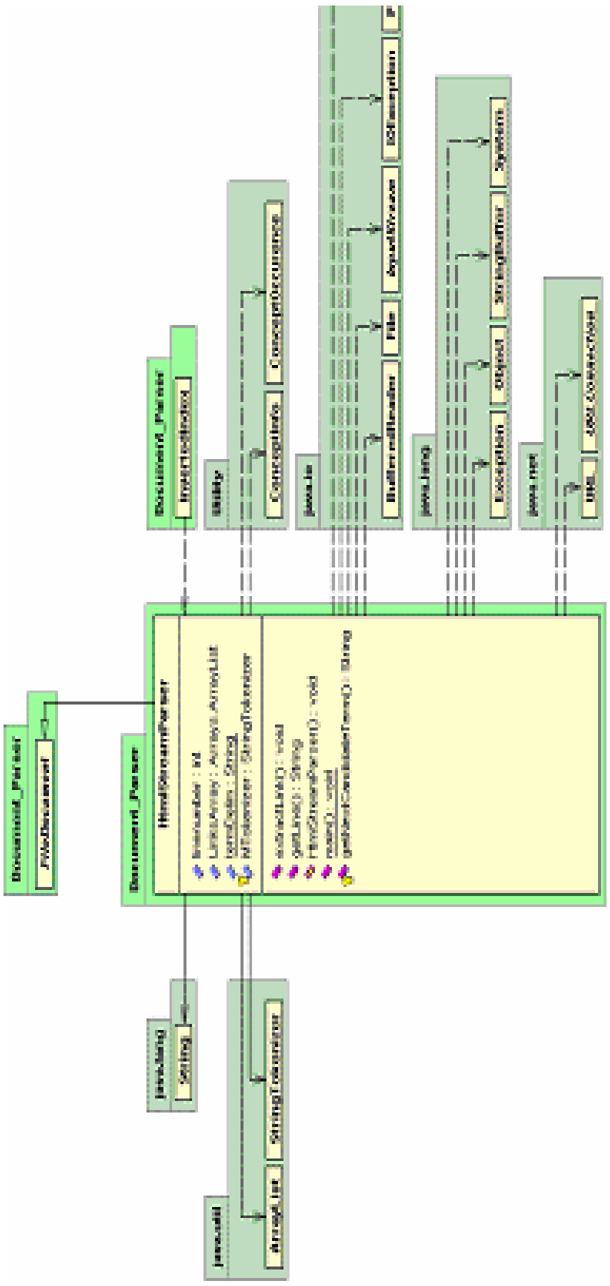
Appendix G: Package: Document_Parser; Class: HtmlFileParser Class Diagram



Detailed Description of Class HtmlFileParser:

Identification	:	HtmlFileParser
Type	:	Class
Superclass	:	FileDocument
Purpose	:	An HTML file document where HTML commands are removed from the term stream.
Function	:	To include HTML terms, just create a Text File Document from the HTML file
Subordinates	:	String getNextCandidateTerm()
Dependencies	:	Document_Parser.InvertedIndex
Resources	:	java.io java.util

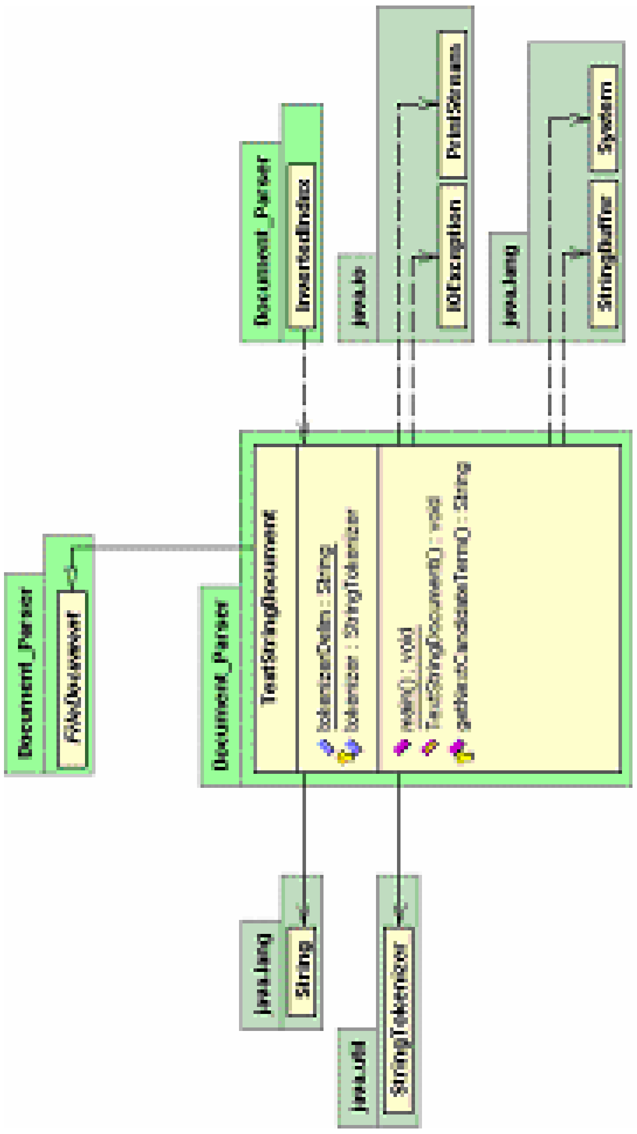
Appendix H: Package: Document_Parser; Class: HtmlStreamParser Class Diagram



Detailed Description of Class HtmlStreamParser:

Identification	:	HtmlStreamParser
Type	:	Class
Superclass	:	FileDocument
Purpose	:	An HTML stream where HTML commands are removed from the term stream.
Function	:	To include HTML terms, just create a Text File Document from the HTML file
Subordinates	:	String getNextCandidateTerm() String getLine()
Dependencies	:	Document_Parser.InvertedIndex Utility.ConceptInfo Utility.ConceptOccurence
Resources	:	java.io java.util

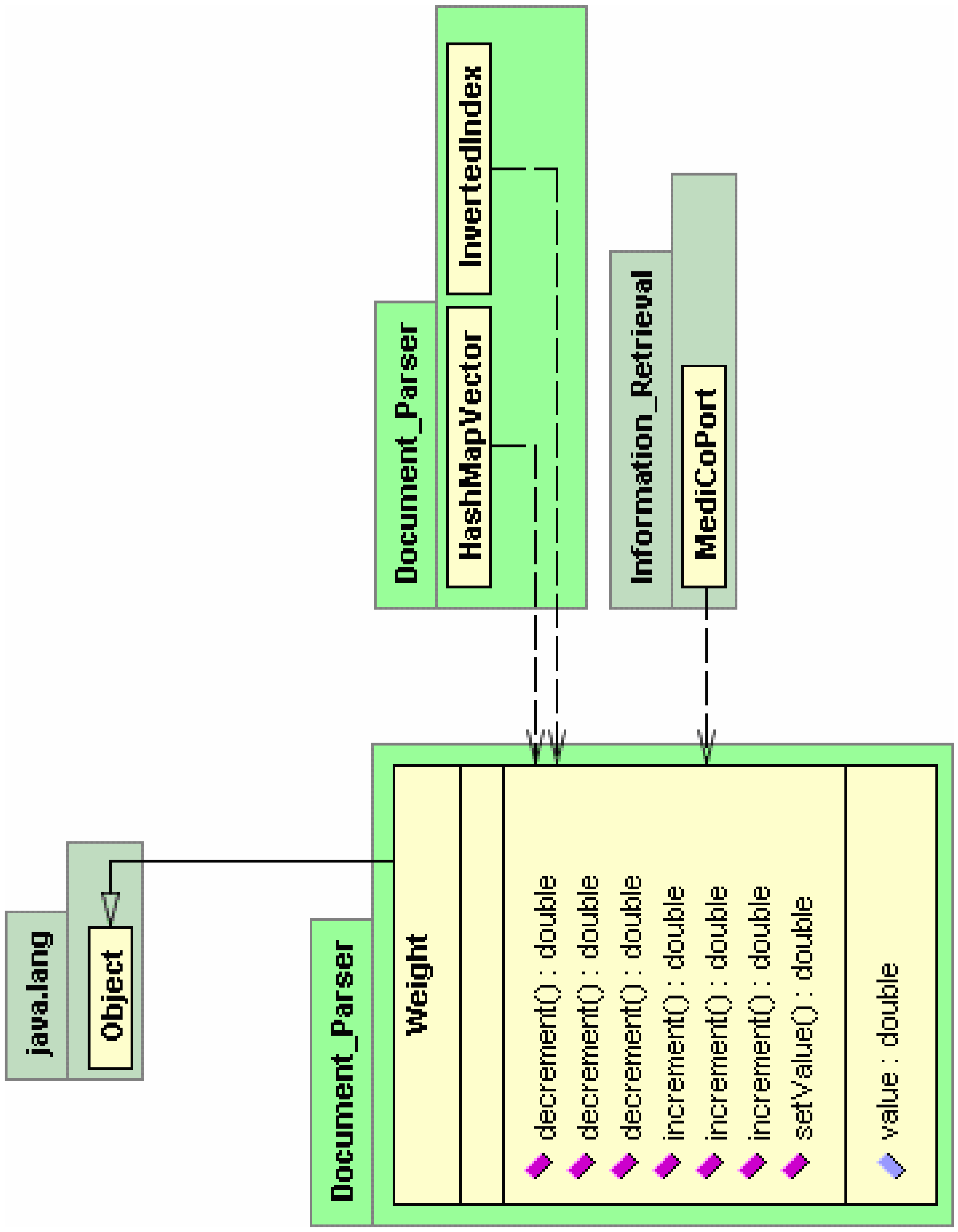
Appendix I: Package: Document_Parser; Class: TextStringDocument Class Diagram



Detailed Description of Class TextStringDocument:

Identification	:	TextStringDocument
Type	:	Class
Superclass	:	FileDocument
Purpose	:	An object that handles query strings as if text documents
Function	:	This class convert the query string to the query term list
Subordinates	:	String getNextCandidateTerm()
Dependencies	:	Document_Parser.InvertedIndex
Resources	:	java.io
		java.util

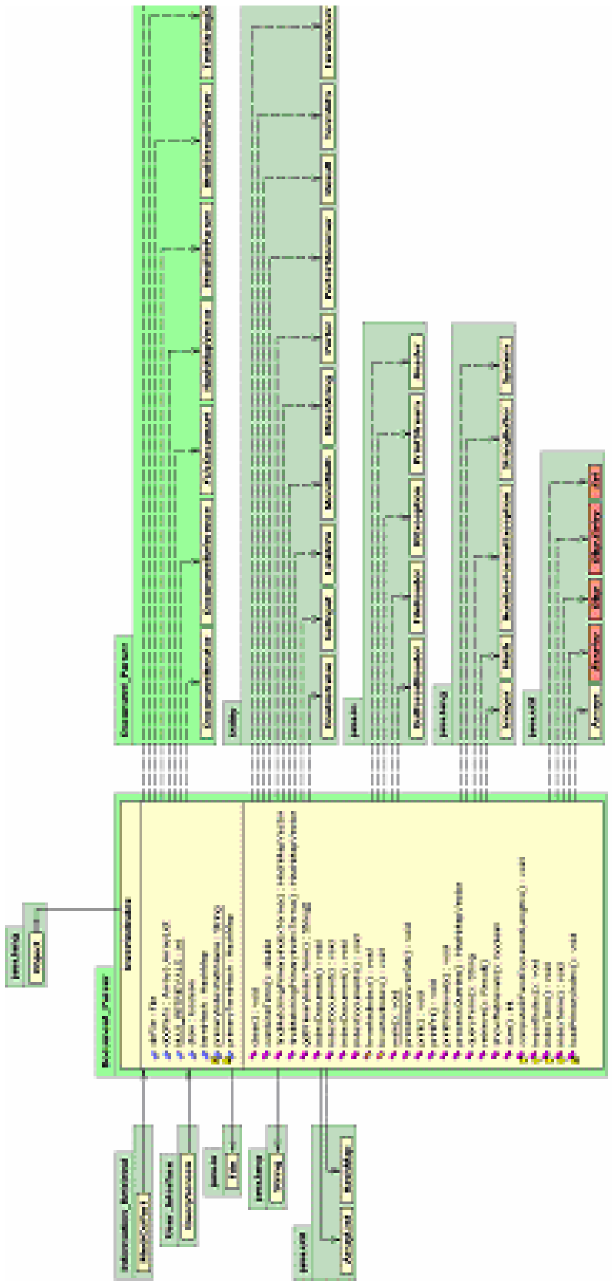
Appendix J: Package: Document_Parser; Class: Weight Class Diagram



Detailed Description of Class Weight:

Identification	:	Weigth
Type	:	Class
Superclass	:	Object
Purpose	:	A simple wrapper data structure for storing a double weight
Function	:	as an Object that can be put into lists, maps, etc. and then incremented, decremented, and set
Subordinates	:	double increment() double decrement() double getValue() double setValue(int value)
Dependencies	:	Document_Parser.InvertedIndex Document_Parser.HashMapVector
Resources	:	java.io java.lang java.net java.util

Appendix K: Package: Document_Parser; Class: InvertedIndex Class Diagram

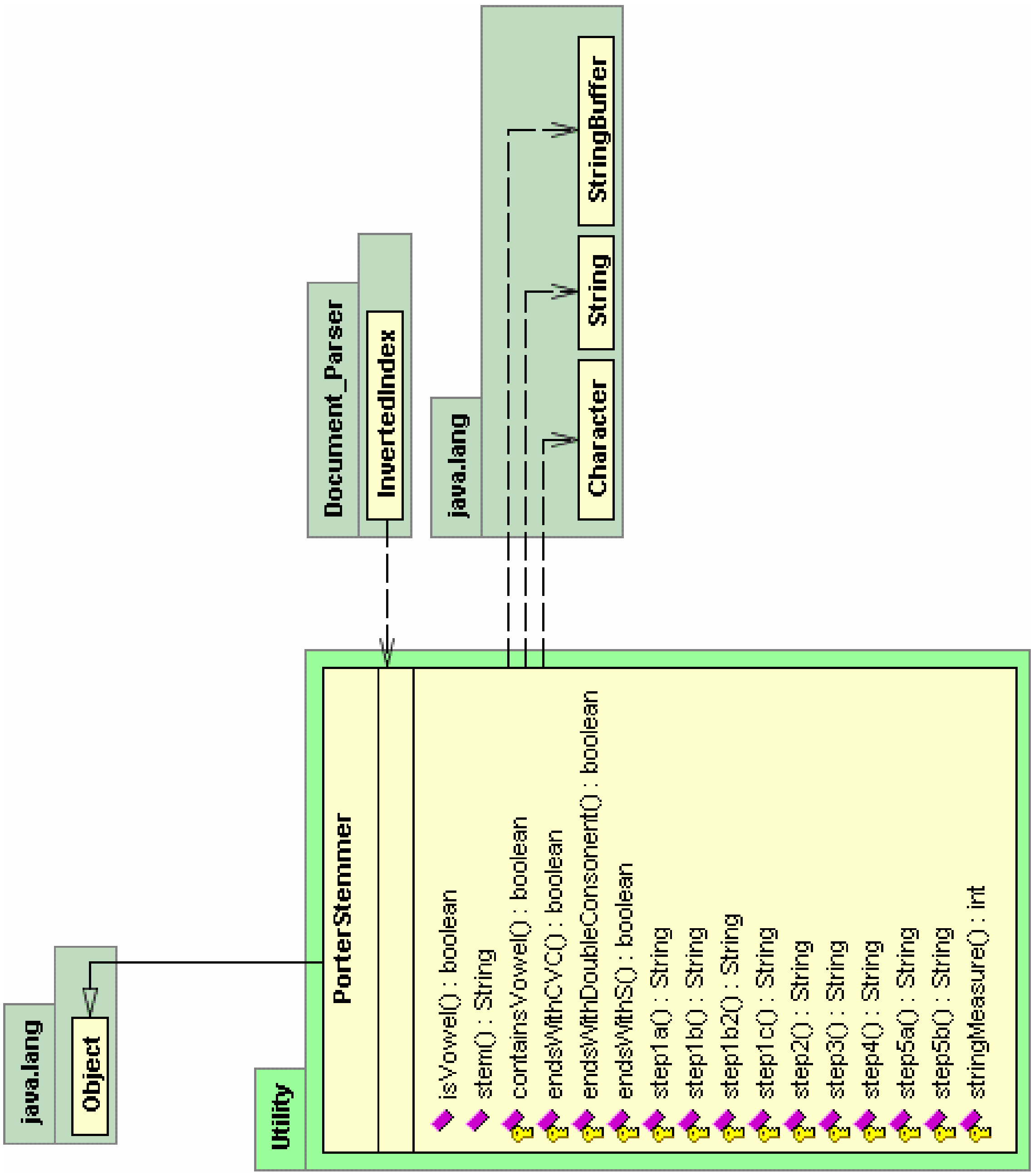


Detailed Description of Class InvertedIndex:

Identification	:	InvertedIndex
Type	:	Class
Superclass	:	Object
Purpose	:	An inverted index for vector-space information retrieval.
Function	:	Contains methods for creating an inverted index from a set of documents
Subordinates	:	void importIndex() void indexDocuments() void indexTerm(String term, int count, DocumentReference docRef) String queryTerm(java.lang.String queryterm) void print() void printPri() String[] getPrimaryIndexTerm() void loadPrimaryIndex() int size() HashMapVector processQueries(String query) HashMapVector findMatchingPrimaryIndexTerms(String query, HashMapVector queryVector) void presentAnswerSet(HashMapVector queryVector,Result[] retrievals) boolean showRetrievals(Utility.Result[] retrievals) void printRetrievals(Utility.Result[] retrievals, int start) Result[] retrieve(HashMapVector vector) double-combineTerm(String term,double count,hashMap retrievalHash) HashMapVector findMatchingPrimaryIndexTerms(String query)

Dependencies	:	User_Interface.QueryScreen
		Document_Parser.Weight
		Document_Parser.TextStringDocument
		Document_Parser.HtmlStreamParser
		Document_Parser.HtmlFileParser
		Document_Parser.HashMapVector
		Document_Parser.FileDocument
		Document_Parser.DocumentReference
		Document_Parser.DocumentIterator
		Utility.LinkInfo
		Utility.DoubleValue
		Utility.GetInput
		Utility.MoreString
		Utility.PorterStemmer
		Utility.TermInfo
		Utility.Result
		Utility.TermOccurence
		Utility.MoreMath
Resources	:	java.io
		java.lang
		java.net
		java.util

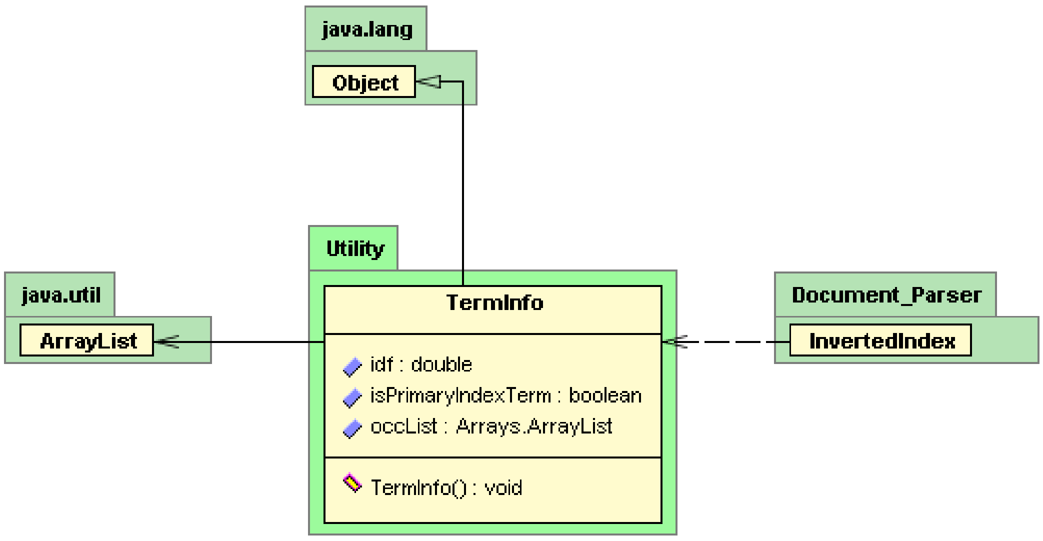
Appendix L: Package: Utility; Class: PorterStemmer Diagram



Detailed Description of Class PorterStemmer:

Identification	:	PorterStemmer
Type	:	Class
Superclass	:	Object
Purpose	:	The Porter stemmer for reducing words to their base stem form.
Function	:	This class implements the PORTER stemming algorithm, which is fully described in "An algorithm for suffix stripping",
Subordinates	:	String stripAffixes(java.lang.String str) boolean isVowel(char c)
Dependencies	:	Document_Parser.InvertedIndex
Resources	:	Java.lang

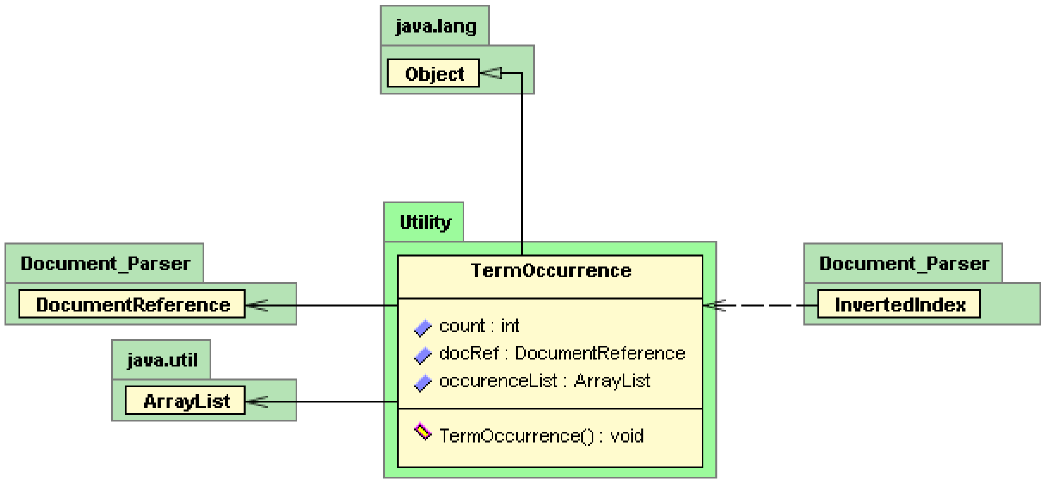
Appendix M: Package: Utility; Class: TermInfo Diagram



Detailed Description of Class TermInfo:

Identification	:	TermInfo
Type	:	Class
Superclass	:	Object
Purpose	:	A lightweight object for storing information about a term in an inverted index.
Function	:	This class holds occurrence list
Subordinates	:	This class has no method
Dependencies	:	Document_Parser.InvertedIndex
Resources	:	java.io java.lang java.util

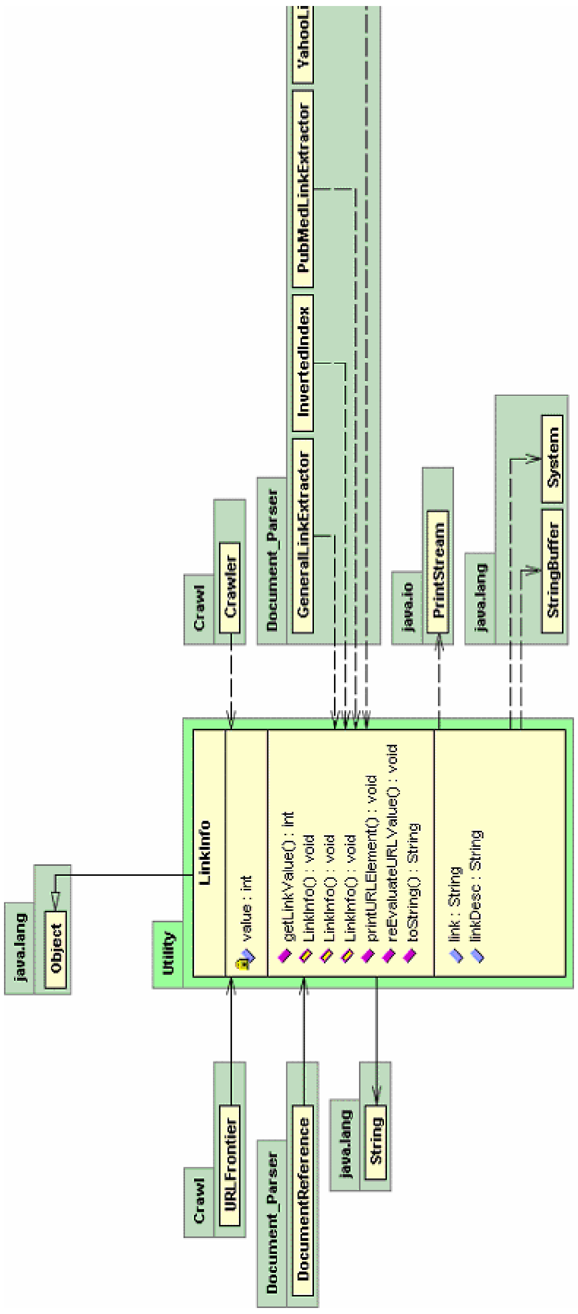
Appendix N: Package: Utility; Class: TermOccurence Diagram



Detailed Description of Class TermOccurence:

Identification	:	TermOccurence
Type	:	Class
Superclass	:	Object
Purpose	:	A lightweight object for storing information about an occurrence of a term in a Document.
Function	:	This class stores document references and term occurring counts
Subordinates	:	This class has no method
Dependencies	:	Document_Parser.InvertedIndex
Resources	:	java.io java.lang java.util

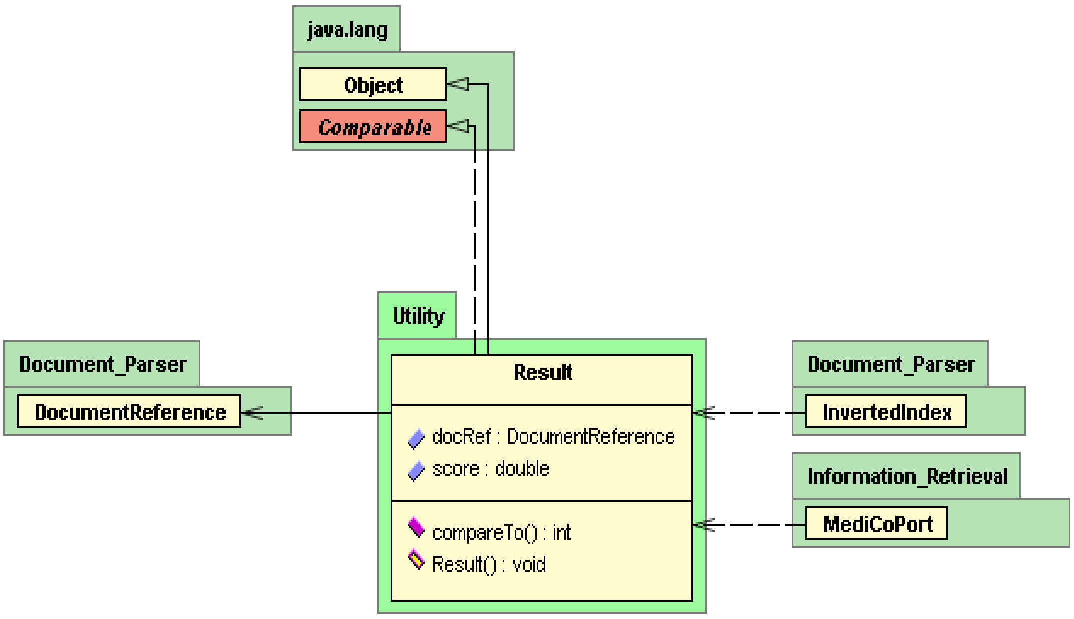
Appendix O: Package: Utility; Class: LinkInfo Diagram



Detailed Description of Class LinkInfo:

Identification	:	LinkInfo
Type	:	Class
Superclass	:	Object
Purpose	:	A lightweight object for storing information about an document's link
Function	:	This class holds document's link information and evaluates link value.
Subordinates	:	String getLink() String getLinkDesc() int getLinkValue() void reEvaluateURLValue(String URL, int newValue,int oldValue) void printURLElement()
Dependencies	:	Crawl.crawler Crawl.URLFrontier Document_Parser.GeneralLinkExtracter Document_Parser.InvertedIndex Document_Parser.DocumentReference.
Resources	:	java.io java.lang java.util

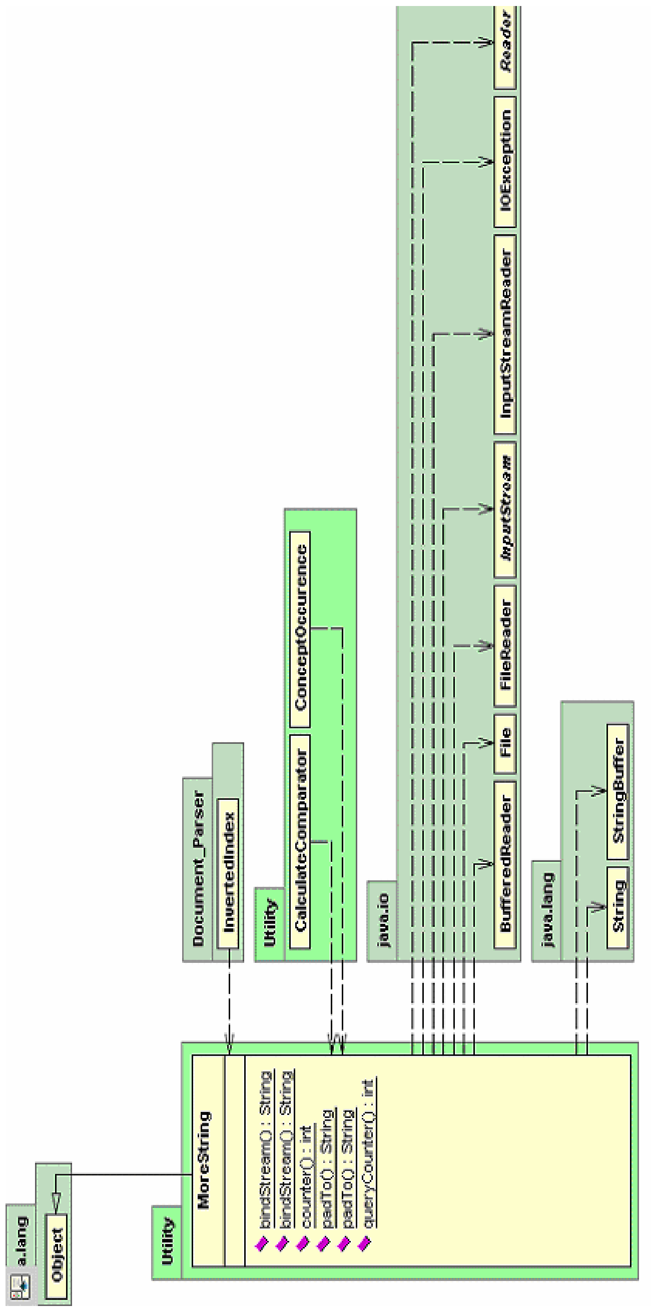
Appendix P: Package: Utility; Class: Result Diagram



Detailed Description of Class Result:

Identification	:	Result
Type	:	Class
Superclass	:	Object
Purpose	:	A lightweight object for element of answer set
Function	:	Compares this Result to another for sorting from best to worst.
Subordinates	:	int compareTo(Object obj)
Dependencies	:	Document_Parser.InvertedIndex
		Document_Parser.DocumentReference
Resources	:	java.io
		java.lang
		java.util

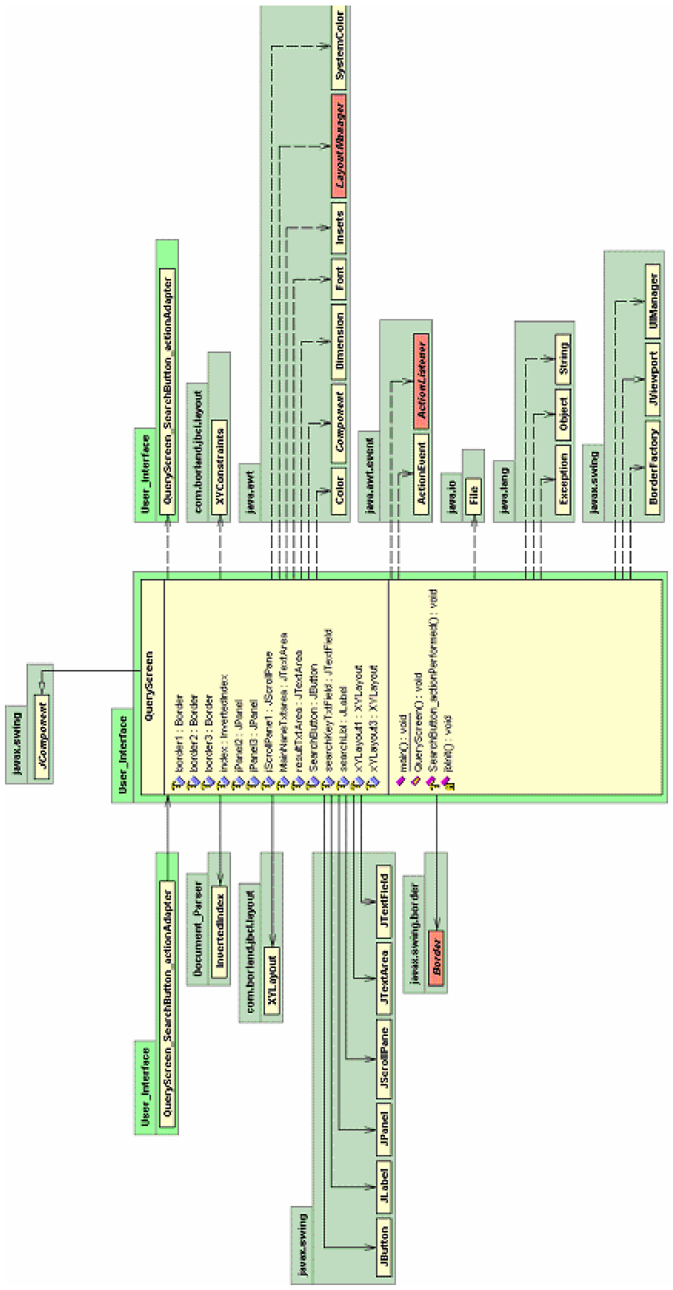
Appendix Q: Package: Utility; Class: MoreString Diagram



Detailed Description of Class MoreString:

Identification	:	MoreString
Type	:	Class
Superclass	:	Object
Purpose	:	This object provide string facility
Function	:	Pad a string with a specific char on the right to make it the specified length. It checks the query string has primary index term or not
Subordinates	:	static String padTo(String string, int length, char ch) static String bindStream(File HTMLFile) static int counter(String parentString, String childString, int count) static String bindStream(InputStream HTMLStream) static int queryCounter(String parentString, String childString, int count)
Dependencies	:	Utility.CalculateComparator Utility.ConceptOccurence Document_Parser.InvertedIndex
Resources	:	java.io java.lang java.util

Appendix R: Package: User_Interface; Class: QueryScreen Diagram



Detailed Description of Class QueryScreen:

Identification	:	QueryScreen
Type	:	Class
Superclass	:	JComponent
Purpose	:	Interface with system users
Function	:	This class interacts with system users
Subordinates	:	void actionPerformed(ActionEvent e)
Dependencies	:	Document_Parser.InvertedIndex
Resources	:	java.awt.*; javax.swing.*; com.borland.jbcl.layout.*; javax.swing.border.*; Document_Parser.InvertedIndex; java.io.File; java.awt.event.*;