To

Amma and Appa

BIOGRAPHY

Santthosh Babu Selvadurai was born on 31st of December 1981 in Salem, Tamil Nadu, India. He received his Bachelor of Engineering (B.E) Degree in Electrical and Electronics Engineering from Coimbatore Institute of Technology, Coimbatore (TN), India in 2003. After his undergraduate studies he joined Computer Sciences Corporation (CSC) as a Programmer Analyst where he was working on Enterprise Computing Systems especially IBM z/OS for about 3 years. Santthosh has been a graduate student at North Carolina State University, Raleigh, NC since August 2006. During this time he worked under the guidance of Dr. Gregory T. Byrd in field of grid computing and web services. Santthosh served a 3 month internship term with EMC Corporation at RTP, Raleigh, NC during summer 2007 where he was involved with the implementation of a distributed information infrastructure for one of their products. He has also been a Teaching Assistant for Computer Systems Programming (ECE209) course during fall 2007 semester at NC State University.

ACKNOWLEDGMENTS

First, I would like to thank my parents for everything they gave me in life. My mother Vimala and father Selvadurai have been a constant source of inspiration nurturing my ideas and supporting my decisions all the way to my graduate studies in spite of several financial and social difficulties.

I sincerely thank my advisor Dr. Gregory T. Byrd for giving me an opportunity to work under his guidance. He has been a constant source of inspiration, encouraging me to achieve more in every walk of my graduate studies. I have learned several incredible things from him including joyful positive attitude towards work and perfection at doing things. I thank him for his trust, continuous support and for all of his time. Working under him has been a very rewarding experience.

I would like to thank Dr. Munindar P. Singh for developing my knowledge on web services, introducing to me the emerging world of service oriented computing and also for serving on my thesis committee.

I would like to thank Dr. Yan Solihin for inspiring and fostering my knowledge on parallel computing architectures, for his constructive comments and for serving on my thesis committee.

Finally, I thank the almighty for all the grace and support to achieve my goals.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Search engines are information retrieval mechanisms designed to find information stored on computer systems. They help users minimize both time for finding required information and the amount of information that must be consulted to solve problems. Most popular search engines are used to search the World Wide Web. Other kinds of search engines include enterprise search engines, personal search engines and mobile search engines [33]. These search mechanisms over the recent past have taken several dimensions like personalized search, sponsored search, collaborative search, mobile search and so forth.

Finding desired information on the web in a timely and cost effective way is a problem of widespread interest. But very often users need to consult several disjoint search engines to meet their information needs other than regular web search, as the required information is scattered in several disjoint databases. Also the gap between the rate of information explosion on the web and the speed of search crawlers is constantly increasing [34]. A more concerning factor is the wealth of information that the deep web [1] hides from the web crawlers. Much of the deep web information has never been publicly indexed [35]. Most of these deep web contents are highly relevant to every information need, market and domain. Studies report the existence of thousands of special-purpose search engines on the web [1] which help us mine the deep web as well. Querying the right set of these individual engines to retrieve relevant information can be very time consuming and inefficient [36].

The solution to this problem is *metasearch engines*. A metasearch engine is a system that provides unified access to multiple existing search engines. A metasearch engine generally does not maintain its own index of documents; instead it queries several

participating search engines and aggregates their individual results into a unified result set, re-ranked based on the relevance to the query.

Metasearch engines increase the search coverage, solve the scalability issues in searching eclectic information sources, facilitate the invocation of multiple search engines and improve information retrieval effectiveness [2]. Building a metasearch engine involves several design and technical challenges, including database or component engine selection, query analysis, query scheduling or dispatch, rank aggregation and result merging [18]. Clearly, result merging is a key component of the metasearch system. The effectiveness of a metasearch system is closely related to the result merging algorithm it employs. Studies have shown that simple result merging strategies can outperform most popular web search engines [2].

In this thesis we develop a general purpose framework in Java to build metasearch engines, and use this system to evaluate a new type of result merging approach that deems to yield highly relevant results for user queries. The metasearch framework is highly modular and flexible. Component search engines can be added or removed on the fly. Any of the segregation (query preprocessing), aggregation (result merging), or scheduling modules can be dynamically changed at runtime to suit the context of the search. This dynamic nature of the framework helps us compare and evaluate the performance of individual search engines and different metasearch configurations. Component search engines are normally queried using Simple Object Access Protocol (SOAP) [37], Remote Procedure Calls (RPC) [37], XML-RPC (eXtensible Markup Language RPC) [37] or AJAX (Asynchronous JavaScript and XML) calls. Our metasearch framework primarily uses Apache Lucene [3], a feature rich, open source, high performance, and cross-platform text search engine library to provide

the necessary underlying infrastructure support. Testing of the framework modules has been done using the JUnit [4] Library.

We make use of this framework to study a new type of rank aggregation technique which we call content-directed result merging algorithm. In this, the user presets the perspective of his search by uploading a relevant document or by entering detailed description that can help the ranking mechanism to efficiently rank the metasearch result set. User feedback has been a key force to improving search result relevance [5], but this algorithm takes a radical approach by obtaining explicit user input and using it to arrange the results in the most relevant order unlike conventional user feedback based machine learning techniques [6].

A prototype metasearch engine has been implemented based on the framework using Google Web Developer kit (GWD). The prototype is highly configurable, and it provides user interfaces to add and remove component engines, make simple queries and content-directed queries by uploading the guide document or entering a descriptive text.

We evaluate the performance of our content-directed result merging algorithm against similar result merging approaches and individual search engines by comparing the relevance of the obtained result set. Queries from TREC (Text REtrieval Conference) [18] Web Track standards are used for evaluation.

Outline

The outline of this thesis is as follows.

Chapter 2 gives the information retrieval basics and an introduction about metasearch engines. The chapter explains the components of a metasearch engine, methods of querying

several search engines, and technical challenges in scheduling, coverage, timeliness and relevance of results.

Chapter 3 introduces rank aggregation and result merging, a key stage in metasearch. Different ranking algorithms and result merging strategies are discussed in this chapter. We propose a novel approach to ranking the metasearch result set using user-provided content-direction.

Chapter 4 explains the architecture of the proposed metasearch framework model. The chapter outlines the need for a metasearch framework, design decisions and grounds for exploiting content-direction in metasearch ranking.

Chapter 5 presents the implementation specific details of the framework and the metasearch engine prototype that we have built to evaluate the performance of the content-direction algorithm.

Chapter 6 discusses in detail the problems in assessing search mechanisms, the evaluation strategy used for the framework and the content-direction technique. The chapter also provides detailed comparison results on the performance of our algorithm against individual search engines.

* * *

Chapter 2

METASEARCH ENGINES

This chapter provides a brief overview of Information Retrieval (IR) systems, problems of scale and complexity in IR, metasearch engines, and their architectural overview.

2.1 Information Retrieval Systems (IRS)

2.1.1 Information Systems

An Information System (IS) is a system of persons, data records and activities that process the data and information in a given organization, which may include manual processes and automated schemes. Computers have dominated information systems by aiding automation of information processing efforts. The four important computer based information systems [6] are Management Information Systems (MIS), Database Management Systems (DBMS), Question-Answering (QA) systems and Information Retrieval (IR) systems.

In the context of information systems, information retrieval can be defined as the process that deals with the representation, storage and access to documents or representatives of documents (document surrogates). The input information is likely to include natural language texts, document excerpts and abstracts. The output of an IR system is response to search requests, with each request containing a set of references. These references are intended to provide the users with information about items of potential interest. Information

retrieval has significant overlap with other information systems. Figure 2.1 illustrates the functionality and overlap of each information system with information retrieval systems.

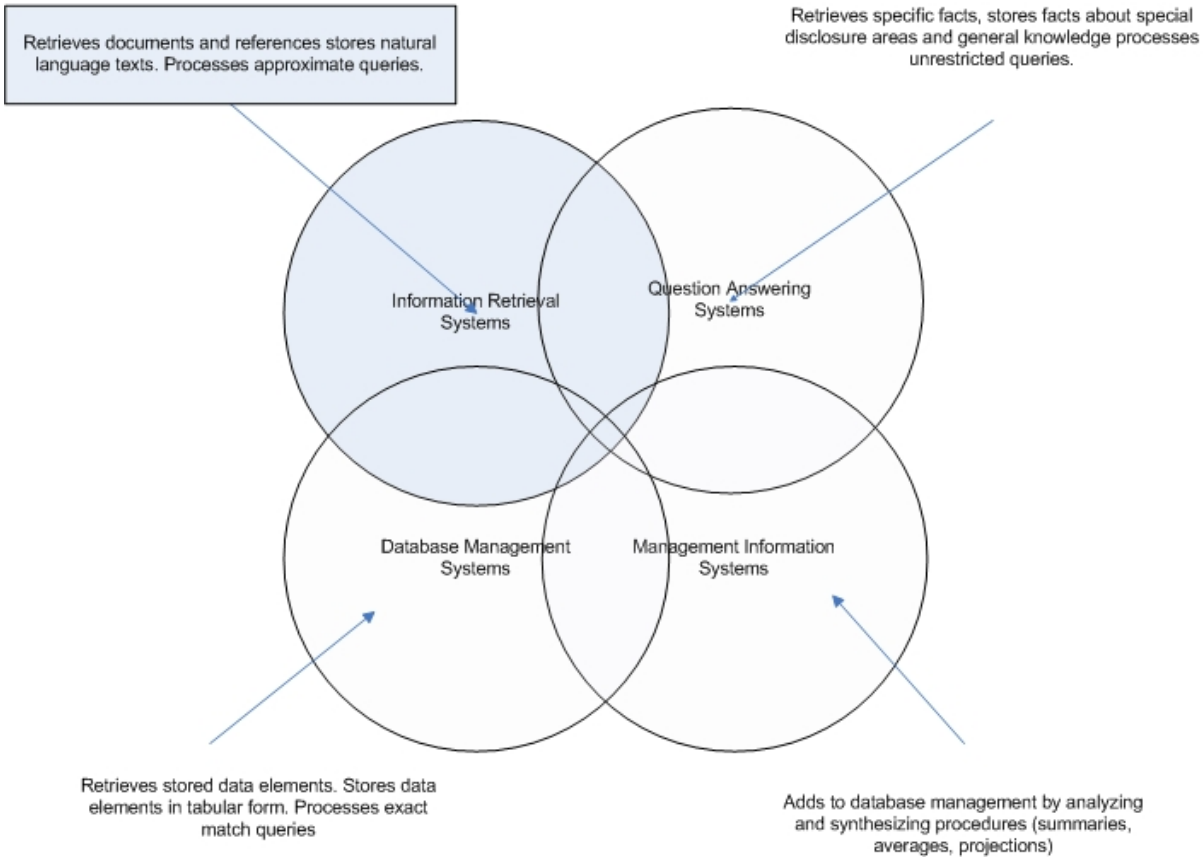

Figure 2.1 Overlap among information system types [6]

2.1.2 Functional approach to IR

Several different information retrieval systems exist but most of them use a functional approach to IR. Figure 2.2 provides an illustration of functional overview of information retrieval. According to this approach, information retrieval system consists of three components, a set of information items (DOCS), a set of requests (REQS) and a set of mapping mechanisms (SIMILAR) [6]. In a retrieval process, the information items are first converted to a special form using a classification or indexing language (LANG). The indexing language is either pre-specified (controlled) or taken freely from the text of the

information items and information requests (uncontrolled) or sometimes a combination of them. Whichever type is used, an information item is usually assumed to be representable by a list of elements from the indexing language.

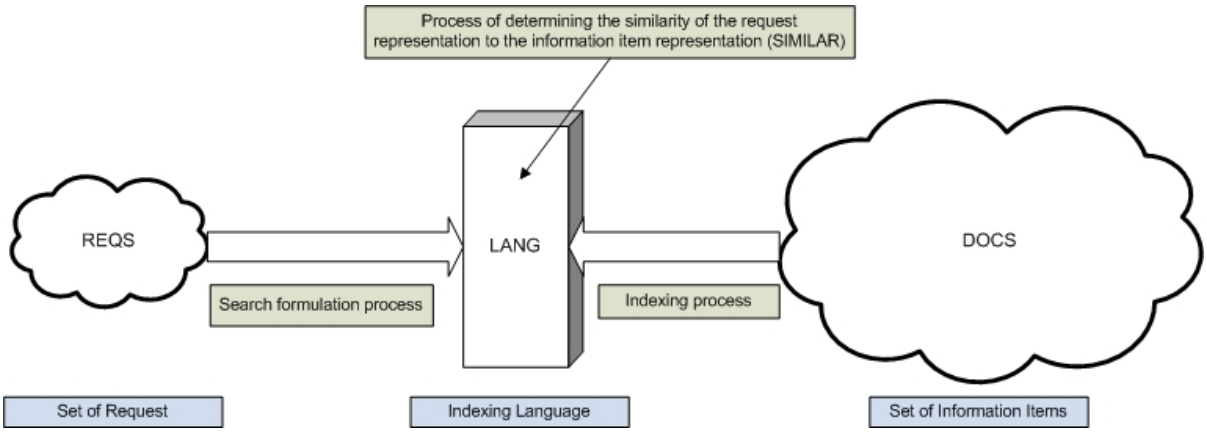

Figure 2.2 Functional Overview of information retrieval [6]

The mapping of information items to the indexing language is called the indexing process. The requests are also converted into a representation consisting of elements from LANG, referred to as the query negotiation process.

Based on the elements of indexing language, procedures are setup to determine which information items are to be retrieved in response to a particular query. SIMILAR is the relation operator and retrieval function that determines the similarity of the various information items to a given request.

2.2 Search and retrieval

2.2.1 Searching process

A search process involves several steps. Flowchart 2.3 provides an illustration of how an informed user searching databases of information items that use controlled index languages can obtain optimal results [7].

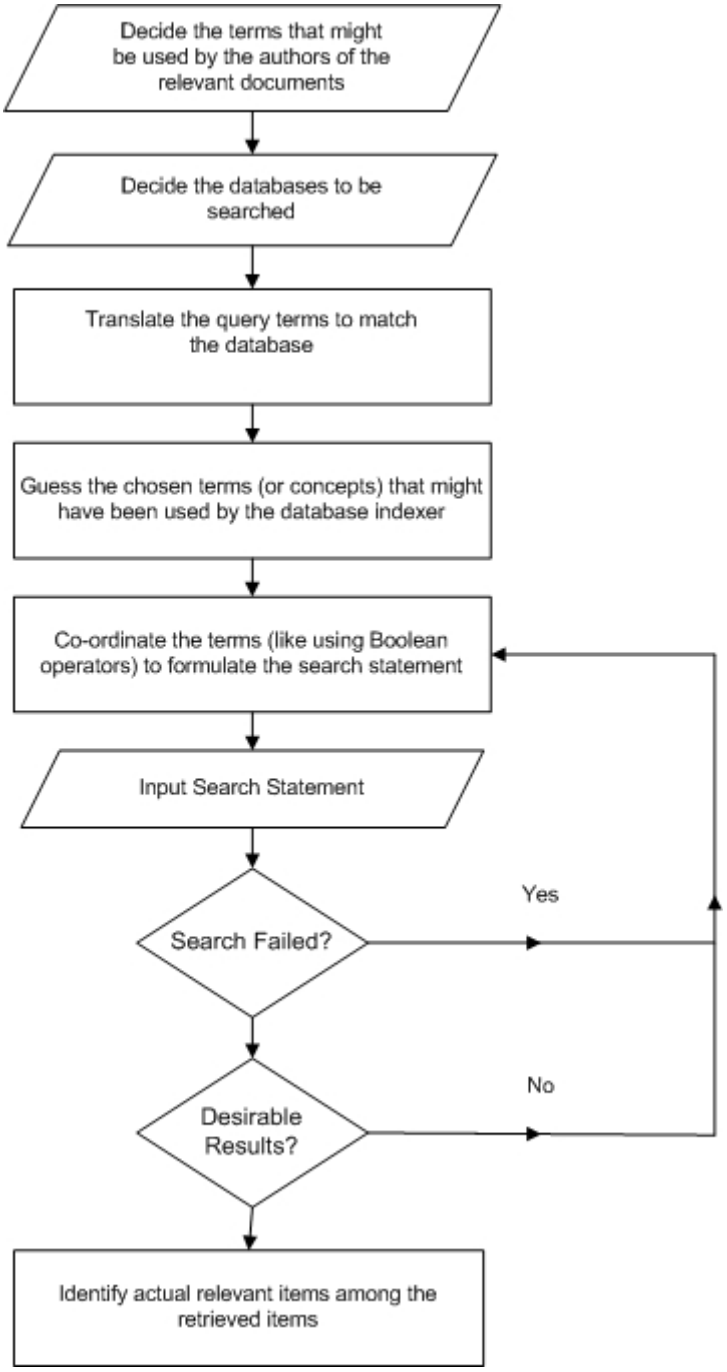

Figure 2.3 Steps to perform an optimal search

One major task in the searching process relates to the coordination of terms (first four steps in flowchart) in order to formulate the actual search statement. The result of search depends largely on how adequately the search terms are combined. Boolean search

techniques have been widely used since the beginning of computerized information retrieval [38].

2.2.2 Retrieval models

Several information retrieval models have been proposed to study and improve the retrieval process. The IR models can be broadly classified into three groups; user-centric or cognitive models, system-centric models, and alternative models [38].

The cognitive models take a holistic view of information retrieval. Other than the retrieval mechanisms used in matching queries with stored information, they also take into account the ways in which the user information needs can be formulated as a query, the human-computer interactions that take place during the search process, the social and cognitive environments in which the process takes place and how the information is used by the user to meet specific information need.

The system-centric models provide an explicit statement on the workings of the information search and retrieval mechanism. They are based on logical and mathematical principles, such as Boolean search, Probabilistic retrieval and Vector processing [6, 7, 38]. In a Boolean search model, queries are compared with the term set used to represent document contents. A probabilistic model does this comparison by computing relevance probabilities for the documents of a collection. The vector processing model represents both the documents and queries by term sets and compares global similarities between queries and documents. The Best match searching and relevance feedback model (explained in section 2.2.3) also falls under the system-centric category.

Alternative IR models have also been proposed which do not involve use of significant statistical analysis or probabilistic calculations. Two of the most popular

alternative information retrieval models are the natural language processing model which performs syntactic, semantic and pragmatic analysis of the query consuming huge system resources and the hypertext model which requires a necessary hyperlinked structure of documents to identify order of relevance.

2.2.3 Best match searching and relevance feedback model

Best match searching is aimed to produce a ranked output; hence it requires a method to measure the relative significance of retrieved items, which again requires some method of weighting search terms. A similarity measure is comprised of two major components: 1) a term weighting scheme that reflects the importance of a term by allocating numerical values to each index term in a query or document, and (2) a similarity coefficient which uses these weights to calculate the similarity between a retrieved item and a query.

A best match search matches a set of query words against the set of words corresponding to each item in the database, calculates a measure of similarity between the query and the item, and then sorts the retrieved items in order of decreasing similarity with the query [6]. This involves some quantitative measure of similarity between the query and each of the items in the file, and the ranking is formed on the basis of these similarities. Studies show that weighting schemes that use term frequency, collection frequency and vector length normalization tend to produce the best results [8, 9].

The relevance feedback process is a controlled, automatic process for query reformulation, where the basic idea consists of choosing important terms attached to certain previously retrieved items that have been identified as relevant by the user, and enhancing the importance of these terms in the new query formulation [10]. In this case, an initial search is carried out to produce a ranked list of outputs, and the user inspects a few top-ranking

documents to ascertain their relevance to the given query. The user's relevance data enables the system to calculate a new set of weights that should reflect the importance of each query term more accurately. Studies have shown the relevance feedback process to be an inexpensive and effective way of reformulating queries based on previously retrieved relevant and non-relevant documents [10].

This best match search and relevance feedback model is the most relevant to our work as it provides a theoretical foundation for our content-directed result merging algorithm.

2.3 Metasearch Engines

2.3.1 Motivation and Goals

Quiet often, information needs of users are stored in the databases of multiple search engines; this applies to the Word Wide Web as well. The rate of information explosion on the internet is much higher than the rate at which web search engines index the web. It is becoming much harder for individual search engines to keep themselves up to date with the expansion of internet [1]. Also, users frequently require their search to cover personal or organizational databases along with the world wide web of documents.

It is highly inefficient and inconvenient for an ordinary user to manually invoke multiple search engines and identify useful documents from the returned result sets. To support unified access to multiple search engines, a metasearch engine can be constructed. The key motivation behind the construction of a metasearch mechanism is to increase the coverage and scope of search, handle the search of voluminous information efficiently, provide a single point of access for several search interfaces, and improve the retrieval effectiveness.

In a session, a metasearch engine has to receive a query from the user, invoke underlying search engines (also referred as component search engines) with the user query as a parameter, retrieve information relevant to the query, unify the individual result sets into a single ranked list, and order them according to relevance. Different component search engines may accept queries in different formats; the user query may thus need to be translated into an appropriate format for each local system. Result sets are likely to be a list of document identifiers, such as Uniform Resource Locators (URLs) for web pages or documents, with a short description.

2.3.2 Component Architecture

Apart from the underlying search engines, a metasearch mechanism has four primary software components: a database selector, a document selector, query dispatcher and a result merger. Reference software component architecture of a metasearch engine is illustrated in Figure 2.4. The numbers on the edges indicate the sequence of actions for a query to be processed.

The problem of identifying potentially useful databases to search for a given query is known as *database selection.* Database selector component is responsible for database selection for each query, thereby avoiding wasting of resources in searching irrelevant data sources. The role of database selector is more pronounced when there is a long list of component engines that need to be queried. Database selection is not an issue when querying public databases like the internet search engines or a minimal set of data sources unless there are severe performance concerns.

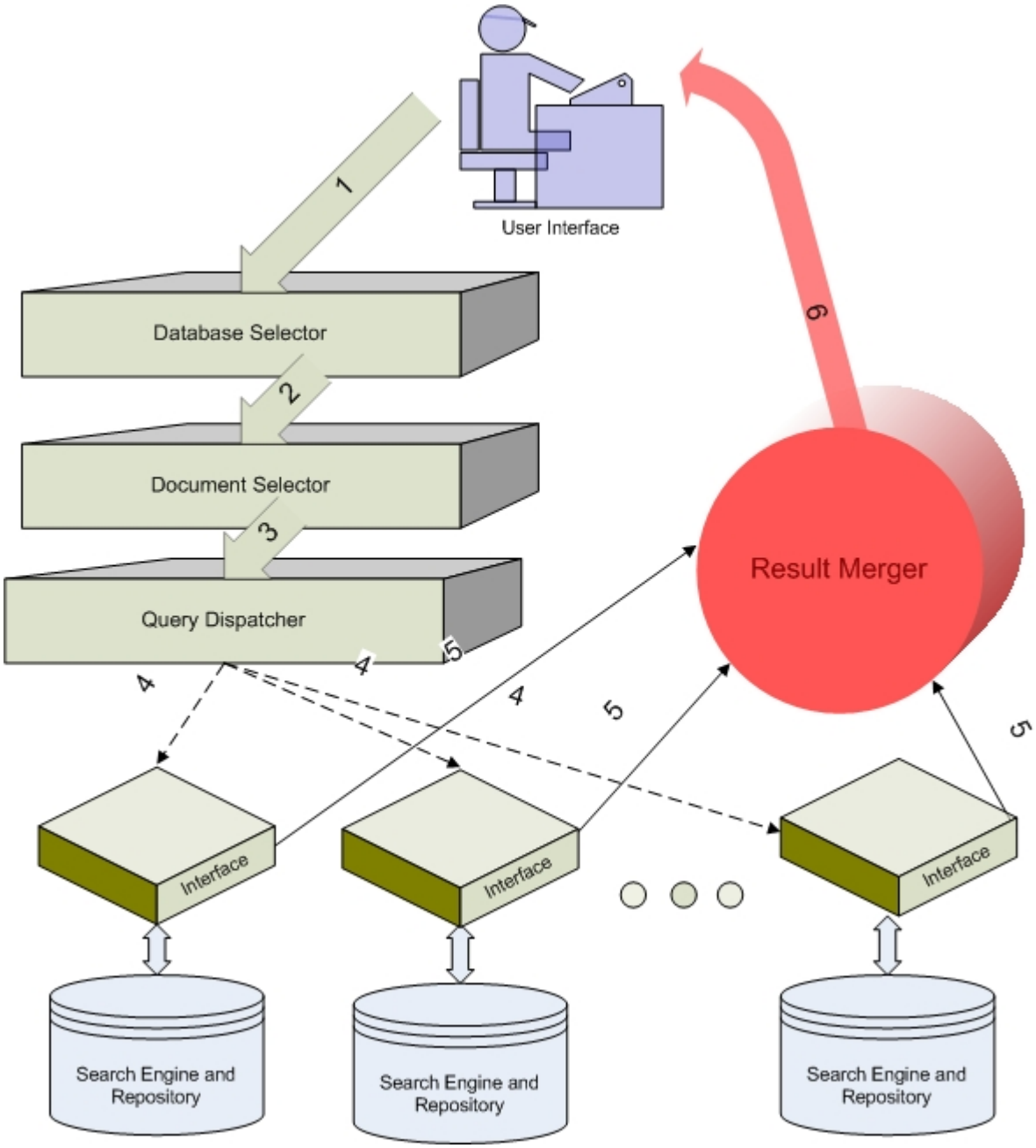

Figure 2.4 Metasearch Component Architecture

For each component search engine selected by the database selector, the *document selector* determines what documents to retrieve from the database of that search engine. The goal is to increase the retrieval effectiveness by minimizing the number of useless documents. Several factors may affect the selection of documents from a component search

engine such as the number of useful documents, similarity function or specific target of search like web pages, videos, news, maps, books, documents, products, and so forth.

The query dispatcher is responsible for establishing a connection with each selected search engine and passing the query to it. The original query may need to be translated to a new query before being sent to a component search engine. For example, the query input to the metasearch engine may be Boolean or vector space queries, while the component engines may be serving heterogeneous information [11] in different types and formats. Query dispatcher may also try to adjust the relative weights of query terms in the original query to fetch optimal results.

Result merger combines the results into a single ranked list. This component has the most impact on retrieval effectiveness. The relevance ordering of documents is directly related to the ranking algorithm used by the result merger. An ideal result merger would rank all the returned documents in descending order of their global similarities with the user query. We shall discuss several result merging algorithms in the next chapter.

2.3.3 Heterogeneity

One of the serious challenges in designing a metasearch engine is the management of complexities involved with the heterogeneous environments of component search engines. Each component search engine many have its own indexing method, document term weighting scheme, query term weighting scheme, similarity function, document database, document versions and result presentation schemes.

Heterogeneity posts two major challenges with respect to database selection algorithms. One is to identify appropriate representatives that permit fast and accurate estimation of database usefulness. At the same time the representatives should be small in

comparison to the size of the database, and should be easy to obtain and maintain. Our work does not deal with database selection, so we will not be discussing this in detail.

The result merging also faces issues due to heterogeneity of the component engine. The main problem is that the same document may have different global and local similarities with a given query. For example a document listed as highly relevant by a component search engine (local similarity) may be categorized as low relevance document when compared with results from other search providers (global similarity). Result merging techniques have to find ways to estimate the global similarities of documents so that documents returned from different component search engines can be properly merged.

In the next chapter we examine the techniques that have been proposed to deal with problems in result merging which is one of the key focuses of our research.

* * *

Chapter 3

RANK AGGREGATION & RESULT MERGING

3.1 Foundations

Traditional result merging techniques have been focusing on normalization of scores [11, 12] to make them more comparable across different search engines and thereby uniformly ranking the retrieved results. But these techniques require the component search engines to provide the local scores.

In the context of metasearch engines, a voting-based result fusion called as Borda count [13] is more appropriate. In Borda count, each component search engine is considered a voter that has a specified number of votes or points, and each returned result is considered a candidate. Each voter's top ranked candidate is assigned n points (where n is the number of candidates), the second top ranked candidate is given n-1 points, and so on. For candidates that are not ranked by a voter (i.e. the result candidate was not retrieved by corresponding search engine), the remaining points of the voter will be evenly divided. The results are then ranked in descending order of the total reported. For Borda count to be efficient each voter has to return a large number of results (in hundreds) and also the voters need to have substantial overlap among the retrieved results. There are also schemes that take search engine usefulness into consideration like D-WISE [21].

All these require some data to be collected from each component search engine before hand. This is not applicable in a metasearch context where a component engine can join the metasearch on the fly. Therefore our research focuses only on result merging approaches without collecting any sample data in advance from component search engines.

The result records returned from today's search engines are referred to as Search Result Records (SRRs). A typical SRR contains the URL, title and summary (snippet) of the retrieved document. Some of the earlier works [15, 16] have focused on utilizing the SRRs to gauge the relevance of the results, but these works primarily focus on merging news results rather than general purpose search results.

The most effective result merging methods uses a combination of evidences from document title, snippet and the search engine usefulness [16]. First, for each SRR the similarity between the query and its title and the similarity between the query and its snippet are computed; then the two similarities are linearly aggregated to rank the result. In the next section we discuss a list of such rank aggregation methods, followed by a brief description of our content-directed result merging strategy.

3.2 Merging Algorithms

3.2.1 Top Document search engine score (TopD)

Let S_j represent the usefulness score of search engine j with respect to query Q. The TopD method requires the document frequency of every term be collected in advance. Here document frequency of term refers to the number of times a particular term appears in the document. TopD algorithm uses the similarity between Q and the top ranked document returned from search engine j (denoted as d_{1j}) to estimate S_j. This ranking assumes the highest ranked document to be the most relevant to the user query based on the search engine's ranking criteria. The similarity functions used for comparison can be *Cosine*[1] [15]

[1] *Cosine similarity function compares each term weight in the query with the term frequency in the document*

or *Okapi*[2] [14]. Once S_j is computed the ranking scores (rs_{ij}) can be computed using the expression

$$rs_{ij} = 1 - (r_i - 1) * S_{min} / (m*S_j) \quad \ldots(1)$$

Here m is the total number of documents desired, S_{min} is the smallest search engine score among all search engines selected for this query, and r_i is the local rank of a document *i* returned from search engine *j*.

The technique is not very effective, as the top ranked document needs to be retrieved from the local server for the merging process. Also the rank of top ranked results from all used search engines will be the same, therefore one needs to compute adjusted ranking scores by multiplying the ranking scores computed by S_j. If a document is retrieved by multiple search engines, we need to compute its final ranking score by summing up all the adjusted ranking scores.

3.2.2 Top SRR based search engine score (TopSRR)

TopSRR uses each component search engine's top *n* returned Search Result Records (SRRs), instead of the top ranked document, to estimate the search engine score S_j with respect to the query Q. This is done with the intuition that the SRRs are representative of the original documents. Specifically, all the titles of the top *n* SRRs from search engine *j* are merged together to form a title vector TV_j, and all the snippets are also merged into a snippet vector SV_j. The similarities between query Q and TV_j and between Q and SV_j are computed separately and then aggregated into the score of search engine j. This search engine score is computed as

[2] *Okapi similarity function considers the weighted sum of Okapi weights for terms with the query and document.*

$$S_j = c_1 * \text{Similarity}(Q, TV_j) + (1\text{-}c_1) * \text{Similarity}(Q, SV_j) \quad \ldots(2)$$

Where c_1 is a constant and Similarity function can be either Cosine or Okapi.

3.2.3 Simple Similarity rank between SRR and query (SRRSim)

The SRRSim ranking scheme directly computes the rank of documents using SRRs, instead of first computing the search engine scores and then the individual document ranks. The SRRSim uses the fact that the SRR is representative of the corresponding full document; thus one can rank SRRs returned from different search engines based on their similarities with the query directly, using the right similarity function.

The similarity between SRR R and query Q is computed as the weighted sum of similarity between the title T of R and Q and the similarity between the snippet S of R and Q.

$$\text{Sim}(R,Q) = c_2 * \text{Similarity}(Q,T) + (1\text{-}\ c_2) * \text{Similarity}(Q,S) \quad \ldots(3)$$

If a document is retrieved from multiple search engines with different SRRs, then the similarity between the query and each such SRR will be computed and the largest one will be used as the final similarity of this document with the query for result merging.

3.2.4 Compound similarity rank between SRR and Query (SRRSimMF)

All the previous ranking algorithms propose metasearch engines to use only the similarity functions to determine the relevance of documents against the given query. They do not take the proximity, frequency and ordering of terms into consideration. The impact of order and proximity of information in the result ranking is important and obvious. To better rank SRRs, this algorithm takes several features with respect to the query terms, like the

number of distinct query terms appearing in the title and the snippet (NDT), the total number of occurrences query terms in the title (TNT) and the snippet (TNS), the location of the occurred query terms (TLoc), ordering and adjacency information (ADJ), and the window size containing the distinct occurred query terms (WS). To quantify the matches based on the different features identified above the SRRSimMF algorithm aggregates the scores into numeric values based on the following expressions

$$Similarity = TNDT * (c_3 * Sim(T,Q) + (1-c_3) * Sim(S,Q)) / QLEN \quad \ldots(4)$$

TNDT is the total number of distinct query terms appeared in title and snippet

QLEN is the length of the query

Optimal value of c_3 is 0.2 computed using a genetic algorithm [19], intuitively this means that lesser significance is given to the similarity between title and query and more significance is placed on the similarity between the query and the description.

$$Sim(T,Q) = S_{TNDT} + (W_1 * S_{ADJ} + W_2 * S_{WS} + W_3 * S_{TNT}) / QLEN \quad \ldots(5)$$

$$Sim(S,Q) = S_{SNDT} + (W_1 * S_{ADJ} + W_2 * S_{WS} + W_3 * S_{TNS}) / QLEN \quad \ldots(6)$$

$$S_{TNDT} = No.\ of\ Distinct\ Query\ Terms\ (NDT)\ that\ match\ TITLE\ /\ QLEN \quad \ldots(7)$$

$$S_{SNDT} = No.\ of\ Distinct\ Query\ Terms\ (NDT)\ that\ match\ SNIPPET\ /\ QLEN \quad \ldots(8)$$

$$S_{TNT} = Count\ of\ matching\ terms\ in\ Title\ (TDT)\ /\ Title\ Length\ (TITLEN) \quad \ldots(9)$$

$$S_{TNS} = Count\ of\ matching\ terms\ in\ Snippet\ (TDS)\ /\ Snippet\ Length\ (SLEN) \quad \ldots(10)$$

$$S_{WS} = (Title/Snippet\ length - Window\ Size)\ /\ (Title/Snippet\ length) \quad \ldots(11)$$

S_{ADJ} is set to 1 if the ordering is same in query and title/snippet otherwise the value is 0

W_1, W_2 and W_3 are weights for each of the terms

Previous study [18] has shown that the window size and adjacency information offer little to the similarity calculation and they are computationally expensive, hence they can be safely ignored without much loss of similarity mapping. Equations (5) and (6) become

$$Sim(T,Q) = S_{TNDT} + (S_{TNT} / QLEN) \quad \ldots(12)$$

$$Sim(S,Q) = S_{SNDT} + (S_{TNS} / QLEN) \quad \ldots(13)$$

Equations (11) and (12) are the ones used in actual implementations.

3.3 Content-direction

3.3.1 Motivation

All general purpose search engines have a ceiling on the query length for performance reasons, which limits the effectiveness of the similarity functions and the relevance of retrieved results. Most times there isn't enough information to judge the relevance between the query and the retrieved results. The best match searching and relevance feedback model discussed in section 2.2.3 show the importance of query reformulation for effective retrieval of matching results. Reformulation is an automatic, controlled process which chooses important terms attached to certain previously retrieved items that have been identified as relevant by the user (referred as guide artifacts) and enhances the importance of these terms in the new query [9]. Studies have shown the relevance feedback model to be very effective, but query reformulation has a significant computation and complexity overheads. Instead of query reformulation, we suggest the use of a guide artifact for similarity analyses in conjunction with the query and SRR to rank the results.

3.3.2 Content-Directed Result merging algorithm (CDRM)

In this algorithm, the user specifies a guide artifact, typically a document relevant to the field or domain or namespace in which the user is performing the search. When the results are retrieved from several component search engines, a similarity analysis of the SRRs is done as in SRRSimMF, including both the query and the guide document. The guide document supplements the relevance information that is missing in the limited query terms for making informed decisions. A guide document can have thousands of distinct terms, and its size is only limited by the performance. Like the query, the guide document is compared with result titles and snippets individually, and the scores are aggregated to compute the rank. For CDRM the similarity function is computed as

$$Similarity = [TNDT * (c_3 * Sim(T,Q) + (1-c_3) * Sim(S,Q)) / QLEN] + [TNDT_{Guide} * (c_3 * Sim(G,T) + (1-c_3) * Sim(G,S) / (TITLEN + SLEN)] \quad \ldots(14)$$

Here,

$TNDT_{Guide}$ is the total number of distinct query terms appeared in guide document

$$Sim(G,T) = S_{GTNDT} + (S_{GTNT} / TITLEN) \quad \ldots(15)$$

$$Sim(G,S) = S_{GSNDT} + (S_{GTNS} / SLEN) \quad \ldots(16)$$

$$S_{GTNDT} = No.\ of\ Distinct\ Title\ Terms\ (NDT)\ that\ match\ GUIDE\ /\ TITLEN \quad \ldots(17)$$

$$S_{GSNDT} = No.\ of\ Distinct\ Snippet\ Terms\ (NDT)\ that\ match\ GUIDE\ /\ SLEN \quad \ldots(18)$$

$$S_{GTNT} = Count\ of\ matching\ guide\ terms\ in\ title\ (TDT)\ /\ TITLEN \quad \ldots(19)$$

$$S_{GTNS} = Count\ of\ matching\ guide\ terms\ in\ snippet\ (TDS)\ /\ SLEN \quad \ldots(20)$$

As in the SRRSimMF we ignore the location and adjacency information. Even with a reasonably-sized guide document, computing location and adjacency features can be

expensive, and these tend to have little effect on the relevance ranking for a little set of results [18]. These information features are more relevant when there is a huge volume of results to be ranked, which is typically not the case with a general purpose metasearch engine.

3.3.3 Use Cases

To better understand the effectiveness of CDRM in retrieving relevant results, let us look at some use cases.

1. When a computer engineer working on the architecture of memory buses tries to use a general purpose search engine to search the query term '*bus*', he indeed refers to memory bus or the ones used in computer to perform data transfer. A general purpose retrieval mechanism would retrieve documents pertaining to vehicle buses, which are totally irrelevant to the context of memory buses. Using a CDRM, the engineer will be able to present the namespace of the search as computing in the form of a relevant document, and all the results to his queries will be compared with the guide for relevance before being ranked for presentation.

2. When a manufacturer queries a web search engine for '*black belt*', he is likely to be flooded with documents that relate to the one worn around the waist to support clothing or a level of proficiency with martial arts, whereas his actual intention is to retrieve documents pertaining to *Six Sigma*. By providing a relevant Six Sigma document as guide, the manufacturer will be able to narrow his search to the namespace of his choice using CDRM and retrieve better matching results.

* * *

Chapter 4

ARCHITECTURAL FRAMEWORK

This chapter provides details on the architectural framework proposed for building metasearch engines with support for dynamic configuring of component search engines and content-directed result merging.

4.1 Requirements

A framework that supports building of metasearch engines that can handle eclectic information sources and still maintain integrity has to be highly modular; provisions for database selection, query reformulation, workload management, scheduling, result merging, re-ranking and dynamic reconfiguration should be present. Extending the framework to support new communication protocols should be easy. Facilities to support addition or removal of components and dynamic switching of component parameters should be available. Framework should provide architectural support for feedback techniques like content-directed result merging and be flexible enough to extend support for unstructured information sources like multimedia. The architecture should lend itself well to realizing the metasearch engines in a parallel or clustered environment. The framework should have an integrated exception handling mechanism and a unified logging scheme.

4.2 Design

With the aforesaid requirements in mind, we propose a simple to use, flexible metasearch framework, consisting of four components: segregator, scheduler, aggregator, and search providers. The configuration of the metasearch engine is maintained in an XML

document. Making use of a standard representation scheme like XSD (XML Schema Document) to specify the configuration keeps the complexity levels under control yet provides a powerful way of extending the features of the framework.

The input to the engine or the query will be a data artifact, which can take several forms including documents, data streams, literals, etc. The output of the engine is an XML document that can be rendered as a data mash up presented over a user interface (UI) of choice. Figure 4.1 illustrates the metasearch processing using a block diagram

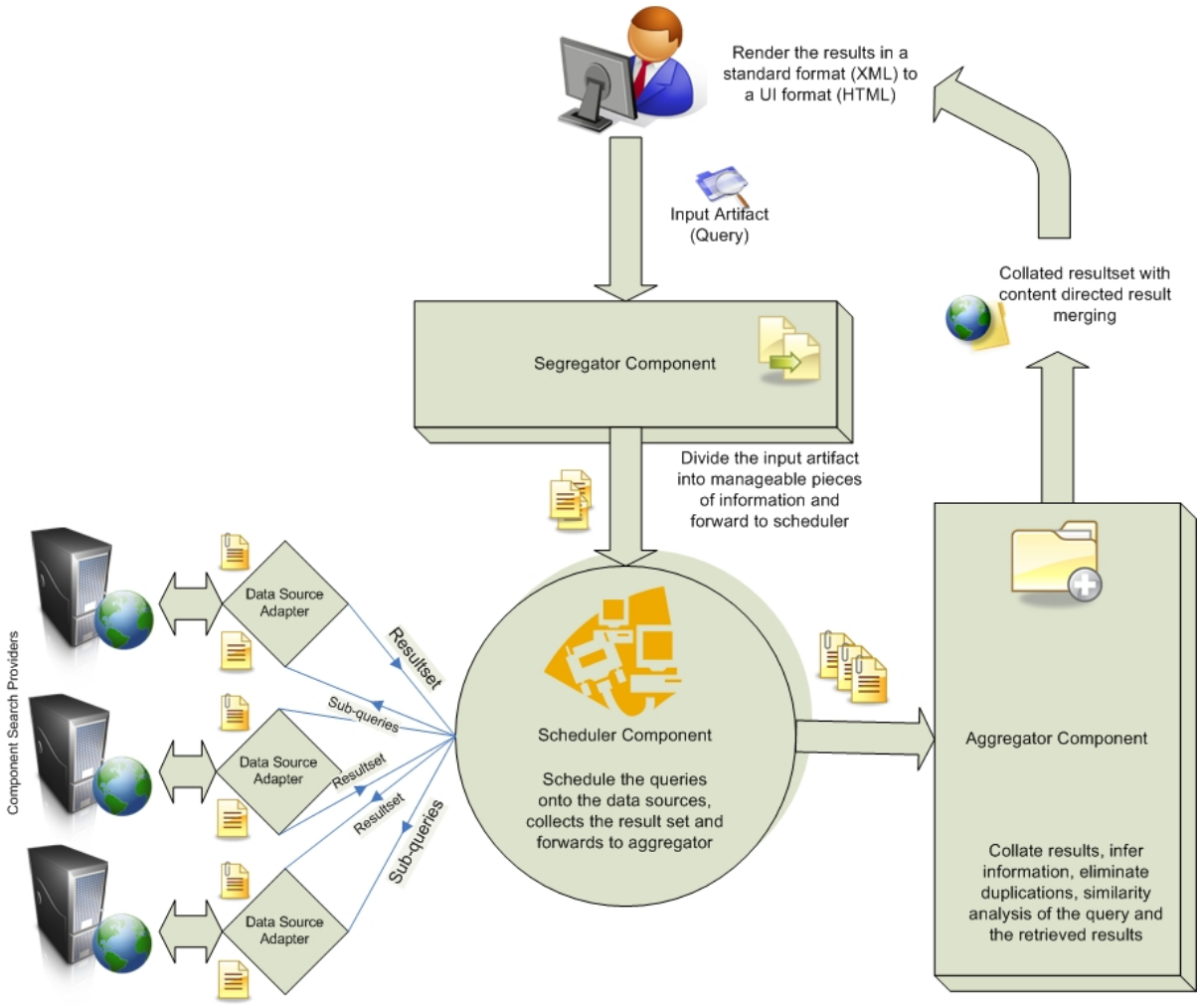

Figure 4.1 Metasearch Framework

4.2.1 Query Segregator

First the data artifact flows into the segregator component which takes the data artifact from the user as input and optionally divides it into manageable pieces based on size and nature of content. A more feature rich segregator may go beyond simple tokenization and will provide support for syntactic parsing, named entity detection, classification, summarization and translation. Segregator implementations will leverage the functions of existing open source search frameworks to build sensible sub-queries for the scheduler component. For example when it comes to text content in the artifacts, we can use Apache Lucene [3] to extract well-formed queries out of the textual content. If no query preprocessing or reformulation is intended, then we can use a segregator stub to just forward the query to the scheduler, which is the next component.

4.2.2 Query Scheduler

The second component of the metasearch processing is the scheduler. Scheduler is essentially a query manager which takes preprocessed input artifact from the segregator and schedules the search on several configured data sources in parallel. Scheduler is highly configurable by the users as they can choose the repositories they would like to search by including or excluding them from the configuration document and specify variety of options for each search provider. An advanced scheduler can go beyond simple literal scheduling and provide support for media content as well; in such a case, scheduler has to be aware of the search providers that support the required media formats. The results from the individual search providers are gathered and forwarded to the aggregator for post processing, which is essentially rank aggregation and result merging. The input artifact is also forwarded in its

original form to facilitate re-ranking. Because turnaround time of search providers can vary significantly, synchronizing the dynamic search provider threads can be the most challenging aspect of the scheduler design. In order to meet the performance goals, the scheduler may use an underlying GRID or cluster to schedule search tasks over a distributed infrastructure.

4.2.3 Result Aggregator

Aggregator takes care of collating the results into a consolidated result set. Other responsibilities of aggregator may include elimination of irrelevant contents and duplicates (filtering), and reformatting the contents. In addition, aggregator is the component where optional re-ranking of results can be performed using a feedback technique like click-through data or content guidance; in such a case, aggregator acts as the inference engine by categorizing the content and establishing relationships between the source and the search results from different parts of the content. Several advanced features have to be incrementally added to the aggregator to refine the result set. The aggregator also takes care of packaging and formatting the source and extracted result set into a standard output format, like XML, that can be easily cast by web or application user interfaces.

4.2.4 Search Provider

Search providers are components which accept a set of query artifacts and return local search results in a standard format. Providers can take the form of extensions to database driver implementations, which help retrieving structured information from databases, or adapters to search indexes built over a set of unstructured documents by applications based on Lucene, or drivers to a content repository like Apache Jackrabbit [32] or even web services that provide definitions or search facilities over public databases [27]. The

framework delegates the communication and protocol management to the data source adapters there by providing easy interfaces to extend the framework for diverse information sources.

4.3 Comparison with Component Architecture

In comparison with the general metasearch component architecture (discussed in section 2.3), we can observe that the database selector and document selector roles are fulfilled by the parameters in the framework configuration file. This helps reduce the complexity of designing complex database and document selector modules and keeps the implementation simple. The query dispatcher role is taken over by the scheduler component and result merger role by the aggregator. The segregator serves to support the framework configuration in document selection and query preprocessing. The service providers are the interfaces to the search engine repositories.

In the next chapter we will explain the reference implementation based on this architecture and a prototype metasearch engine that we have built using the reference implementation.

* * *

Chapter 5

IMPLEMENTATION

This chapter deals with the implementation details pertaining to the metasearch framework. We first discuss the infrastructure used, and then get into the details of the component implementations and content-directed result merging, and finally, describe a metasearch prototype based on the reference implementation.

5.1 Implementation Infrastructure

The framework and all involved components have been developed using Java programming language in Eclipse environment. The object oriented features of Java helps to keep the framework flexible and easy to extend. Eclipse [24] plug-in architecture provides an integrated development environment for Java and necessary library support for the development. Figure 5.1 provides a brief overview of the infrastructure implementation.

Quiet often component search engines are made available to the user community in the form of web services. Search service providers issue Web Service Description Language documents (WSDL) that details the types and functions available through the service. We have used Apache Axis2 [25], which is a robust web services engine, to provide the necessary infrastructure support for creating service clients to access component search services.

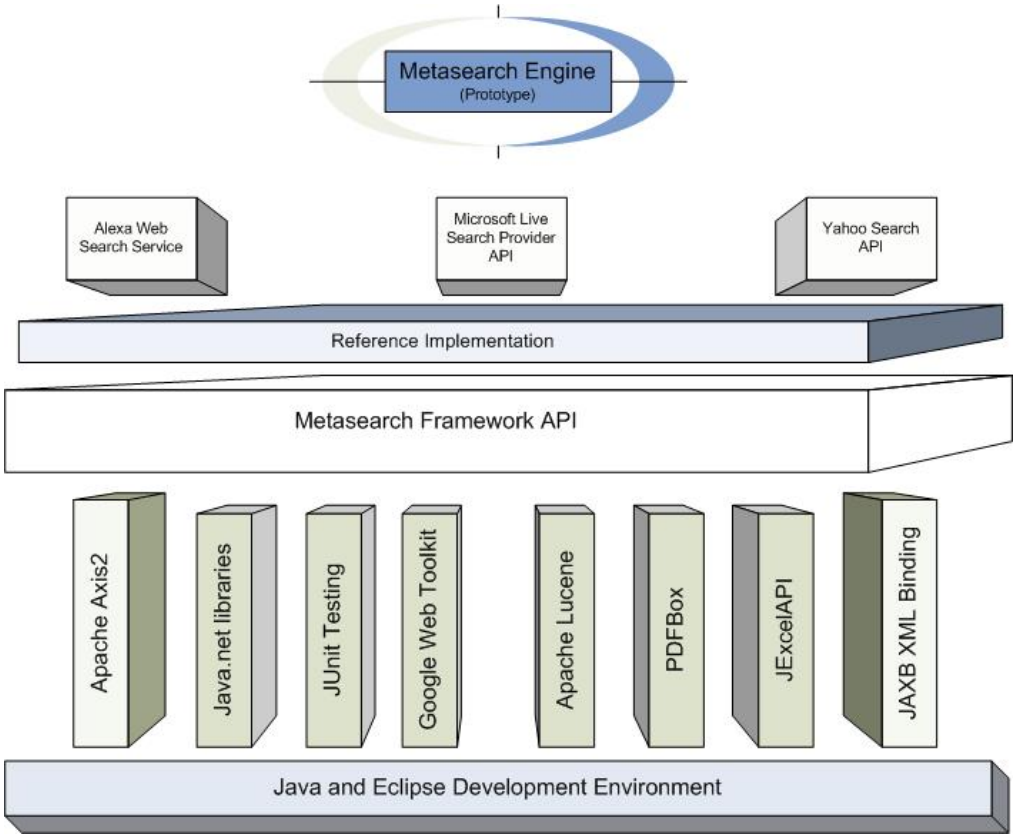

Figure 5.1 Implementation Infrastructure

Apart from web services, we also make extensive use of Java's networking libraries for service provider communications.

We also make extensive use of XML documents inside the framework. Most of the document exchanges with service providers render XML documents. Also the configuration and result schema documents are in XML format. To facilitate the development and upkeep the maintainability of the framework we have used the Java and XML Binding (JAXB) reference implementation [26] to bind XML schema documents into classes and objects. This helps us focus on the algorithms rather than the exchange formats and schema standards.

Apache Lucene [3] provides the necessary indexing and text search support for our framework. Lucene is a high-performance, cross-platform, full featured text search engine

library. We use its functionality to remove redundant and unnecessary terms and literals from the query, and especially in content-direction.

For unit testing we have used the JUnit [28] framework. Through the support provided by this framework we have developed individual test cases for most of the components and have developed some test suites that test the entire system. Performance evaluations are done using the JUnit tests.

Java documentation (javadoc) utility is used to generate API (Application Programming Interface) documentation for the framework. Google Web Developer Toolkit (GWT) [29] is used to generate AJAX user interface for the prototype. Portable Document Format (PDF) support is provided by the PDFBox [31] library. Reporting infrastructure is provided by the JExcelAPI [30].

Apache Log4J is used for logging in the application and Subversion is used for version control.

5.2 Framework

A metasearch operation can be initiated by creating an object to the `DREFramework` class, setting a valid configuration to the framework object in the form of input XML configuration document or `DREConfiguration` object and invoking the `processArtifact()` method. The configuration is automatically validated while it is being set. Framework exceptions are internationalized and exception handlers are placed in the `edu.ncsu.dre.exception` package. File and type comparison utilities are placed in the `edu.ncsu.dre.utils` package. Figure 5.2 illustrates the entire class hierarchy of the metasearch framework.

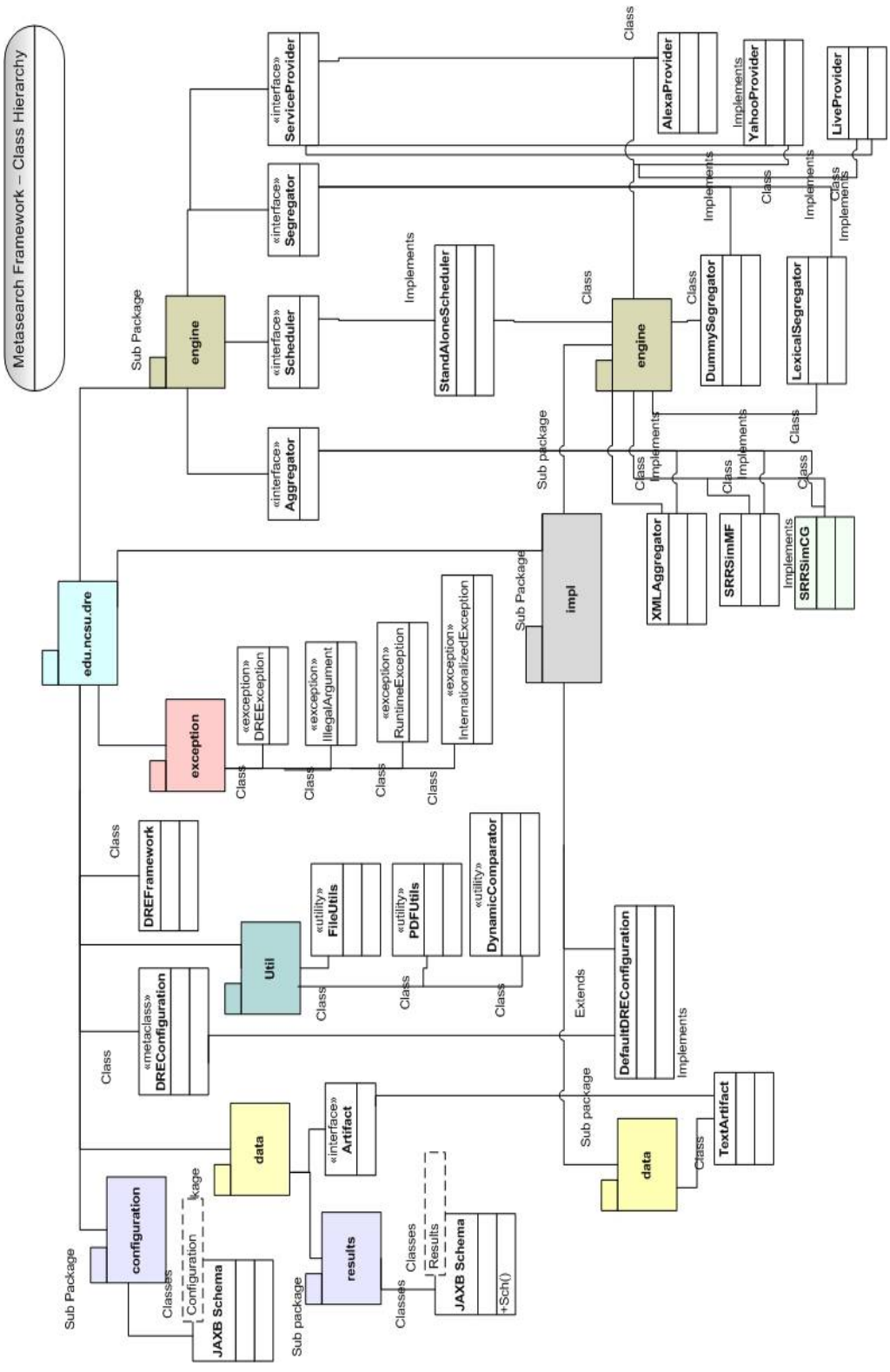

Figure 5.2 Class hierarchy of Metasearch Framework

5.2.1 Configuration

The configuration sets the framework components used for the metasearch operation; configuration can be either loaded from an XML document that follows the schema `configuration.xsd` (shown in Figure 5.3) or by creating an object to `DREConfiguration` class. A static binding of the XML schema objects is done using JAXB; this allows runtime modifications to the configuration. Users can use their own segregator, aggregator, scheduler,

Figure 5.3 Configuration XSD

and their own list of service providers by modifying the configuration document. Arguments to the framework and individual components can be set using the parameters and option hash tables respectively. Arguments are represented as (key, value) pairs are directly available to the framework and its components once the configuration is loaded. Figure 5.4 shows a sample configuration for the SRRSimCG with content guidance document.

```
<?xml version="1.0" encoding="UTF-8"?>
<EngineConfiguration>
  xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
  xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="configuration.xsd">
  <Segregator>
       <Handler>edu.ncsu.dre.impl.engine.DummySegregator</Handler>
       <Option key="Comment" value="This is the LexicalSegregator"/>
  </Segregator>

  <ResearchScheduler>
       <Handler>edu.ncsu.dre.impl.engine.StandAloneScheduler</Handler>
       <Option key="Comment" value="This is the StandAloneScheduler"/>
  </ResearchScheduler>

  <Aggregator>
       <Handler>edu.ncsu.dre.impl.engine.SRRSimCGAggregator</Handler>
       <Option key="Comment" value="This is the SRRSimCGAggregator"/>
  </Aggregator>

  <ServiceProvider>
       <Handler>edu.ncsu.dre.impl.engine.LiveSearchProvider</Handler>
       <Option key="ID" value="Livesearch"/>
       <Option key="AppID"     value="7E8E2A6CDEEE7248E0EBF23EDD20303F86364CCE"/>
       <Option key="Culture" value="en-US"/>
       <Option key="SafeSearch" value="Off"/>
       <Option key="Source" value="web"/>
  </ServiceProvider>

  <ServiceProvider>
       <Handler>edu.ncsu.dre.impl.engine.YahooSearchProvider</Handler>
       <Option key="AppID"
value="eCBIC3LV34EN3FHl35AMrMhA7JoOi4jfPPy1VQrSNr51qWbeO43DkDDLSqG0jmVg"/>
       <Option key="ID" value="Yahoo"/>
  </ServiceProvider>

  <ServiceProvider>
       <Handler>edu.ncsu.dre.impl.engine.AlexaSearchProvider</Handler>
       <Option key="ID" value="Alexa"/>
  </ServiceProvider>

  <Parameter key="Comment" value="SimpleStandAlone DRE Framework."/>
  <Parameter key="ContentGuide" value="mem-sched.pdf"/>
</EngineConfiguration>
```

Figure 5.4 Sample Framework Configuration

5.2.2 Artifact

Artifact interface (`edu.ncsu.dre.data.Artifact`) provides an outline for the data and structure of the query that will be run on the metasearch. An abstraction is made here so that the framework can be easily extended to support multi media content. Normal text queries can directly cast their contents into artifact using the TextArtifact (`edu.ncsu.dre.data.impl.TextArtifact`) class.

5.2.3 Artifact Segregation

Segregator (`edu.ncsu.dre.engine.Segregator`) is the common interface for all components that perform preprocessing on the artifact before it is scheduled for search in the scheduler. Preprocessing can take a variety of forms like modifying the order of query terms, adding operators, splitting the task into sub queries etc. The processed artifact and its subsets if any are placed in a Collection (frequently a list structure) and passed to the scheduler for scheduling. We have included two segregator implementations with the framework namely DummySegregator (`edu.ncsu.dre.engine.impl.DummySegregator`) and LexicalSegregator (`edu.ncsu.dre.engine.impl.LexicalSegregator`). DummySegregator does no preprocessing but just forwards the query to the scheduler. LexicalSegregator lexically analyses the query using Apache Lucene and segregates the query terms for efficient search.

5.2.4 Metasearch Scheduling

The scheduler interface (`edu.ncsu.dre.engine.ResearchScheduler`) provides the structure for implementing a metasearch scheduler. The `scheduleResearch()` method

accepts a list of query artifacts and a list of component search providers. A typical scheduler implementation is expected to instantiate each of the component search providers as threads and passes the query artifacts to them one after another. The threads are polled one after another for completion, once completed the results are accumulated in a hash map data structure and returned to the aggregator for result merging and re-ranking, each artifact or a sub set of it is mapped with a list of SRR elements. The framework includes a standalone scheduler (`edu.ncsu.dre.engine.impl.StandaloneScheduler`) that implements the aforesaid functionality. More complex and distributed schedulers can be designed extending the standalone scheduler which is left for future work.

5.2.5 Result Aggregation

Result Aggregator (`edu.ncsu.dre.engine.Aggregator`) acts as the common stub to all aggregator components. They take the hash map containing the individual artifacts and corresponding results to perform merging operations on them and render the resulting aggregated ranked list as an XML stream. Aggregator implementation is where we embed the result merging algorithm that determines the efficiency of the metasearch scheme used. Framework is packaged with four of this; each providing a different functionality.

SimpleXMLAggregator (`edu.ncsu.dre.engine.impl.SimpleXMLAggregator`) renders the merged result as a standard XML document, simple result merging is used.

SimpleHTMLAggregator (`edu.ncsu.dre.engine.impl.SimpleHTMLAggregator`) renders the merged result as an XHTML document, which can be directly presented on a browser interface. Simple result merging is used here.

SRRSimMFAggregator (`edu.ncsu.dre.engine.impl.SRRSimMFAggregator`) renders the merged result as an XML document; the merging strategy used here is SRRSimMF which has been previously discussed.

SRRSimCGAggregator (`edu.ncsu.dre.engine.impl.SRRSimCGAggregator`) renders the merged result as an XML document; the merging strategy used here is SRRSimCG. Here the content guidance (CG) is provided as option to the framework and the aggregator uses this CG to rank the results.

5.2.6 Search Provider support

The service provider interface (`edu.ncsu.dre.engine.ServiceProvider`) extends the Java thread interface and provides the necessary ground work for constructing a search service provider class. Threaded implementations of this interface are expected to return a hash map, containing the artifacts as keys and the corresponding results as values, when invoked with the function `gatherInformation()`. We have implemented three search providers in our framework: Alexa Search Provider (`edu.ncsu.dre.engine.impl.AlexaSearchProvider`), Microsoft Live Search Provider (`edu.ncsu.dre.engine.impl.LiveSearchProvider`), and Yahoo Search Provider (`edu.ncsu.dre.engine.impl.YahooSearchProvider`). They retrieve the results from Amazon's Alexa web cache, Live search and Yahoo repositories respectively.

5.2.7 Content-direction support

Content-direction support is built into the framework. When user provides a parameter as Content Guide to the engine configuration, it is available to all the components across the framework. An aggregator implementation that uses the content guidance builds a

list of literals and terms using Apache Lucene library to build several lists, each containing matching terms from the query, content guide and SRR's. The lists aid in computing distinct match counts, match counts, and simple counts of literals and terms between the query, guide and results that are being compared. Using these computed values we can directly calculate the similarity function and rank of SRR using the expressions given in section 3.3.2.

5.3 Metasearch Prototype

Based on this metasearch framework, we have built a prototype metasearch engine using the Google Web Developer Toolkit. The toolkit provides the servlet container environment, server and client stubs and the necessary AJAX interface development framework. The end user has a web form, through which he can set the content guidance document as a PDF or paste guidance text (the content guidance step is optional). When queries are input through the form, the merged results are rendered as an HTML document and presented on the browser interface.

5.3.1 User Interaction Model

Users have two ways of using the prototype. They can use the simple search feature to query the component search engines without content-direction or they can set the content-direction prior to a search and then perform a query which will order the results guided by the content-direction document. The available user interaction methods are illustrated in the figure 5.5

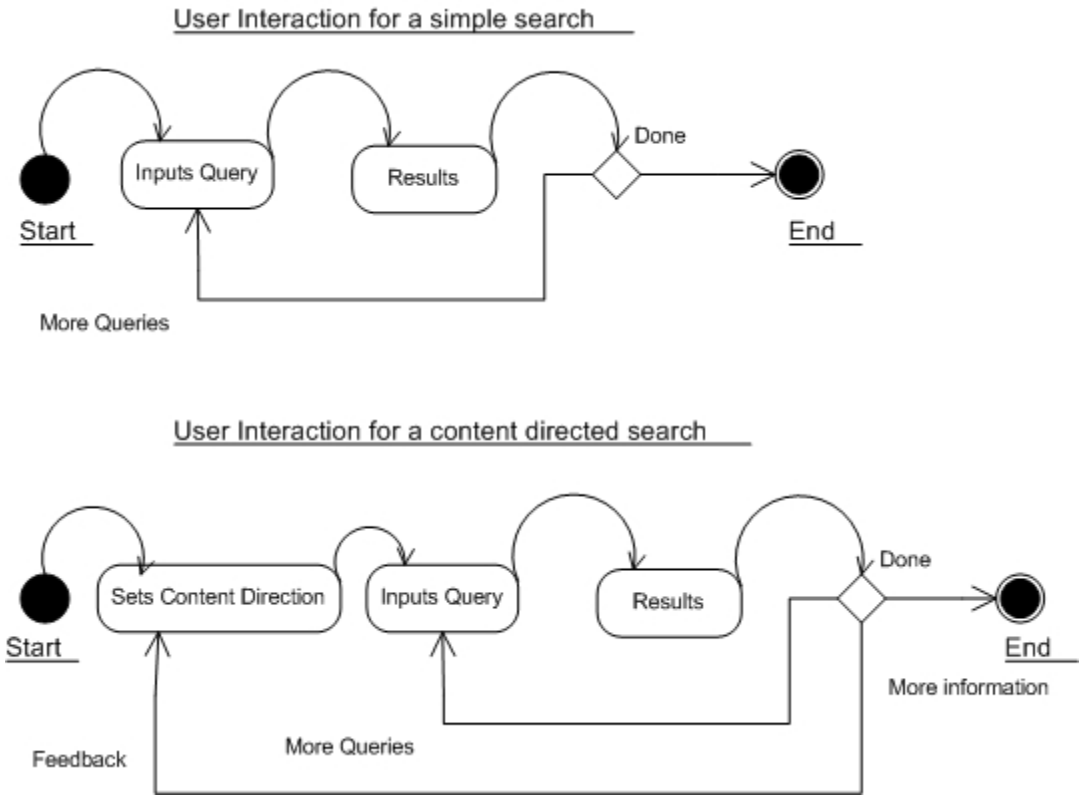

Figure 5.5 User interaction model

We have used this prototype and reference implementation to evaluate the performance of content guidance in effectively ranking the search results. The next chapter details the evaluation strategies used and the results obtained.

***

Chapter 6

EVALUATION and RESULTS

This chapter provides a brief outline on the evaluation standards available to gauge the effectiveness of information retrieval systems, specifically metasearch engines. Based on these standards we evaluate the effectiveness of our metasearch engine with content-directed result merging.

6.1 Evaluation Standards

The Text REtrieval Conference (TREC) organized by National Institute of Standards and Technology (NIST) sets standards to measure the retrieval effectiveness of an IR system [19].

6.1.1 TREC Tracks

Each TREC workshop consists of a set of tracks or areas of focus in which particular retrieval tasks are defined. The tracks act as incubators for new research ideas in information retrieval. They define the information retrieval problem and provide the necessary infrastructure and framework (test collections, evaluation methodology etc) to support the retrieval research on its task. Each track is supported by a user community linked by a mailing list. The community provides recommendations to TREC program committee and researches on improvements to the tracks that will be run in a given year of TREC.

There are specific TREC tracks for each kind of problem in an IR context. Below are few TREC tracks with their descriptions that give us a broad idea of the track evaluation framework and the IR problems they attempt to solve.

a) Blog Track [2007] – Aims to explore information seeking behavior in blogosphere

b) Enterprise Track [2007] - Track to study enterprise search, satisfying a user who is searching the data of an organization to complete some task.

c) Genomics Track [2007] – Study retrieval functions with respect to genomics data

d) Legal Track [2007] – Effective retrieval of digital documents collection

e) SPAM Track [2007] – Standard evaluation of current and proposed SPAM filtering approaches and retrieval tasks.

f) Terabyte Track [2006] – Investigates the scalability issues with IR test collection based evaluation to significantly larger document collections than those used in TREC.

g) HARD Track [2005] – High Accuracy Retrieval from Documents by leveraging additional information about the searcher and/or the search context.

h) Robust Retrieval Track [2005] – Focus is on individual topic effectiveness rather than average effectiveness.

i) Web Track [2002, 2004] – Features search tasks on a document set that is a snapshot of the World Wide Web.

6.1.2 TREC Web Track

Among the aforesaid tracks the Web Track is the one that is most relevant to our metasearch result merging evaluation. The 2002 TREC Web Track provides a set of queries or topics and a huge repository of documents containing a crawl of the Internet done in early 2000. Each web topic consists of four parts: an index number, a title, a description and a narrative. The track defines 50 topics indexed from 551 to 600. An example Web Track topic is shown in Figure 6.1

```
<webtrack>
 <topic>
  <num> Number: 551 </num>
  <title> intellectual property</title>
  <desc> Description: Find documents related to laws or regulations that
protect intellectual property </desc>
  <narr> Narrative: Relevant documents describe legislation or federal
regulations that protect authors or composers from copyright infringment ,
or from piracy of their creative work. These regulations may also be related
to fair use or to encryption.</narr>
 </topic>
</webtrack>
```

Figure 6.1 Sample Web Track

6.1.3 TSAP Methodology

We cannot be using the general purpose web track directly to evaluate our result merging algorithm -- the web track is based on fixed repository of documents whereas we are using general purpose search engines that index the latest documents of the Internet as component service providers. Instead we will use the web track topics to query our metasearch. Only title part is used as a query for metasearch. As titles are short, they are representative of Internet queries submitted by real users. The average length of the titles of these fifty topics is 3.06. The narrative describes what documents should be considered relevant to the corresponding query topic. The information in the narrative is used as the standard criteria for us to judge the relevancy of the collected result documents.

As it is difficult to know all the relevant documents to a query in a search engine, the recall and precision metrics used for evaluating IR systems cannot be used for evaluating internet search or metasearch engines. A well known alternative to performing this type of evaluation on search engines is the TREC-style average precision (TSAP) [20]. As like previous studies [18] that propose newer result merging algorithms we use TSAP at cutoff N, denoted as TSAP@N to evaluate the effectiveness of each result merging algorithm.

$$TSAP@N = \left(\sum_{i=1}^{N} r_i \right) / N$$

Here,

Where $r_i = 1/i$ if the i-th ranked result is relevant and $r_i=0$ if the i-th result is not relevant. The TSAP@N metric considers both the number of relevant documents in the top N results and the ranks of each of the relevant documents and yields a larger value when the results are more relevant.

6.1.4 Test Bed and Evaluation Strategy

We use our prototype based on the reference implementation as our test bed. Three component search engines are used in the test bed namely the Microsoft Live Search [21], Yahoo Search [22] and Amazon's Alexa Web Search [23]. We run the queries from 2002 Web track using scripts for four different configurations. In the first configuration we run queries on component search engines in the normal context and compute TSAP@N[3] for each of them. In the second configuration we run queries on each of the component search engines with web track narration as the content-direction for each query and compute TSAP@N.

In the third configuration we run all component search engines together in a metasearch context with SRRSimMF (i.e. SRRSimCG without content-direction) result merging strategy and compute TSAP@N. Finally we run web track topic queries in the metasearch context with query narration as the content-direction for each query and compute TSAP@N. We compare TSAP@N for all these configurations to measure the relevance improvements.

[3] N = 5 or N = 10

6.2 Results

6.2.1 TSAP@N and Retrieval Effectiveness

Table 6.1 lists the TSAP@N for each of the component search engines and the composite metasearch engine, with and with out content-direction. Figure 6.2 illustrates Table 6.1 as a bar chart representation.

	Alexa	Live	Yahoo	MetaSearch
TSAP@10	0.18867619	0.232587302	0.249544	0.241180952
TSAP@10 WithCG	0.214880159	0.253565079	0.268646	0.272753968
TSAP@5	0.286533333	0.363866667	0.393	0.3806
TSAP@5 WithCG	0.347533334	0.414466667	0.437333	0.429733334

Table 6.1 Comparison of Retrieval Effectiveness using TSAP

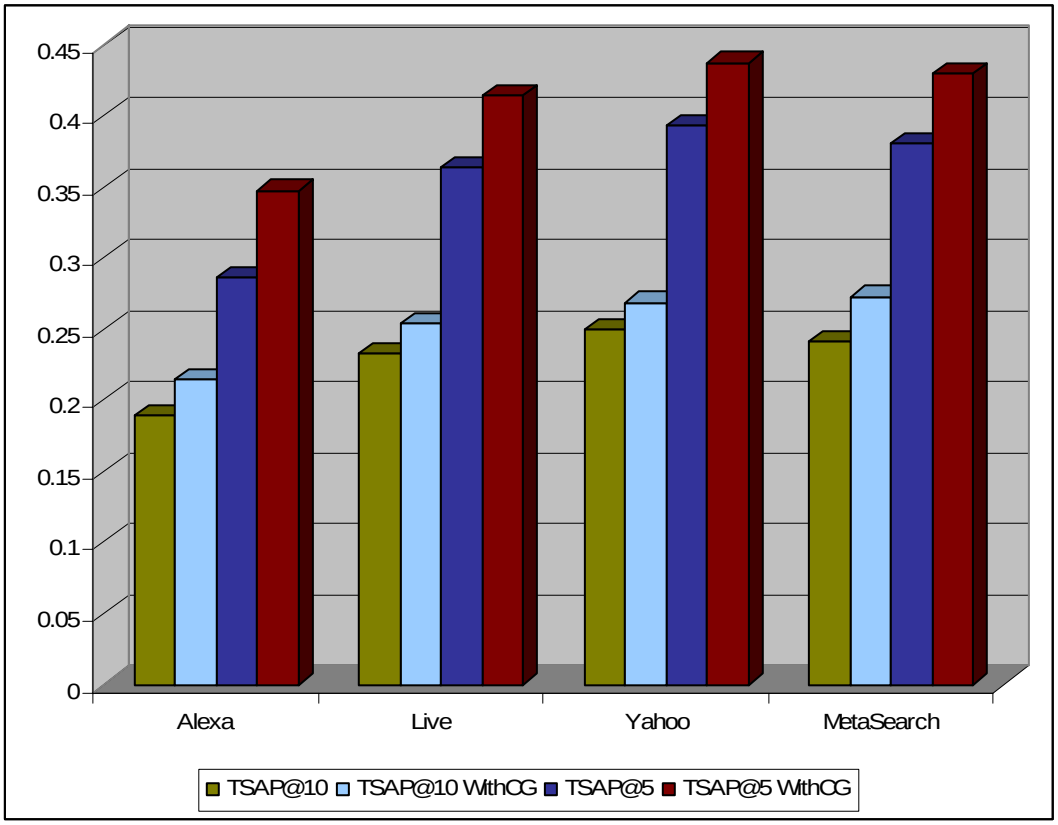

Figure 6.2 Retrieval Effectiveness using TSAP for different configurations

Figures 6.3 and 6.4 illustrate TSAP@5 and TSAP@10 respectively for Amazon's Alexa web search service with and without Content-direction. Relevance scores of Alexa have been found to be the least among the compared engines. Content direction improves Alexa's relevance of results for the top five SRR's by 21% and top 10 SRR's by 14%.

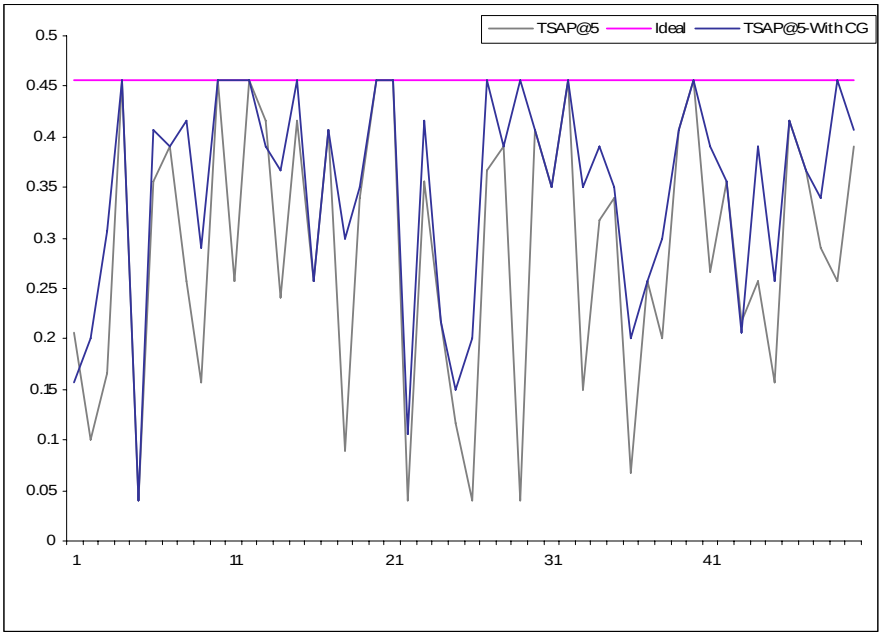

Figure 6.3 TSAP@5 for Amazon Alexa Web Search with/without Content-direction

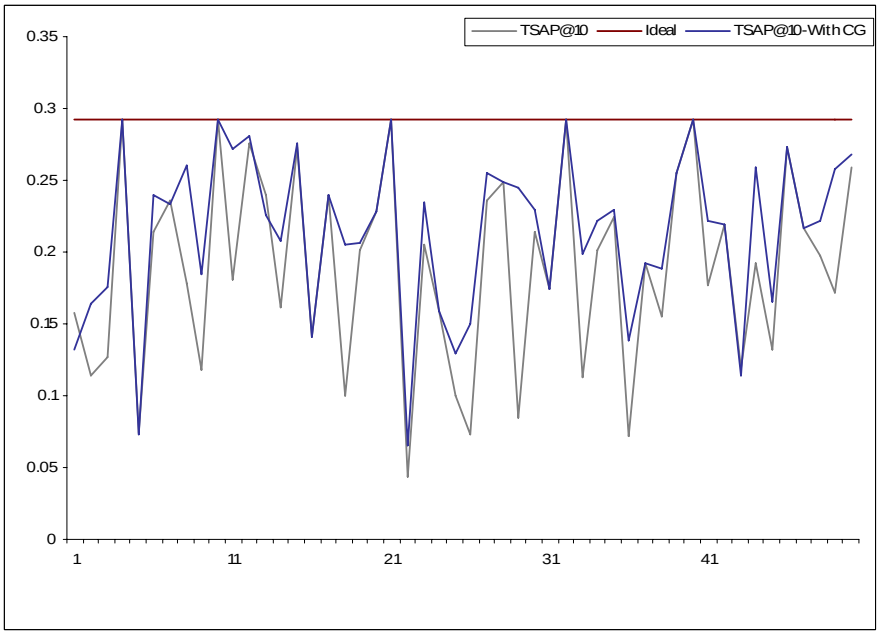

Figure 6.4 TSAP@10 for Amazon Alexa Web Search with/without Content-direction

Figures 6.5 and 6.6 illustrate TSAP@5 and TSAP@10 respectively for Microsoft Live search with and without Content-direction. Content direction improves Live search's relevance of results for the top five SRR's by 9% and top 10 SRR's by 14%. There are certain data points that illustrate poor performance of content-direction than the relevance of results without it.

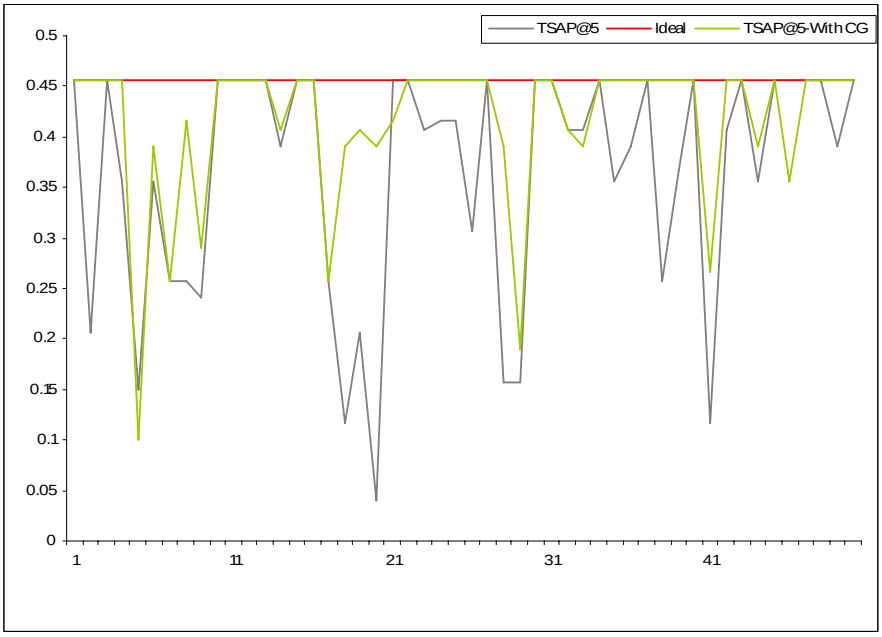

Figure 6.5 TSAP@5 for Microsoft Live Search with/without Content-direction

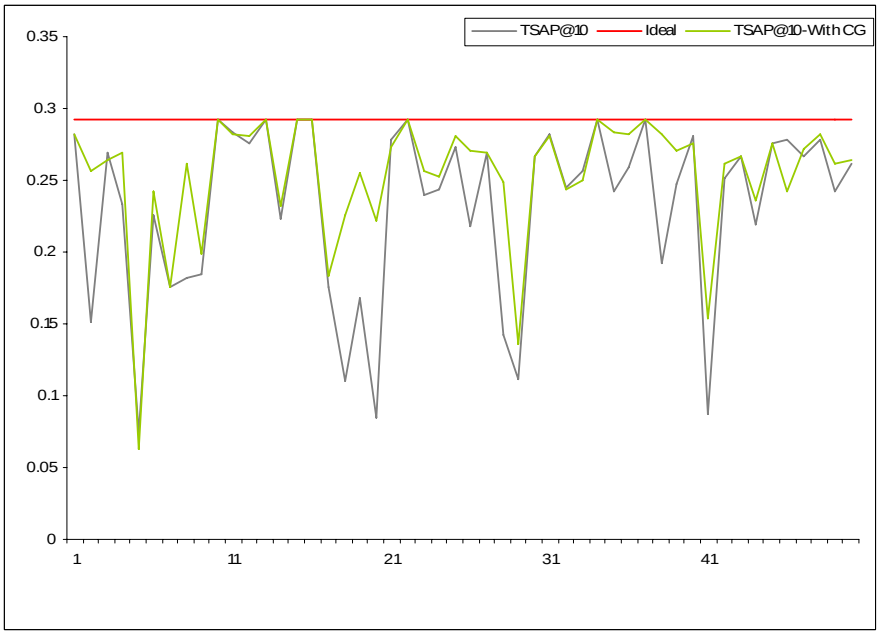

Figure 6.6 TSAP@10 for Microsoft Live Search with/without Content-direction

Figures 6.7 and 6.8 illustrate the TSAP@5 and TSAP@10 respectively for Yahoo search with and without Content-direction. Yahoo provides the best relevance results; harmful effects of content-direction are more prominent here but overall content-direction to provide better relevance of results. Relevance gain of first 5 SRR's is 7.5% and for all 10 SRR's is 11%.

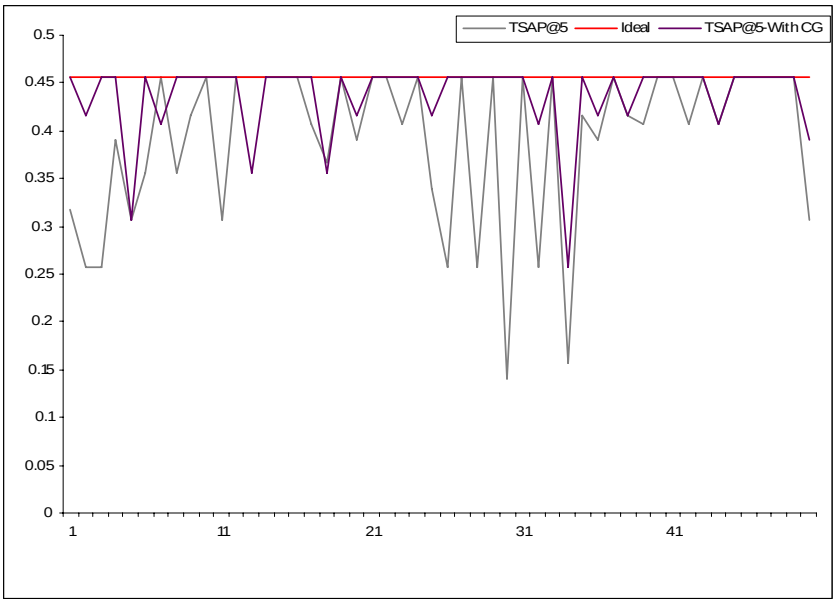

Figure 6.7 TSAP@5 for Yahoo Search with/without Content-direction

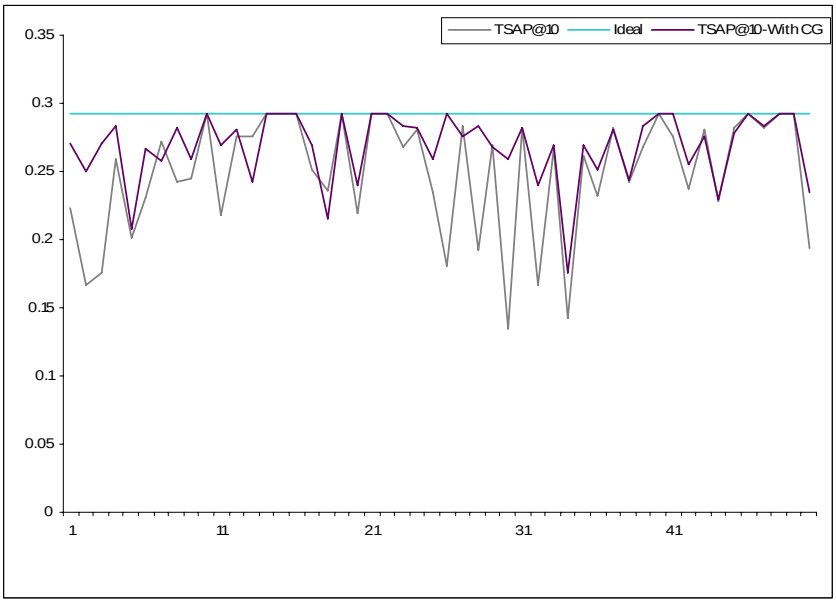

Figure 6.8 TSAP@10 for Yahoo Search with/without Content-direction

Figures 6.9 and 6.10 illustrate the TSAP@5 and TSAP@10 respectively for Metasearch with and without Content-direction. Results are comparable to the best performing component engine (here Yahoo). Both TSAP@5 and TSAP@10 gain 13% improvement in relevance of results.

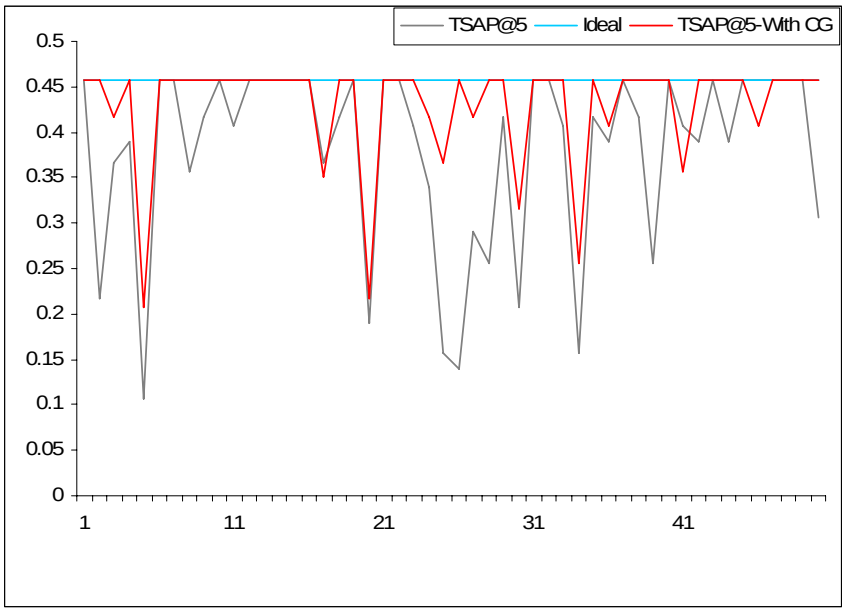

Figure 6.9 TSAP@5 for Metasearch with/without Content-direction

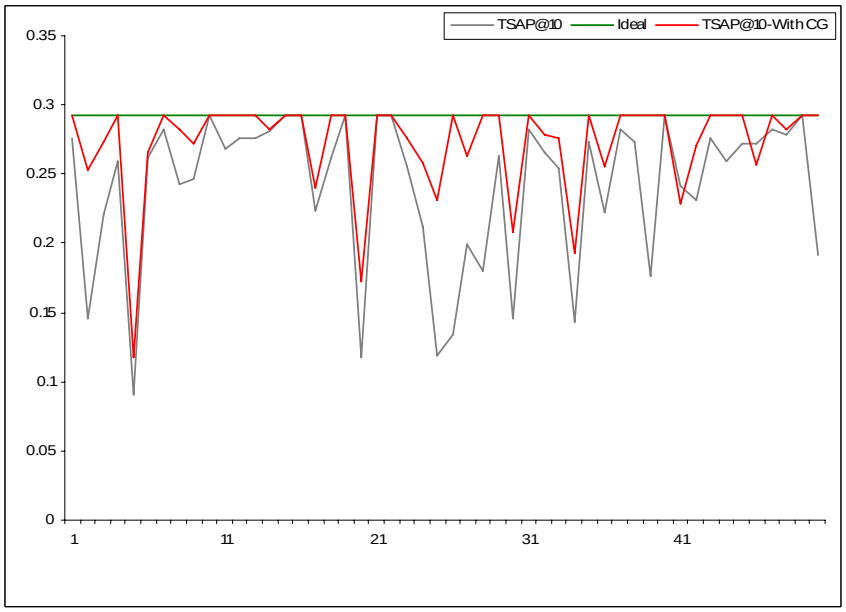

Figure 6.10 TSAP@10 for Metasearch with/without Content-direction

6.2.2 Discussion

From the results, we observe that the content-direction boosts the relevance of results uniformly for individual search engines as well as metasearch engines. The improvement is more prominent where the search engine suffers from poor relevance ordering (Alexa). The mechanism is particularly helpful for metasearch engines to improve the relevance ordering where more results are available for ordering. The selection of most relevant results for metasearch from a pool of SRR's with content-direction provides a performance comparable to the best performing search engine.

The content-direction effect is not always positive; there are certain queries in which the content-direction degrades the relevance of results. On closer observation, these queries had content-direction narrations specifying the results which needed to be excluded from the results. As we are not doing a semantic or logical analysis of the content-direction input the system has interpreted these directions erroneously and hence the aberration. Semantic analysis of content-direction prior to use in relevance ordering is left to future work.

In this chapter we presented the evaluation strategy, experimental setup, results that discuss the impact of content-direction on relevance ordering. We conclude in the next chapter with a discussion of future work.

***

Chapter 7

CONCLUSION AND FUTURE WORK

7.1 Conclusion

Metasearch engines are distinguished from conventional search mechanisms, by their focus on information retrieval from eclectic sources. In this thesis we have investigated the implementation of metasearch engines, important challenges in result merging, and a new content-directed method for providing highly relevant search results.

We have designed a modular framework based on service-oriented architecture to build metasearch engines. The framework supports dynamic addition and removal of participant search providers. A reference implementation of the framework has been done in Java, to demonstrate flexibility of the architecture. Individual components like segregator, scheduler and aggregator can be seamlessly reconfigured to produce highly customized search systems without significant effort. A prototype metasearch engine has been developed to demonstrate the flexibility of the framework with three commercial search service providers as component search engines.

This thesis proposed content-directed result merging algorithm for rank aggregation in metasearch engines. We have described how user queries, augmented by additional information can be leveraged to provide highly relevant search results. We believe that ideas of content-direction are interesting in the context of information retrieval systems. Content-direction provides additional feedback to the system, enabling the system to make informed decisions while ranking the results based on relevance and providing an opportunity for filtering the irrelevant ones. Content-direction provides an easier and faster way of narrowing

down search results of interest in a particular domain without repeating the search with several complex queries. The success of content-direction in result merging and rank aggregation has been demonstrated using the metasearch prototype.

Next generation search technologies are increasingly focusing on the personalization of search, to handle the explosive growth of information in all walks of human knowledge. This thesis has focused on improving information retrieval through metasearch engines that provide relevant results using user provided content-direction.

7.2 Future work

The metasearch framework can be a good choice for testing new approaches in providing highly relevant search results. The framework itself can be augmented and extended in several ways. New segregation algorithms that take structured queries for processing can be formulated. Similarly, advanced aggregation algorithms that learn from implicit feedback can be devised to provide highly relevant search results apart from content-direction. New component search schedulers can benefit by utilizing an available grid or clustering infrastructure to efficiently use the network resources. The metasearch framework is also an excellent platform to compare relative performances among a set of search engines.

Several improvements can be done to the content-direction methodology as well. The system still does not support content that negatively impacts the queries. With rapid information explosion, quick filtering of the unrelated results based on the content-directed similarity function can be one of the future directions of interest. There can also be extensions of content-direction usage, along with other implicit feedback mechanisms to personalize search engines for every user.

REFERENCES

[1] Michael K. Bergman, 2001, The Deep Web Surfacing Hidden Value, http://www.press.umich.edu/jep/07-01/bergman.html

[2] W. Meng, C. Yu, and K. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 2002.

[3] Apache Lucene, 2006, http://lucene.apache.org/

[4] JUnit.org Resource for Test Driven Development, 2007, http://www.junit.org/

[5] T. Joachims, F. Radlinski, Search Engines that Learn from Implicit Feedback, *IEEE Computer*, Vol. 40, No. 8, August, 2007.

[6] Gerald Salton, Michael J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill Book Company 1983.

[7] Cleverdon, C. 'Optimizing convenient online access to bibliographic databases' In: P. Willet (ed.), *Document retrieval systems*, London, Taylor Graham, 1988 32-41.

[8] Al-Hawamdeh, S. and Willet, P., 'Comparison of index term weighting schemes for the ranking of paragraphs in full-text documents', *International journal of information and library research*, 1, 1989, 116-30

[9] Salton, G. and Buckley, C., 'Term-weighting approaches in automatic test retrieval', *Information processing and management*, 24(5), 1988, 513-23

[10] Salton, G. and Buckley, C., 'Improving retrieval performance by relevance feedback', *Journal for American Society for Information Science*, 41(4), 1990, 288-97

[11] D. Dreilinger, A. Howe. Experiences with Selecting Search Engines Using Metasearch. *ACM TOIS*, 15(3), July 1997, pp.195-222.

[12] E. Selberg, and O. Etzioni. The MetaCrawler Architecture for Resource Aggregation on the Web. *IEEE Expert*, 1997.

[13] J. Aslam, M. Montague. Models for Metasearch. *ACM SIGIR Conference*, 2001, pp.276-284.

[14] B. Yuwono, D. Lee. Server Ranking for Distributed Text Resource Systems on the Internet. *International Conference on Database System For Advanced Applications*, 1997, pp.391-400.

[15] E. Glover and S. Lawrence. Selective Retrieval Metasearch engine. *US Patent Application Publication* (US 2002/0165860 A1), November 2002.

[16] E. Selberg, and O. Etzioni. The MetaCrawler Architecture for Resource Aggregation on the Web. *IEEE Expert*, 1997.

[17] S. Robertson, S. Walker, M. Beaulieu. Okapi at trec-7: automatic ad hoc, filtering, vlc, and interactive track. *7th Text REtrieval Conference*, 1999, pp.253-264.

[18] Yiyao Lu, Weiyi Meng, Liangcai Shu, Clement Yu, and King-Lup Liu. Evaluation of Result Merging Strategies for Metasearch Engines . *6th International Conference on Web Information Systems Engineering (WISE05).*, pp.53-66, New York City, November 2005.

[19] TREC Tracks, http://trec.nist.gov/tracks.html

[20] S. Lawrence, and C. Lee Giles, Inquirus, the NECi Meta Search Engine. *Seventh International World Wide Web Conference*, 1998.

[21] Microsoft Live Search API 1.1, *Microsoft Live Developer Center*, http://msdn2.microsoft.com/en-us/library/bb251794.aspx

[22] Yahoo Search API, *Yahoo! Developer Network*, http://developer.yahoo.com/search/

[23] Alexa Web Search, *Amazon Web Services, Amazon.com,* *http://www.amazon.com/gp/browse.html?node=269962011*
[24] Eclipse *http://www.eclipse.org/*
[25] Axi2 *http://ws.apache.org/axis2/*
[26] JAXB Reference implementation, Sun Glass Fish Project, *https://jaxb.dev.java.net/*
[27] SearchSystems.net, *http://www.searchsystems.net*
[28] JUnit, Resources for test driven development, *http://www.junit.org/*
[29] Google Web Developer Toolkit (GWT) *http://code.google.com/p/google-web-toolkit/*
[30] Java Excel API, *http://jexcelapi.sourceforge.net/*
[31] PDF Box, *http://www.pdfbox.org/*
[32] Apache Jackrabbit, *http://jackrabbit.apache.org/*
[33] Naren Ramakrishnan, Search: The New Incarnations, *IEEE Computer,* August 2007
[34] A. Gulli and A. Signorini. Web is more than 11.5 billion pages. *WWW 2005*.
[35] Marcus P. Zillman, Deep Web Research 2007 Report, *LLRX.com*, 2007.
[36] H. He, W. Meng, C. Yu, Z. Wu, Automatic Extraction of Web Search Interfaces for Interface Schema Integration, *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, 2004.
[37] Inderjeet Singh ... [et al.], Designing Web services with J2EE 1.4 platform : JAX-RPC, SOAP, and XML technologies, *Addison-Wesley*, 2004.
[38] G.G. Chowdhury, Introduction to modern information retrieval, *Facet publishing*, 2004

***