COMPASS: A Concept-based Web Search Engine for HTML, XML, and Deep Web Data

Jens Graupmann, Michael Biwer, Christian Zimmer, Patrick Zimmer,
Matthias Bender, Martin Theobald, Gerhard Weikum

Max-Planck Institute for Computer Science
66123 Saarbruecken, Germany
{graupman, mbiwer, czimmer, pzimmer, mbender, mtb, weikum}@mpi-sb.mpg.de

1 Introduction

Today's web search engines are still following the paradigm of keyword-based search. Although this is the best choice for large scale search engines in terms of throughput and scalability, it inherently limits the ability to accomplish more meaningful query tasks. XML query engines (e.g., based on XQuery or XPath), on the other hand, have powerful query capabilities; but at the same time their dedication to XML data with a global schema is their weakness, because most web information is still stored in diverse formats and does not conform to common schemas. Typical web formats include static HTML pages or pages that are generated dynamically from underlying database systems, accessible only through portal interfaces.

We have developed an expressive style of concept-based and context-aware querying with relevance ranking that encompasses different, non-schematic data formats and integrates Web Services as well as Deep Web sources. Coined COMPASS (***C**ontext-**O**riented **M**ulti-Format **P**ortal-**A**ware **S**earch **S**ystem*), our system features this new language that combines the simplicity of web search engines with the expressiveness of (simple forms of) XML query languages ([7]).

2 Features of COMPASS

2.1 Concept-based Search

All web search engines are based on keyword-based search, but this style of query is strongly limited in its expressiveness. For example, neither a search for documents that contain either the keyword or one of its synonyms is possible, nor a search for documents in which one search term describes a concept or concept property while another one describes an instance value or a refinement of the first term.

Also, today's search engines are, for example, not capable of including Deep Web sources because they are unable to assign keywords to matching form fields. XML query languages, on the other hand, can support this feature by interpreting some keywords as element names corresponding to concepts, while interpreting other keywords as element contents corresponding to instance values, but are typically rather obscure to the average user.

As an example, consider a query about the book 'War and Peace' written by Tolstoy. If we can only use keywords to express this query, there is no way to specify any semantic relationship between these terms. In our system, we can express this query as follows: *title='war and peace' AND author=tolstoy.* Thus, we can use the semantic relationships between the terms for query processing purposes.

2.2 Context-aware Search

Another limitation of current web search engines is the fact that all keywords have to be matched on a single page. In reality, however, the desired information is often spread over multiple pages. Consider a query about a book store in our home town selling the book 'War and Peace'. Whereas the address of the book store (in our case 'Saarbrucken') is typically on the book store's home page A, a list of all available titles, including 'War and Peace', might be on a different page B that can be reached via a direct link from A. By following HTML-style href-links as well as XML-style XLinks, our system allows to exploit this link structure in order to find the combination of pages A and B as an appropriate query result.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

2.3 HTML, XML, and Beyond

XML is the desired format for all kinds of semistructured data. If XML were already ubiquitously used, the semantic relationships between query terms would be explicit by the hierarchical tree structure of the document. Thus, we apply heuristics to transform HTML pages and other web formats into semantically annotated XML documents. In most cases, simple heuristics can lead to meaningful XML data with a clearer structure than the original HTML documents. An example for our heuristics is the transformation of HTML structures like `<b>Title:</b>Tolstoy` into the following structure: `<Title>Tolstoy</Title>`.

We have implemented a framework with different modules to convert various data formats into XML. Currently these modules include HTML2XML and PDF2XML. Other modules are under development. These modules are based on heuristic rules, and may be combined with other information extraction approaches such as [1, 8].

2.4 Web Services and the Deep Web

Many information sources are not accessible via crawling. Rather, one needs to fill out a query form for retrieving documents that are dynamically generated from underlying database systems. The key difficulty in automatically generating meaningful queries against such web portals is to assign the appropriate values (which may be given merely as a set of keywords) to the available set of form fields. For this task, our system can analyzes the interface (e.g., the HTML form) to determine the available parameters ([4]). The result of the analysis is stored as meta information in a registry and used to generate wrappers that encapsulate the portal interface as a web service. Thus, each HTML form that we have successfully analyzed is represented by a WSDL interface in our system ([2]) and its information can be exploited when executing a query.

3 The COMPASS Query Language

The internal query language of COMPASS resembles a highly simplified version of mainstream languages like SQL, XPath, or XQuery. Search conditions refer to concepts and values, which correspond to element names and contents in an XML setting and attribute names and values in a SQL setting. Our query language includes the following types of conditions:

Keyword conditions: Keyword search is supported because of its benefits for querying data without a global schema or even an unknown strucuture (see also [3] for keyword search in an XML context). An example is: `A[keyword]=Tolstoy`.

Concept-value conditions: A concept-value condition has the form *concept=value* and would be the preferred type of searching if all web or intranet data were richly annotated with concepts corresponding to XML tags and values appearing in element or attribute contents. The comparison operator could be generalized to include type-specific comparisons (e.g., on dates). An example for this condition type is `A.title="War and Peace"`.

Similarity conditions: We have added a similarity operator ~ that can be applied to both concepts and values. This operator was first introduced in the XXL query language ([7]). It expands a term in the query with similar terms, supplied by an ontology service that the user can interact with. For example, adding the similarity operator to the concept *author* would not only return matches for the concept *author* but also for *writer* and other highly similar results. An example is `A.~author=Tolstoy`.

Path conditions: To express that different pages should be connected (within a small distance), we use two kinds of path conditions: Reachability through a direct link is expressed by a dot notation combining multiple variables, and reachability through a path of arbitrary length uses the wildcard symbol '#'. These conditions take into consideration hyperlinks, the parent-child relationship within XML documents, and also arbitrary XPointer or XLink references across document boundaries; so we consider connectivity in a global data graph.

For efficiency, COMPASS currently supports only conjunctions of search conditions. We believe that this is sufficient to cover almost all typical queries that users pose. Upon query execution, documents that satisfy one specific search conditions are bound to a variable; by using this variable as a prefix in other conditions one can reference this document in other search conditions. For example, once again consider the query about a book store in our home town that offers the book 'War and Peace' by Tolstoy. In our query language, this query could be expressed as follows:

```
SELECT A,B FROM INDEX
  WHERE A.~address=Saarbrucken
  AND   A[keyword]="book store"
  AND   B.title="War and Peace"
  AND   B.~author=Tolstoy
  AND   A.B
```

In this case, matches for the conditions *address = Saarbrucken* and *bookstore* are bound to the variable A, which can be used to express the path condition *A.B* that additionally requires all matches for A to directly link to a match B for *title="War an Peace"* and *~author=Tolstoy*.

4 The Architecture of COMPASS

4.1 System Overview

Figure 1 illustrates the main parts of the COMPASS prototype system. The crawler component collects the

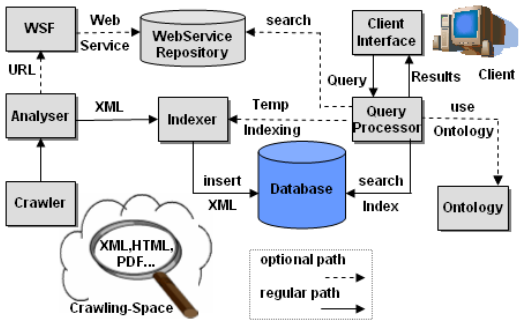

Figure 1: The architecture of COMPASS

data. It offers the functionalities of a Web Crawler combined with a local File Crawler. Depending on the data format, a transformation and semantic annotation into XML is performed by the Analyzer component in a heuristic manner. If applicable, the WSF component (see Section 4.3) creates a Web Service wrapper and saves the resulting WSDL description into a Web Service registry for later use during query processing. Eventually, all transformed data is inserted into the centralized index by the Indexer component.

We provide a graphical user interface that allows users to construct search requests in a visual manner without any knowledge of the COMPASS query language itself. For semantic expansion of similarity conditions, the user can interact with an ontology service. The query processor analyzes the query and can invoke further modules. In order to include Deep Web sources, the query processor interacts with the Web Service registry that attempts to find applicable Deep Web sources and accesses these using Web Services when executing the query. Finally, the results of the query evaluation are scored and returned to the user as a ranked list.

The system is completely implemented in Java running on a Tomcat application server. COMPASS uses Oracle 9.2i as its underlying relational database. The graphical user interface is implemented in the form of a JAVA applet using the JHotDraw library.

4.2 Data Indexer

COMPASS uses a centralized data index for efficent search evaluation. All data and also the relationships between documents are represented in a relational database. All data formats are transformed into XML by using heuristics as well as external annotation tools such as GATE ([6]). The documents' structure and tags are available for efficient evaluation of concept-value conditions, because all documents are inserted preserving their original link structure. This link structure is captured to evaluate path conditions. Moreover, to efficiently probe reachability by arbitrary paths (including paths across documents via XLink, XPointer, or href references), the transitive closure is materialized. Information about dynamically created Web Services is included in the index as well to select appropriate services when a user wishes to include Deep Web portals in her search.

4.3 Web Service Framework

When a crawl discovers a portal candidate (a web page that contains at least one HTML form), the Web Service Framework ([2]) is invoked which applies heuristic rules for generating a WSDL description on the fly. Typically, highlighted text next to form fields will become parameter names, and the type of a form field determines the corresponding parameter type (e.g., an enumeration type in the case of a pull-down menu). The generated WSDL descriptions and additionally generated Java classes are stored in a registry. The main index holds pointers to Web Services for invocation during query processing.

4.4 Query Processing and Ranked Retrieval

Internally, the query processor transforms each submitted query into an operator tree consisting of Java objects. A concept-value condition (see Section 3) is evaluated by first looking up the occurrences of both the concept name and the value in the index and then comparing the relative positions of each occurrence pair. In a good match, the concept is found 'above' the value with regard to the document structure; that is, the value should occur in a child node or a descendant of the node that matches the concept name. The distance between the two matches is reflected in the scoring of a result.

Simple keyword conditions are satisfied if the keyword occurs on a page. If the similarity operator ~ is used and, thus, this query condition is expanded with similiar terms, a page is matched if at least one of these terms is found. The score for this condition depends on the similarity of the matched term compared to the original query term as well as on the term frequency.

Path conditions (i.e., reachability through a single link or arbitrary path) refer to multiple pages bound to different variables. Whenever the query processor has determined candidate matches for local conditions and bound to variables, it tests the path condition using the materialized transitive closure, and discards candidates that are not connected or too far apart. The score of a path condition is based on the path length between the considered pages.

The order in which the conditions are evaluated is determined by a coarse selectivity estimation using simple statistics about term frequencies. Portals are included if the concepts of one or more concept-value conditions can be matched with parameter descriptors of generated Web Services. When portals are included into the search, the corresponding Web Services are invoked during query evaluation and their results are stored and indexed in a temporary table. The query

processor treats the page with the query form and a result page returned by the portal as a single logical unit. This way, a result page is bound to the corresponding variable even if some conditions are actually satisfied by the query submission page (e.g., names of concepts appearing as labels in the form and values appearing in the result page).

The overall ranking of a query match is computed based on the partial scores for all conditions, using a simple probabilistic model with independence assumptions.

5 Demo

We demonstrate the search on a large XML data collection, queries on a combined index containing XML and HTML web documents, and finally the additional integration of Deep Web portals.

5.1 Web Encyclopedia Data

We indexed a version of the Wikipedia project ([9]), a free web encyclopedia that is collaborately created by the internet community. The data collection consists of more than 215000 documents which do not follow a common schema. The documents are highly interconnected, with a total of more than 2 million links to other Wikipedia articles.

For example, consider a query about towns along the river Rhine that have hosted the Olympics. Figure 2 illustrates how this query can be posed using our graphical user interface. The system will search for information units that include the keyword *Olympics* and, at the same time, are connected to an information unit that matches the keyword condition *town* as well as the concept-value condition *river=rhine*.

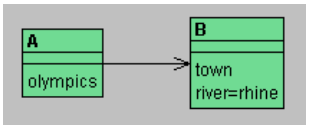

Figure 2: Query: Web Encyclopedia Data

5.2 Extended Bibliographic Data

The DBLP project ([5]) provides bibliographic information on major computer science journals and proceedings with almost 500000 articles. From this data we have created single XML files representing each occuring author, publication, and conference journal respectively; preserving their interconnections by adding more than 4000000 XLinks between the documents. Additionally, we crawled the authors' home pages that were also linked to by DBLP.

For example, consider a query about German professors that published an article in the journal of VLDB'02. This query is difficult because the necessary information is spread over multiple information units: while the authors of VLDB'02 are contained in the respective conference document, the origin of the author can only be found on the author's homepage on the web. COMPASS can exploit the fact that conference journals contain XLinks to the respective authors and all authors' documents contain XLinks to the respective authors' homepages. The resulting COMPASS query is shown in 3.

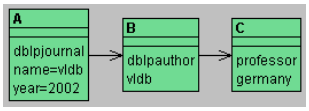

Figure 3: Query: Extended Bibliographic Data

5.3 Deep Web Data

We automatically generated Web Service wrappers for some popular portal sites including *www.amazon.com* and *www.imdb.com*. Thus, we are able to transparently access these portals when executing a query.

For example, consider a query about all conference articles and published books of Michael Stonebraker. This query combines the conference articles matched from Stonebraker's bibliographic data with the results about his other publications returned from Amazon.

References

[1] R. Baumgartner, S. Flesca, and G. Gottlob:. Visual web information extraction with lixto. *VLDB*, 2001.

[2] J. Graupmann and G. Weikum:. The role of web services in information search. *IEEE Data Engineering Bulletin 25(4)*, 2002.

[3] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: Ranked keyword search over xml documents. *ACM Sigmod Conference*, 2003.

[4] H. He, W. Meng, C. T. Yu, and Z. Wu. Wise-integrator: An automatic integrator of web search interfaces for e-commerce. *28th Conference on Very Large Data Bases (VLDB)*, 2003.

[5] M. Ley. Digital bibliography & library project. http://www.informatik.uni-trier.de/ ley/db/.

[6] U. of Sheffield. GATE – General Architecture for Text Engineering. http://gate.ac.uk/ie/.

[7] A. Theobald and G. Weikum. The index-based xxl search engine for querying xml data with relevance ranking. *EDBT*, 2002.

[8] V.Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Automatic data extraction from data-intensive web sites. *SIGMOD*, 2002.

[9] Wikipedia project. http://www.wikipedia.org.