IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4845

CS-Engine: Development of a Cross Search Engine for Multiple Heterogeneous Databases

Min Song and Il-Yeol Song[1]
College of Information Science and Technology
Drexel University
Philadelphia, PA 19104
(min.song, songiy)@Drexel.edu

Peter P. Chen[1]
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803
pchen@lsu.edu

ABSTRACT

Along with the growth of the Web, users have encountered the issue of information overload. As the Web becomes more prevalent, users are faced with a growing information burden. Given this situation, it is critical for a search engine to provide an "integrated solution" to users. With this in mind, we present the design and implementation of a cross-search component for CS-Engine (Cross-Search Engine). The CS-Engine allows users to search heterogeneous, multiple databases with one command. The CS-Engine is distinguished from other search engines in that it allows users to search both public domain web data and proprietary databases of a company. The CS-Engine is also different from meta-search engines because the CS-Engine does not need to trigger other search engines and translate a query for other search engines. Cross-search capability provided by the CS-Engine alleviates users' inconvenience of switching among different databases to satisfy their information needs. We conclude our paper with technical lessons learned as well as organizational issues encountered during the development phase.

1. INTRODUCTION

It is difficult to develop a large scale information system due to its size and complexity. Web-based Information Retrieval (WIR) system is one of the most difficult systems to develop [3, 11]. Because of the complexity of the system and the difficulty in indexing and searching heterogeneous data with different data structures and formats, a WIR system tends to focus on either publicly available web data or subscription-based proprietary data.

With growth of Internet, plenty of web search engines have become available. Existing search engines such as Google and AltaVista, however, don't provide users with the capability of searching multiple databases including proprietary databases. For instance, suppose a user wants to search both a patent database and web data in public domain through a single query transaction. Existing search engines are not the user's first choice to satisfy his/her information needs because the search engines currently don't provide such a capability.

Croft [2] states that one of the top requirements of a search engine is to provide an integrated solution to the user. Given demand on the integrated solution by the users, many search engines began to look at how to manage both many of the existing applications and various resources with different data structures and formats. One approach to an integration solution is to index and search multiple heterogeneous data, and this is where the current engines haven't explored rigorously.

In this paper, we present the system called Cross-Search Engine (CS-Engine) using Object Oriented technologies. The search engine is called Cross-Search because it can search multiple, heterogeneous databases with a single transaction.

We note that meta-search engines also provide cross-searching capability [6]. CS-Engine, however, is not a meta-search engine. First, CS-Engine has its own indexing component, which means there is no need to trigger other search engines. Second, it is not necessary to translate a query into another to use multiple search engines. The second difference allows the CS-Engine to avoid the problem meta-search engines encounter in constructing constraint rules imposed on translating queries by different search engines.

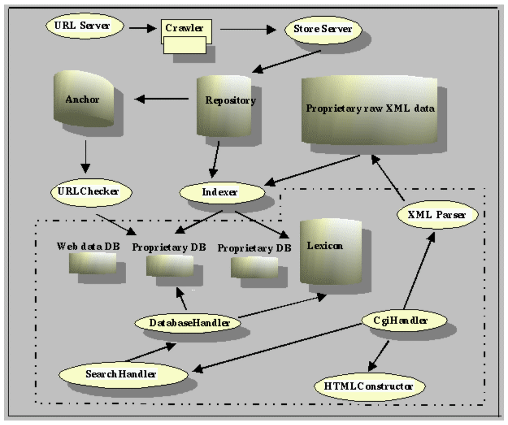
The rest of the paper is organized as follows: Section 2 describes the overall architecture of CS-Engine. Section 3 illustrates design of CS-Engine, whereas Section 4 describes the implemented system. Section 5 discusses lessons we learned during the system design and development. Section 6 concludes the paper.

2. SYSTEM ARCHITECTURE

A typical Web IR system consists of three components: 1) crawler, 2) index, and 3) search component [1]. Our CS-Engine adopts a generic architecture proposed by Brian and Page [1]. Figure 1 illustrates a simplified overview of CS-Engine architecture. Although this paper aims to focus on the search component, the other components are mentioned briefly to help the reader understand the architecture of CS-Engine. The following seven components constitute the CS-Engine architecture:

1) URLServer: It sends lists of URLs to be fetched to the crawlers. The fetched web pages are then sent to the StoreServer.
2) StoreServer: The StoreServer compresses and stores the fetched web pages into a repository. Every web page has an associated ID number called a RecordID that is assigned whenever a new URL is parsed out of a web page.
3) XML Parser: The XML Parser processes a set of proprietary XML formatted data before it is fed into the indexer. This component is developed based on a standard XML parsing library.
4) Indexer: The indexer performs a number of functions. It reads the repository, uncompresses the documents, and parses them. Each record consists of RecordSection and TermSection. Each record also contains a word frequency called WF. The WFs record a set of words, their positions in document, and term weights. The indexer stores these WFs into a set of "DBs". Each distinct domain data such as biomedical research XML data or web HTML data can be indexed in a separated DB, respectively (Web data DB and Proprietary DB). The Searcher can manage a dynamic list of DBs and search multiple DBs simultaneously.
5) URLChecker: The URLChecker reads the anchor file and converts relative URLs. URLs without domain name specified are converted into absolute URLs, meeting the requirements of the http- protocol. This module also serves as a placeholder for our editorial department to validate a set of chosen URLs.
6) Searcher: The Searcher looks up the Lexicon database and DBs produced by the indexer and retrieves a set of records. The searcher is run by a web server and uses the lexicon built by the indexer together with

Figure 1. The Overall System Architecture. The component surrounded by dashed lines represent CS-Engine



the inverted index and the DBs to answer queries.

7) HTMLConstructor: This component was developed with a standard XSLT processor. A set of templates for user interface of CS-Engine was developed in XSL. In the case of displaying the results on the summary page after the Searcher has retrieved the results in XML format, a result page in XSL template is transformed to HTML format by an XSLT processor.

The search component of CS-Engine is inside the dashed line of Figure 1. In this paper, we emphasize on development of the search component of CS-Engine.

3. SYSTEM DESIGN

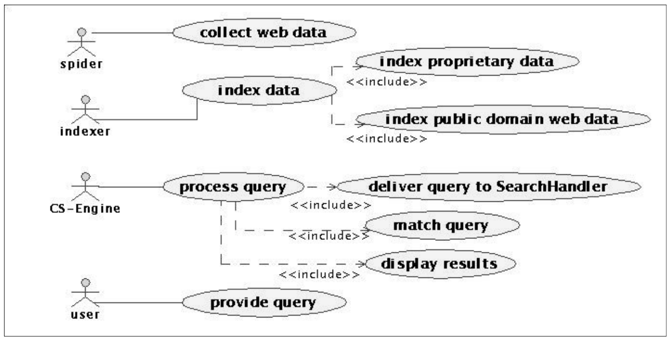
Throughout the development cycle of the system, UML was used to facilitate communication among developers and to embed object-orientation in the system. UML diagrams we developed include use case, class, and sequence diagrams.

3.1. Use Case Diagram

The important UML modeling that provides useful knowledge about the usage of a system is the use case diagram. Use case diagrams document the functionality of a system and users of the system [4]. The actors are shown as agents who interact the use case with system events. This use case diagram shows CS-Engine consisting largely of three components: 1) crawling, 2) indexing, and 3) searching component.

As illustrated in Figure 2, there are four main actors in the CS-Engine: 1)spider - crawling web data in public domain, 2)indexer - indexing public domain data as well as proprietary data, 3)search engine - delivering the query through CGI, looking for relevant documents, and

Figure 2. Use Case Diagram



displaying results, and 4)user - providing the query. In order to manage multiple, heterogeneous databases in an effective way, the following two *include use cases* were designed: 1) index proprietary data and 2) index public domain data. The *index proprietary data* use case involves parsing and indexing the company's data in an XML format. The *index public domain data* use case pertains to spidering and indexing the web sites in the public domain.

3.2 Class Diagram

In this section, we present the structure of CS-Engine at class diagram level and show how design patterns are used in the design of the CS-Engine. Class diagrams provide a static representation of the structure of a system. Class diagrams appear in varying levels of detail depending on the phase of the lifecycle. Figure 3 depicts a high-level conceptual class diagram of CS-Engine.

Design patterns were introduced during the design phase to provide solutions that can be applied in various circumstances that CS-Engine encounters.

To design a search component for CS-Engine, the following two design patterns were chosen: 1) Factory and 2) Strategy. The Factory design pattern was utilized to create a set of strategies to be processed in the web environment [7]. The classes of CgiStrategyFactory and HandlerFactory in Figure 3 were implemented with Factory pattern. The Strategy design pattern was adopted to process different business logics such as creating html template and initiating query session. The classes implemented using Strategy pattern are CgiStrategy and its subclasses, as well as EngineHandler‑and its subclasses.

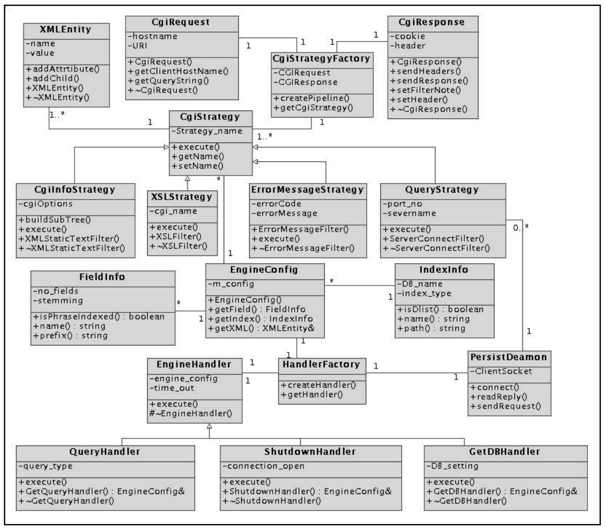
4. SYSTEM IMPLEMENTATION AND DESCRIPTION

In this section, we describe how the CS-Engine works in terms of the system components.

4.1. CGI Component

Given that a CGI program is executed once per request and can be executed many times per second on a heavily loaded server, minimizing start-up time for the CGI is an important consideration. For this reason, the list of filters, if possible, should be initialized statically at link time. This can be accomplished by creating an array of <name, value> pairs where the name is a filter name and the value is a function pointer that can be invoked to create a filter. Using this hash table, a CgiFilterFactory class can be defined that takes such a table as its sole constructor argument and searches the table whenever a new filter is requested. Note that the caller is expected to check for a null return value in case the specified name does not correspond to a known filter.

Figure 3. A class diagram for CS-Engine



4.2. QueryHandler and DatabaseHandler Components

The backend server can be a persistent server that performs queries against indexes. All requests and responses between the backend server and its clients are XML documents. The server is designed as a C++ class library, which accepts parsed XML requests and generates parsed XML responses.

The backend server is also implemented using the Strategy pattern. Individual strategy objects (called handlers) are invoked in response to each request [9]. The choice of handler for a particular request is based on the root node of the request in XML. The main functionality of the server involves executing queries against arbitrary collections of indexes and returning the results of those queries. Each query is specified using an XML request syntax. The following example illustrates some possible tags in a query:

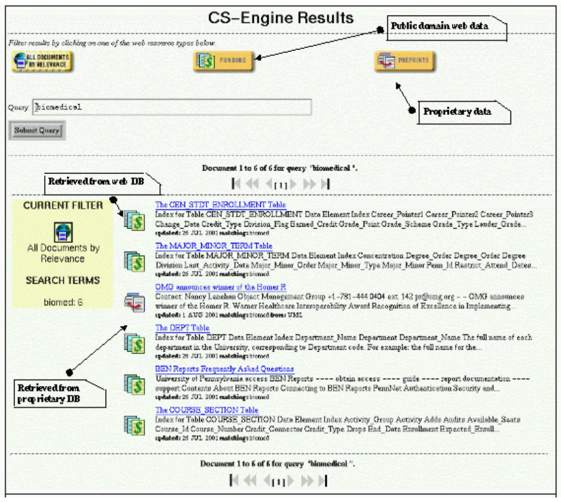
The scenario is that a user enters a query with a demand of getting an exact match as well as a partial match. With QueryHandler and DatabaseHandler Strategy patterns, multiple Boolean queries can be conducted with a probabilistic query in parallel against multiple target indexes. Also, the fields included in the result set, as well as the range of documents returned in the result set, can be specified as part of the query.

4.3. XSL Component for HTML page generation

CGI maintains a directory containing template XSLT template files. Every possible response page that CGI can generate has a corresponding template file. Some responses, such as requests to edit query terms or show a start page, will not require interaction with the backend server. Any page requiring query results needs interaction with the backend server to obtain search results. XSLT combines an XSL template file with an XML data set to produce an output file (in our case HTML page).

As illustrated in Figure 4, CS-Engine allows the user to edit either Boolean or probabilistic queries. If the query that the user types in is surrounded by a single quotation, the CS-Engine interprets it as a probabilistic query; otherwise, all queries are assumed to be Boolean queries. All the HTML pages are constructed dynamically by XSL templates based on XML format input. Once the query is executed, CS-Engine displays a set of summary pages containing information about each hit found by a query. Figure 5 also shows that CS-Engine enables the user to restrict search results to particular sets of fields specified at retrieval time to enhance performance. As illustrated in Figure 4, a graphic image displayed next to the retrieved records shows what database the retrieved document comes from. If the user clicks on individual hyperlinks, CS-Engine takes the user to the target web sites with matched terms highlighted and a return button to the summary page.

Figure 4. Search Page of CS-Engine



Document 1 to 6 of 6 for query 'biomedical '.

The CEN_STDT_ENROLLMENT Table
Index for Table CEN_STDT_ENROLLMENT Data Element Index Career_Pointer1 Career_Pointer2 Career_Pointer3 Change_Date Credit_Type Division_Flag Earned_Credit Grade_Point Grade_Scheme Grade_Type Lauder_Grade...
updated: 26 JUL 2001 matching: biomed

The MAJOR_MINOR_TERM Table
Index for Table MAJOR_MINOR_TERM Data Element Index Concentration Degree_Order Degree_Order Degree Division Last_Activity_Date Major_Minor_Order Major_Minor_Type Major_Minor Penn_Id Restrict_Attend_Dates...
updated: 26 JUL 2001 matching: biomed

OMG announces winner of the Homer R
Contact: Nancy Lanehan Object Management Group +1-781-444 0404 ext. 142 pr@omg.org - - OMG announces winner of the Homer R. Warner Healthcare Interoperability Award Recognition of Excellence in Implementing...
updated: 1 AUG 2001 matching: biomed from: UML

The DEPT Table
Index for Table DEPT Data Element Index Department_Name Department Department_Name The full name of each department in the University, corresponding to Department code. For example: the full name for the...
updated: 26 JUL 2001 matching: biomed

BEN Reports Frequently Asked Questions
University of Pennsylvania access BEN Reports ---- obtain access ---- guide ---- report documentation ---- support Contents About BEN Reports Connecting to BEN Reports PennNet Authentication Security and...
updated: 26 JUL 2001 matching: biomed

The COURSE_SECTION Table
Index for Table COURSE_SECTION Data Element Index Activity_Group Activity Adds Audits Available_Seats Course_Id Course_Number Credit_Connector Credit_Type Drops End_Date Enrollment Expected_Enroll...
updated: 26 JUL 2001 matching: biomed

Document 1 to 6 of 6 for query 'biomedical '.

4.4 Indexer Component

As described earlier in the paper, with the framework proposed by this paper, both internal and public available data are indexed by the indexer component of the system. The important difference between indexing proprietary data and public data is that for public data, we only index document surrogate types of data such as title of web page, keyword and author name. With regard to indexing proprietary data, input data is formatted in XML and the pre-defined set of XML elements from data are parsed and indexed. In searching time, when the user wants to view the full text of the public data, a link on the results page (Figure 4) takes the user to the site.

5. LESSON LEARNED

In this section, we summarize the lessons we learned in developing CS-Engine with object oriented technologies. Note that our emphasis of this section is on lessons we learned in designing and building of a specific search engine, rather than theoretic proof as an outcome of a research project with the prototype system for the validation of the concept.

Displaying duplicate records among the multiple databases: Since many instances of duplicate records exist across the databases, the duplicate records need to be identified either in search time or indexing time to properly display duplicate records. If we don't identify and filter out the duplicate records, it confuses the user when viewing the results of their search. In CS-Engine, we decided to filter out the duplicate records in search time by building a hash table for the duplicate records. With this approach, updating indexes is simplified and straightforward. The downside of this approach is that it slows down the search due to the fact that filtering out the duplicate records is done in real time rather than off-line.

Third party software dependency: We used a free third party XSLT library to transform XML to HTML. Some major issues on interface design were raised due to the bugs of the third party library. Since the communication channel with the third party company wasn't established in an efficient manner, it took a while to fix the problems. We felt that it is critical to establish a solid communication channel with the third party vendor early in the development phase.

HTTP Protocol: With regard to developing the web crawler package, we had difficulty in spidering some web sites, where their HTTP responses returned redirection and cookies messages for the HTTP POST method. In addition, since major web sites now place the robot.txt (a simple ASCII document that disallows search engines to spider the sites) in their web server, it was required to negotiate with the site owners to spider their sites. And since negotiation takes time, it is recommended to finish this business negotiation process prior to the development phase.

Handling special characters in XML entity: Since our XML-formatted web database contains data written in English as well as data in other languages, we had to cope with special characters such as ë or ä. The XML parser we used abruptly terminated its execution when it processed those special characters. To work around this problem, we took an ad hoc approach by replacing those characters in raw data with corresponding encoded characters. This is a well-known issue with XML parsers in handling some foreign characters in XML entities. For internationalization, handling of special characters needs to be addressed in the XML parser enhancement.

6. CONCLUSION

In this paper, we have presented the design and development of CS-Engine by using object oriented methodologies. The CS-Engine is differentiated from other search engines in that it allows the user to search heterogeneous, multiple databases within a single query transaction. The CS-Engine allows the user to search both public domain web data and proprietary databases of a company.

The CS-Engine developed in the spirit of OO enables the user to 1) query multiple database indexes selected at query time, 2) generate interface screens dynamically with XML and XSL, 3) edit and execute Boolean/probabilistic queries and 4) filter search results by database type. Cross-search capabilities provided by CS-Engine alleviate the users' inconvenience of switching databases and searching the Internet to satisfy their information needs.

We are currently conducting an intensive testing of how scalable and applicable CS-Engine is in the real-world scenario. In the experiments, the test query varies from one term to 50 term query. In addition, concurrency testing, varying from one user to 200 users, is also conducted.

The further follow-up study would be to evaluate the efficiency and effectiveness of the CS-Engine. In order to evaluate usability of the CS-Engine, the following criteria will be taken into account: 1) measure of the quality of the software, 2) speed of the system and 3) reusability of the components. This is a difficult task yet to accomplish.

In addition, we are currently re-architecting the indexing component of CS-Engine with UML and design patterns. We aim to compare the search component and index component in terms of different OO methods.

FOOTNOTE

1 The research of these authors is supported in part by AFOSR Grant No. F49620-01-1-0264.

7. REFERENCE

[1] Brin, Sergey and Page, Lawrence, The Anatomy of a Large-Scale Hypertextual Web Search Engine, *7th International World Wide Web Conference*, Brisbane, Australia 14-18 April 1998, pp. 234-251.

[2] Croft, Bruce W. What Do People Want from Information Retrieval? *D-Lib Magazine*, November 1995, pp. 754-766.

[3] Cutting D. R., J. O. Pederson, and P. Halvorson. An object-oriented architecture for text retrieval. In *Proceedings of RIAO'91*, 1991, pp. 440-449.

[4] Fowler, M. *UML Distilled*: *A Brief Guide to the Standard Object Modeling Language*. Adison-Wesley 1999.

[5] Hofmeister C, Nord RL, Soni D. Describing software architecture with UML, *1st Working IFIP Conference on Software Architecture (WICSA1)*, Feb 22-24, 1999, Software Architecture, 1999, pp. 145-159.

[6] Lieming Huang, Ulrich Thiel, Matthias Hemmje, Erich J. Neuhold. Distributed Information Search with Adaptive Meta-Search Engines. In Proceedings of *The 13th Conference on Advanced Information Systems Engineering (CAiSE'01)*, 2001, pp. 315-329.

[7] Gamma, E., Helm, R., Johnson, R. and Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995.

[8] Gibb F, McCartan C, O'Donnell R, et al. The integration of information retrieval techniques within a software reuse environment. *Journal of Information Science,* 26, 2000, pp. 211-226.

[9] Li JF, Chen J, Chen P, Modeling web application architecture with UML. *36th International Conference On Technology of Object-Oriented Languages And Systems*, Proceedings, 2000, pp. 32-39.

[10] Spink A, Ozmutlu HC What do people ask for on the Web and how do they ask it: Ask Jeeves query analysis, *Proceedings of American Society for Information Science and Technology*, 38, 2001, pp. 545-554.

[11] Wade S and Braekevelt, P. IR Framework - An Object-Oriented Framework For Developing Information-Retrieval Systems. *Program-Automated Libraries*, 29, 1995, pp. 15-29.