ENTERPRISE INFORMATION SEARCH SYSTEMS FOR HETEROGENEOUS CONTENT REPOSITORIES

Trieu C. Chieu, Shyh-Kwei Chen and Shiwa S. Fu
IBM T. J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA

Keywords: Integration, Federation, Search, Service-Oriented Architecture, Database.

Abstract: In larger enterprises, business documents are typically stored in disparate, autonomous content repositories with various formats. Efficient search and retrieval mechanisms are needed to deal with the heterogeneousness and complexity of this environment. This paper presents a general architecture and two industrial implementations of a service-based information system to perform search in Lotus Notes databases and data sources with Web service interfaces. The first implementation is based on a federated database system that maps the various schemas of the sources into a common interface and aggregates information from their native locations. This implementation offers the advantages of scalability and accessibility to real-time information. The second one is based on a one-index enterprise-scale search engine that crawls, parses and indexes the document contents from the sources. This latter implementation offers the ability of scoring the relevance ranking of documents and eliminating duplications in search results. The relative merits and limitations of both implementations will be presented.

1 INTRODUCTION

Today's business environment is getting more information-driven and data-centric. With the explosive growth of information, enterprises are looking for architectural solutions to information access problems. The projected size of information generated in the next few years will be more than all of the recorded history (Lyman, 2003), and the information will be stored in different autonomous content repositories. The result is a decentralized environment with different specialized content management systems made of various techniques and data models, and operated by different organizations. In order to find and access a stored document, each client program must understand the different semantics of the local schemata. To address this problem, enterprises often use data warehousing solutions, wherein data from multiple sources is collected in a centralized database. While this approach simplifies the access and analysis of the data stored in different repositories, it increases the storage-related hardware and software costs, and may require re-implementing specialized searches in the data warehouses for data retrieval. Inconsistency can arise due to a lag in synchronization between sources and data warehouse.

One alternative to data warehousing is data federation (Haas, 2002), wherein the data can continue to be stored in its native locations and retrieved via a middleware component. A number of federated search engines (Dogpile site, n.d., Metacrawler site, n.d., Myriad Search site, n.d.) have come to market to meet the demand. Federated search is a technology that allows simultaneous search from different databases. It has been promoted as a method of providing "one stop shopping" for searchers. A user enters search terms using one search interface. The search is then executed in all the associated databases, and the results are returned in a single result list. This technology is now being used for journal citation databases, electronic journals, and other library resources. However, keeping the data in its original location and treating the data sources as "black boxes" in the data federation architecture imposes some challenging problems. There are issues such as what is an effective way of communicating with the heterogeneous data sources, what is an efficient way to perform query generation and result integration, and how can one easily eliminate duplicate information from search results.

C. Chieu T., Chen S. and S. Fu S. (2007).
ENTERPRISE INFORMATION SEARCH SYSTEMS FOR HETEROGENEOUS CONTENT REPOSITORIES.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - DISI*, pages 365-371
DOI: 10.5220/0002349603650371

Recently, Service-Oriented Architecture (SOA) has materialized as a preferred solution for integrating distributed applications (Mahmoud, 2005, Weerawaran, 2005, Web Services Architecture, n.d.). In SOA, an application that needs to be accessed by another application can be done by exposing a service interface. A service consumer can discover a service from a service directory and invoke the service remotely. SOA has great promises to make the integration of heterogeneous content repositories much simpler.

In this paper, we present a general architecture and two different implementations of a service-based information search system that integrates data from various, heterogeneous data sources. The first implementation is based on a federated database management system, while the second one is based on a one-index enterprise-scale search engine. To illustrate the complexity and heterogeneity of the environment, we choose two different data sources. The first one is a powerful class of document centric, collaborative groupware system known as Lotus Notes/Domino document database (IBM Lotus Notes/Domino site, n.d.). The second one is a content repository server exposing Web services as search interfaces. We will depict in details the approaches of how to connect, map, search and integrate information from these sources. Finally, the relative merits and limitations of both implementations will be presented and discussed.

2 ARCHITECTURE OF INFORMATION SEARCH SYSTEM

Figure 1 illustrates the conceptual architecture of an information search system that connects to a number of disparate, heterogeneous content repositories. The system is designed based on a service-oriented architecture with service components that interact with external systems through service calls. The front end of the system is a query generator. The back end is an integration layer that consists of a model transformation engine and a number of service adaptors used to integrate the various, heterogeneous data sources. The system also accepts client search request through a service interface. When a search request is received, the query generator customizes the user query into valid queries for the data sources. The transformation algorithm of the query generator relies on the semantic information about the data sources (Chen, 2005). Based on the transformation result, the query generator creates an executable query for all the related data sources and triggers the integration layer with the query.

To serve the integration needs, the integration layer is architected with a pluggable framework to allow multiple adaptors to be plugged in. Adaptors (Fu, 2005) are software modules made up of data source driver and software routines to interact with data sources. Adaptor may have the capability to transform and map the data representation of a remote data source into a common format to facilitate integration. Different adaptors are used for different types of data sources such as file-system adaptor, relational database adaptor, XML Web Services adaptor, etc.

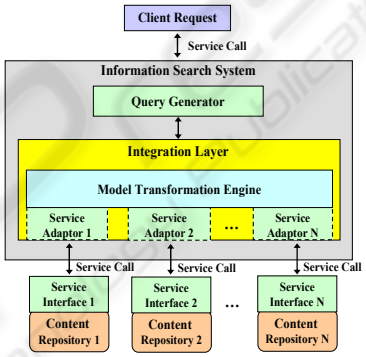

Figure 1: Conceptual architecture of an Information Search System.

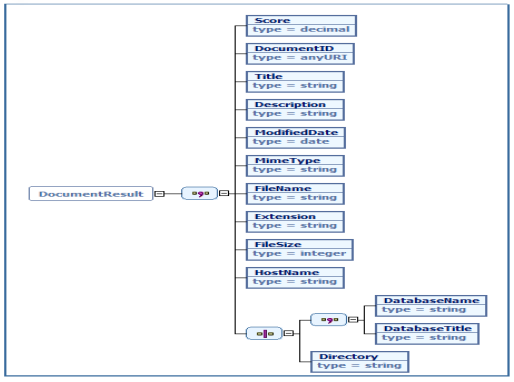

Figure 2: Schema of the common data model used to represent a search result.

The search results from various data sources are combined into a common data structure by the model transformation engine before returning to the search client. Translating and mapping various schemas into a common data representation is essential in the service based architecture (Madhavan, 2003). This approach basically abstracts

the heterogeneity of the various source schemas into a common one, and thus reduces the complexity of the system implementation by separating it into data-source independent and dependent implementations. The "*DocumentResult*" schema of the data model used for a document search result is given in Figure 2. The data structure contains a number of elements to specify the common meta-data information about the resulting document, These elements include the relevance ranking (*Score*) of the document within the search results, its original URI location (*DocumentID)*, server name (*HostName)*, title (*Title),* description (*Description*), file update date (*ModifiedDate*), file name (*FileName*), file type (*MimeType*) and other characteristics of the document file.

3 IMPLEMENTATION OF INFORMATION SEARCH SYSTEM

In this study, two different implementations of the information search systems are presented. The first implementation is based on a federated database management system (DBMS). It uses an IBM DB2 Information Integrator (DB2 II) as the back end integration layer for hosting information from various content sources and generating optimized queries for the sources. The heterogeneous data sources include a file system, a Lotus Notes database (IBM Lotus Notes/Domino site) and an IBM DB2 Content Manager server (IBM DB2 Content Manager site, n.d.). The second implementation is based on one-index enterprise-scale search engine to crawl, parse and index the aggregated contents of the information sources. Basically, this latter one uses IBM WebSphere Information Integrator, OmniFind Edition (IBM WebSphere Information Integrator, n.d.), which provides different crawlers to crawl and aggregate the information of various content sources. The details of these two implementations are given in the following Sections.

3.1 Federated Search Server with Federated Database Management System

Federated database management system (DBMS) provides a flexible and effective means for transparent access to heterogeneous data sources (Hass and Lin, 2002, Tork Roth, 2001). It offers a unified interface for accessing data, thus relieving component developers of much of the burden associated with connecting, combining, filtering and transforming data from the sources.

Figure 3 shows an implementation of a federated search server with a federated DBMS, namely the IBM DB2 Information Integrator (DB2 II). This server exposes its search interfaces through Web services. It consists of a front-end SQL generator as the query generator, and a back-end DB2 II system as integration layer. As data sources, three content repositories consisting of a flat file system, a Lotus Notes database, and an IBM DB2 Content Manager server are used. To communicate with the data sources, three types of wrappers from DB2 II are used. Wrappers are adaptors by which DB2 II interacts with data sources. DB2 II uses routines stored in a library module to implement a wrapper. Each wrapper supports various operations, depending on the capabilities of the data sources that the wrapper is supposed to access. DB2 II provides a set of default wrappers for some common sources. There are the DB2 wrapper, ODBC wrapper, table-structured files wrapper, XML wrapper, and Web Services wrapper. Wrapper is also responsible for mapping the data representation of remote query results into a table format as required by the federated database.

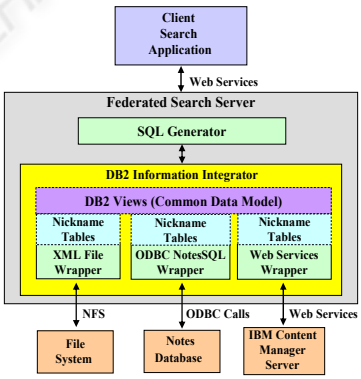

Figure 3: Implementation of a federated search server based on IBM DB2 Information Integrator and wrapper technologies.

As mentioned earlier, Lotus Notes database is a document-based database management system. It has become a mainstay at many enterprises for applications such as workgroup collaborations and complex sales tracking and monitoring. Because Notes organizes data into documents, Notes internal database structure is not relational in design and is far from tabular. Particularly, Notes allows users to create and update documents using forms, and uses views to define and display lists of documents. Its

compound document data structure is designed for maximum flexibility to support the rich object collections in Notes text documents. Since Notes users still have traditional analysis and reporting requirements and need access to Notes information from a variety of tools that rely on SQL data access, a Lotus NotesSQL ODBC driver (Lotus NotesSQL site, n.d.) has been made available for this purpose. Thus, to connect to the Lotus Notes database in our implementation, a NotesSQL driver incorporated with a DB2 II ODBC wrapper is used. Once connected, standard DB2 SQL statements can be used to query the ODBC connection to the Notes database. The wrapper then maps all the Folders, Views and Forms within the Notes database to simple relational database tables as nickname tables.

An example illustrating the mapping of the Notes database views into DB2 II nickname tables using the ODBC NotesSQL wrapper is shown in Figure 4. Basically, all the Notes folders, views and forms in the Notes database are mapped to DB2 II nickname tables, and the Notes form fields and view columns describing the Notes documents are mapped to the columns in a nickname table. Since Notes is more flexible about names than SQL, it allows many special characters and sequences of characters that are not part of the standard SQL syntax when naming a form or view. Thus, certain characters in Notes names such as periods, spaces, backslash and forward slashes are mapped to the "underscore" character in SQL names.

For the DB2 Content Manager repository that exposes its service interfaces as Web services, DB2 II maps the service calls into nickname tables using a Web service wrapper. Both input parameters and output parameters of the Web service calls are represented as columns within the corresponding nickname tables. The tables and columns are discovered and created by a DB2 II utility using the WSDL and XSD files as inputs. The application that accesses the Web services can then use a regular SQL statement for that nickname. The SQL query includes parameters for the columns representing the input parameters. When a Web service call returns complex and nested results, DB2 II uses multiple nickname tables to hold data that are related via primary and foreign key relationships.

Upon receiving a search request, the SQL generator first composes and parses the user query into valid DB2 II queries to be forwarded to the integration layer. The wrappers in the integration layer are responsible in executing the query on individual data sources and retrieve the remote query results back into table format. The final search results are then combined and transformed into a common data representation to be returned to the search client. To simplify the implementation of the model transformation, a DB2 view based on the common schema of "*DocumentResult*" as given in Section 2 is created to hold the intermediate results. The DB2 view is generated using SQL statements by selecting and joining the equivalent column names in the nickname tables using concept mapping and instance mapping (Chen, 2005) based on the semantic relationships about the data source models. Briefly, concept mapping is a renaming method that translates the different names of the same concept in different data sources, and instance mapping is an interpretation method that correlates the equivalent instances from different data models. In the current implementation, the model transformation and mappings are carried out manually by analyzing the semantic concepts in the original data source models.

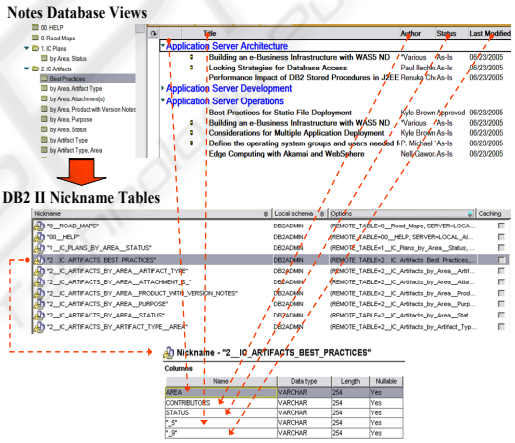

Figure 4: Example illustrating the mapping of Notes database views into DB2 II nickname tables.

3.2 Integrated Search Server with Enterprise Search Engine

Figure 5 illustrates the implementation of an integrated search server with an enterprise search engine. Similar to the federated search server, this server also exposes its search interfaces through Web services. It consists of a front end query generator and a back-end enterprise search engine, namely the IBM WebSphere Information Integrator OmniFind Edition. The query generator interacts with OmniFind through a Java search API. In this implementation, the model transformation and mapping functions of the integration layer are incorporated in the Omnifind search engine. As data

sources, a similar set of content repositories consisting of a flat file system, a Lotus Notes database, and an IBM DB2 Content Manager server are used. To connect to these data sources, three different crawlers, namely a file system crawler, a Notes crawler and a Content Manager crawler provided by the OmniFind engine are used. The crawling, parsing and indexing tasks for these content repositories are performed based on a sequence of scheduled processes in an internal scheduler. The indexing results are combined in a single collection of searchable index made available for search.

In general, the OmniFind search engine is an enterprise-scale search engine to allow searching of items from databases, e-mail archives, Web sites and more. The engine consists of a crawler, a parser-indexer, and a run time that provides client interface services. The crawler is used to spider through online assets within the enterprise. Results are parsed into individual words and links, which are then assembled into an index. This index supports search queries. Particularly, the Notes crawler can find and parse attached documents in Notes forms of Notes database. Data parsed and extracted from the crawler is fed into the indexing engine. After parsing, categorizing, and weighting the data from the crawler, the engine generates an index that becomes the searchable information to answer search queries.

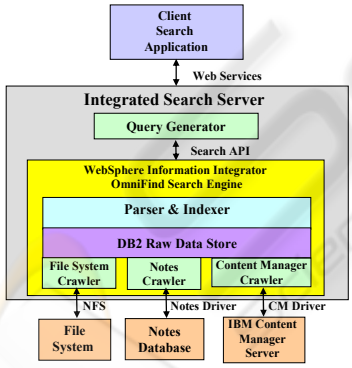

Figure 5: Implementation of an integrated search server based on IBM WebSphere Information Integrator OmniFind search engine.

Upon receiving a search request, the query generator first composes and parses the user query into search queries in XML fragment syntax (Carmel, 2003) to be accepted by OmniFind. Using the OmniFind search API, document search is then carried out by the search engine on the combined search index representing the multiple content repositories. The final search results are organized into a common data representation, and returned to the search client. An example of search results with the search keywords of "websphere DB2 server" is shown in Figure 6. The results illustrate a mixture of search items from all three data sources. Each item in the result list consists of a collection of meta-data based on the common schema "*DocumentResult*" that specifies the corresponding *DocumentID*, relative ranking *Score*, *Title*, *Description*, *ModifiedDate*, and other characteristics of the documents. The items are displayed in an order according to their ranking score that indicates the relative keyword match rate among all documents. The *DocumentID* is an URI that links to the native location of the document, and is used to retrieve the original content from its content repository.

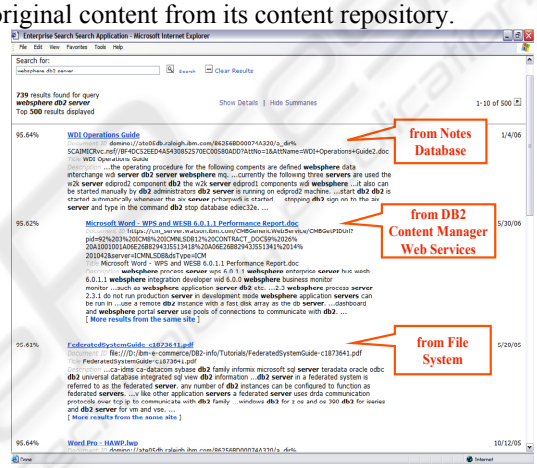

Figure 6: Example of search results from the integrated search server.

4 DISCUSSIONS

Federated search, also known as meta-search, has been promoted as a method of providing "one stop shopping" for searchers. Users exercise a single search interface and find items that they may overlook otherwise. There are a number of Internet federated search sites (Dogpile site, n.d., Metacrawler site, n.d., Myriad Search site, n.d.) that provide the ability to search across multiple search engines. However, the results obtained from these federated engines are not as good as the "one-index" integrated search engines (Google site, n.d., Yahoo! Site, n.d.), because they do not actually crawl and index the documents, hence without enough knowledge to perform analysis to refine the search results. Particularly, they lack the capability to remove duplicate documents from the search results from multiple sources. In contrast, integrated search

engines usually include intelligent algorithms to avoid storing multiple copies of the same documents in the index. Ranking of the relevance of documents is also an issue in federated search. Federated search engines can not perform relevance ranking of documents effectively. They can only predict which document is a better match by examining the titles, snippets, and URIs from the results. In contrast, integrated search engines can rank documents based on contents in the documents, number of links to the documents, and other factors.

Our federated search system also suffers the same limitation as mentioned above for federated search. Particularly, the NotesSQL driver used to connect to the Notes database lacks the capability of discovering document attachments in Notes forms, thus is incapable of providing search results on attached documents.

Although federated search system is subject to lower-quality than integrated search, it possesses other advantages. It is highly scalable because it links directly to data sources and can compose the search results in real-time. Thus, the information obtained is more up to date than the integrated search system. In contrast, the integrated search engine has to produce a local index containing all documents that it finds, and may eventually run out of capacity due to the rapidly growing number of documents to index. Moreover, there is always a time lag between when a document is changed and when the one-index search engine updates its index.

Resource requirement may be another advantage of the federated search server. Federated search system leverages distributed engines, thus may not require much resources to support the runtime performance. On the contrary, integrated search system needs to frequently produce and store a local index, thus may require excessive resources to realize acceptable performance.

5 CONCLUDING REMARKS

In this paper, we present a general architecture and compare two implementations of a service-based information search system to search on multiple content repositories. To illustrate the heterogeneity of the complex environment, we select as data sources a combination of a Notes database, an IBM DB2 Content Manager server that uses Web services for search interfaces, and a Windows file system. First implementation is a federated search system that integrates and maps the various schemas of the sources into a common interface. This system leverages data search capabilities in native source locations, thus offering the advantages of scalability and accessibility to real-time information. It is simple to implement, and does not require much resources to support the runtime. However, it suffers the limitations of incapable to perform relative ranking of search results, ineffective in eliminating duplications, and unable to find document attachments in Notes forms from Notes databases. Second implementation is an integrated search system. This system uses crawlers and indexers to collect and analyze information from different sources into a single index. It offers the ability to perform relative ranking of documents, and eliminates duplications in search results. However, it undergoes the limitation of scalability, and may require excessive resources for acceptable performance in frequently refreshing the forever growing index of information.

REFERENCES

Lyman et. al., 2003. How Much Information 2003?. From http://www.sims.berkeley.edu/research/projects/how-much-info-2003/

Haas et. al., 2002. Data Integration through Database Federation. *IBM Systems Journal*, *Vol. 41, No. 4*, 578-596.

Dogpile site. From http://www.dogpile.com/.

Metacrawler site. From http://www.metacrawler.com/.

Myriad Search site. From http://www.myriadsearch.com/.

IBM Lotus Notes/Domino site. From http://www-142.ibm.com/software/sw-lotus/products/product4.nsf/wdocs/noteshomepage.

Mahmoud, Q. H., 2005. Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI). Sun Developer Network Web site: http://java.sun.com/developer/technicalArticles/WebServices/soa/.

Weerawarana et. al., 2005. Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice Hall.

Web Services Architecture. From http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

Chen et. al., 2005. Semantic Query Transformation for Integrating Web Information Sources. In Proc. 7th Int'l Conference on Enterprise Information Systems, 176-181.

Fu et. al., 2005. An Intelligent Event Adaptation Mechanism for Business Performance Monitoring. In ICEBE 2005, 2005 IEEE Int'l Conference on e-Business Engineering, 558-563.

Madhavan, J., and Halevy, A., 2003. Composing Mappings among Data Sources. In VLDB, 572-583.
IBM DB2 Content Manager site. From http://www-306.ibm.com/software/data/cm/cmgr/.
Hass, L., and Lin, E., 2002. IBM Federated Database Technology. IBM DeveloperWorks Web site: http://www-128.ibm.com/developerworks/db2/library/techarticle/0203haas/0203haas.html.
Tork Roth et. al., 2001. An Architecture for Transparent Access to Diverse Data Sources. Component Database Systems, 175-206.
Lotus NotesSQL site. From http://www-12.lotus.com/ldd/doc/notessql/3.0.1/notes_sql.nsf/66208c256b4136a2852563c000646f8c?OpenView.
IBM WebSphere Information Integrator. From http://www-306.ibm.com/software/data/integration/db2ii/editions_womnifind.html.
Carmel et. al., 2003. Searching XML Documents via XML Fragments. In Proc. 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.
Google site. From http://www.google.com/.
Yahoo! site. From http://www.yahoo.com/.