Bulletin of the Technical Committee on

Data Engineering

June 1998 Vol. 21 No. 2

IEEE Computer Society

Letters

Special Issue on Databases and the World Wide Web

Announcements and Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering (http: www. is open to all current members of the IEEE Computer Society who are interested in database systems.

The web page for the Data Engineering Bulletin is http://www.research.microsoft.com/research/db/debull. The web page for the TC on Data Engineering is http://www.ccs.neu.edu/groups/IEEE/tcde/index.html.

Letter from the Chair of the TC on Data Engineering

To all members of the TCDE:

The executive committee of the TC voted in their last meeting to convert to a mostly electronic version of the Data Engineering Bulletin starting in January, 1999. This means that we will no longer be automatically mailing out paper copies of the Bulletin to all members of the TC. This will cut down on the expenses of the TC without, we hope, causing undue inconvenience to our members, most of whom will get their copies of the Bulletin through the Bulletin web page.

The Bulletin will continue to be available on line as usual at

http://www.research.microsoft.com/research/db/debull.

Due to the hard work of Dr. David Lomet, the issues are presented in European and American postscript formats, with and without illustrations. Recently, individual articles have also been made printable from the online version of the Bulletin. However, we realize that not all TCDE members may be able to view and print the Bulletin from the Web. For this reason, we expect to send a very few paper copies to those of you who request it.

If you would like a paper copy of the Bulletin, please write to Ms. Tracy Woods at the IEEE CS

Tracy A. Woods
IEEE Computer Society
1730 Massachusetts Avenue NW
Washington D.C. 20036-1992 USA
twoods@computer.org

Include in your message a postal address where your paper copy of the Bulletin is to be sent. If you do not send Ms. Woods such a message, we assume that you are able to obtain your copy of the Bulletin electronically, from the web page.

Betty Salzberg
Northeastern University

Letter from the Editor-in-Chief

Our New Financial Plan

Those of you who have read prior letters from me about the Bulletin's financial situation will recall that the TC on Data Engineering does not generate enough revenue for the standard IEEE mechanisms to adequately fund the Bulletin. This has been a long-standing concern of mine. Our new TC Chair, Betty Salzberg, explains in her letter a major step to put our finances on a firmer footing. We will distribute hardcopy of the Bulletin starting in 1999 *only* to those TC members who explicitly request hardcopy. All members will continue to receive email announcing each new Bulletin issue, and will be able to download the issue from our web site

http://www.research.microsoft.com/research/db/debull.

We are including this information in the Bulletin itself, as our current hardcopy distribution is the only sure way we have to reach all our members. I will be reminding you of this new "hardcopy only by request" policy in subsequent email announcements. But the ones who really must pay attention to this are exactly those that we have trouble reaching by email, i.e. those who cannot access the Bulletin web site. If you are one of these and want to continue receiving the Bulletin, you *must* act now. Please read Betty Salzberg's letter on page 1 for specifics.

Changing Editorial Staff

With this issue, the Bulletin has a completely new set of editors. Daniel Barbara's term as Bulletin editor concluded with the March 1998 issue on "Mining of Large Datasets". Daniel also was the editor for the March 1997 issue on "Supporting On-line Analytical Processing". Both of these issues were on subjects that are of current consuming interest within our database community. Daniel did a fine job on both of the issues, bringing together articles from leading researchers in industry and universities. I want to thank Daniel for these excellent issues and his hard work in putting them together.

I am very pleased to announce that Elke Rudensteiner from Worcester Polytechnic Institute (WPI) in Worcester, Massachusetts has agreed to serve as a Bulletin editor. Elke previously was a faculty member at the University of Michigan. Her Ph.D. is from the University of California at Irvine. Elke has been active in the database community for ten years, with interests in OO and multi-media databases, data warehousing, and web and distributed database applications. She is a past NSF Young Investigator. It is a real pleasure for me to welcome Elke to the Bulletin, and I look forward to working with her on an issue coming your way soon.

This Issue

David DeWitt, in an invited talk at a recent SIGMOD Conference, suggested that database research might be "road kill on the Information Highway". I hope that by the time you have finished reading the current issue, that you will agree that, if our community is "road kill", that it has refused to stay dead for long. Surajit Chaudhuri has assembled papers from researchers, both industrial and academic, and developers, presenting a overview of much that is going on in the web and database space. This issue explores how database and the web are "assimilating" each other, and explores a collection of applications of database technology to information management on the web. I think you will end up agreeing with me that our research community is a very lively corpse indeed in dealing with the web. I want to thank Surajit on bringing together this broad collection of intriguing papers.

David Lomet
Microsoft Research

Letter from the Special Issue Editor

The World Wide Web is now part of our everyday life. The enormous success of the web has also generated significant challenges of ensuring that the web transitions to a scalable and a managable infrastructure. Several new standards and technologies have emerged that specifically address these concerns. Two such key developments are XML and Dynamic HTML(DHMTL). These new standards and technologies will reshape the web in many fundamental ways. Specifically, they also impact the way data is stored, exchanged and served over the internet. Although in the early years of web our community has had little impact on the evolution of the web, some of the activities in the recent past have been quite encouraging and have the potential for impact. In this special issue, I have put together a sampling of the ongoing research and entrepreneurial work in the context of the web information systems.

In the first paper, Susan Malaika explains how data from relational database is served over the web today. Her article is an overview of the various ways in which data from relational tables finds their way to a web client. She highlights some of the key system integration issues that discusses some of the tricky issues in supporting transactions and the effect of caching. Her article is written in true internet with references only to URLs. For those of you not familiar with XML, her article serves as a preliminary introduction.

The second paper by Fernandez et.al. describes a system (Strudel) that is geared to the problem of designing and managing a web site by exploiting database techniques. Specifically, the Strudel system uses views to represent data that resides in external sources and is able to present one or more integrated view of the information in a web site. This work illustrates application of some of the past work on querying and representation of semistructured data in the context of the web.

The third and the fourth papers address the very real problem of searching the net and making sense of the results. The paper by Manber and Bigot argues for a universal serach interface to support the ability of the user or applications to customize their serach over the net by connecting to multiple search engines and sites in a flexible way. They develop a model around "search objects" and "search scenarios" where the latter is obtained by composing objects in a variety of ways. While the paper by Manber and Bigot focuses on connectivity to sources, the paper by Gravano and Papakonstantinou, concentrates on the problem of making sense of questions as well as information rerieved from multiple internet sources. They stress the similarity of goals and some of the techniques used for metasearchers that brokers among multiple search engines and relatively old-fashioned problem of information integration across multiple databases. I think exploiting the synergy of these two problems is of importance.

The paper by Brin et.al. talks about two important problems. The paper introduces a novel way to measure the quality of a web page in an answer set and shows its use in the Google search engine (http://google.stanford.edu) for enhanced quality of ranking. I find their design of the ranking algorithm very insightful. The paper also discusses the problem of extraction of relations (structured information) from relatively unstructured web and serves as the introduction to our final paper by Prasad and Rajaraman. This paper provides an overview of the Virtual Database Technology developed by Junglee Corporation to integrate multiple semi-structured internet sources into a single structured database. The authors also comment on the effect of XML in exchanging information about capabilities about sources to enable more efficient integration.

The papers in this issue reflect work in a diverse set of issues - web site management, connectivity, integration and ranking of information from multiple sites, extraction of structured information from the web. These are important problems for the web infrastructure and the database technology has much to offer. Hopefully these articles will provoke your interest in examining how to best leverage database technology in the context of the web.

Surajit Chaudhuri
Microsoft Research

Resistance is Futile: The Web Will Assimilate Your Database

Susan Malaika
IBM Santa Teresa Laboratory
Email: malaika@us.ibm.com

Abstract

Developers of relational database systems have made few concessions to the Web, preferring that Web applications access relational tables through SQL (or other programming interfaces) via intermediate gateway software. Result sets from database queries continue to be delivered to applications for further processing one row at a time. This article describes ways that relational databases are accessed on the Web through HTML and XML, and some system integration issues.

1 Introduction

Many of the relational database activities for the Web are in the following categories:

- *Store and manage Web resources in relational databases:* Storing Web related information, such as hypertext documents and hypertext links, in relational databases, and then making the information accessible through the Web.
- *Query and access Web resources using relational database techniques:* Making Web resources resemble structured data, often by augmenting them with metadata or by building indexes, and then querying, searching or mining the data using database query languages which can operate in conjunction with query optimizers.
- *Store and manage relational data on the Web:* Transforming tabular data in relational databases into formats immediately accessible on the Web and managing the transformed relational data in caches and data stores around the network.
- *Query and access relational data using Web techniques:* Accessing tabular data in relational databases from Web clients via server gateways or through software running in the clients, which is the main topic of this article.

2 The Web and Relational Databases

2.1 The Web

The Web provides easy human access to a variety of resources held on private and public networks. It is made up of client and server systems that communicate through agreed protocols. Servers deliver information in response

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

to client requests, usually without either server administrators or client users being aware of intermediate routing and caching systems that exist between clients and servers. Information delivered from servers can be created in advance by humans or programs *(static content)*, or at the time clients request information *(dynamic content)* and often dynamic and static content are combined. The next three sections describe major elements of the Web in the context of databases.

2.1.1 Universal Resource Locators (URLs)

URLs support a global naming scheme, that humans or programs can employ, for accessing resources such as files held at servers. A URL [7] encompasses the name of a resource as well as the protocol used to access a resource in an extensible way such as:

- *ftp://server:port/resourcename* where FTP is the protocol name used to retrieve a file.
- *jdbc:dbsystem://server:port/resourcename* where *dbsystem* is the name of a database system (such as Informix, Sybase). JDBC (Java Database Connectivity [12]) is a programming interface for Java applications to access relational databases. Because there is no agreed communication protocol from clients to relational database systems, the protocol name jdbc:dbsystem is not supported in general purpose products. The name of the client communication software (the driver name) and the database access request are defined within the client Java application, in addition to the JDBC URL.
- *http://server:port/virtualdirectory/owa/packagename.storedprocedurename* where the latter part of the URL is a database stored procedure request over the HTTP protocol, destined to an Oracle database through an Oracle Web server [6].
- *http://server:port/db2www/?langenv=dtw_sql(perform)&sql="select*$+*+from+$*opera"* where the latter part of the URL (known as a *database query URL*) is an SQL request over the HTTP protocol, destined to a DB2 [3] database through a Web server and Net.Data [19]. The result set is delivered in an HTML page.

2.1.2 Hypertext Markup Language (HTML)

HTML is a document markup language whose definition is expressed in an SGML DTD (Standard Generalized Markup Language Document Type Definition [8] [10]). Clients request HTML documents by specifying their URLs, and servers deliver documents to clients. In addition to markup tags for document structure (such as $<p>$ for a paragraph) and for presentation (such as $<hr>$ for a horizontal rule), HTML contains useful features such as:

- *Forms:* providing a means for humans and programs to supply input parameters to servers often to generate and tailor dynamic content. Input parameters are frequently delivered to server applications as name and value pairs separated by ampersands as in the database query URL example above where the variables are *langenv* and *sql*.
- *Hypertext Links* making it possible for humans and programs to navigate easily within and between static and dynamic content delivered from different servers. These unidirectional links refer to URLs and require no modifications in target resources. Link integrity is not usually monitored, other than through occasional use of link checker software.

Authors can incorporate product specific database constructs in HTML documents, that do not appear in the standard HTML DTD. Database constructs for HTML are usually transformed into regular HTML at servers just before delivery to clients. Here are some examples:

- *HTML Database Markup Tags :* With *Cold Fusion* [1], HTML documents can be extended with database markup tags and database queries to dynamically populate portions of the document with relational database content as illustrated below:

$$< cfquery\ name = "composer\_query"\ datasource = "world\_opera" >$$

$$select\ *\ from\ opera\ where\ comp = "Mozart"$$

$$< /cfquery >$$

- *HTML Database Programming Extensions :* With *Net.Data*, database queries can be incorporated in an HTML document without additional markup tags, using percent signs as delimiters, as illustrated below:

$$\%function(dtw\_sql)composer\_query()\{select\ *\ from\ opera\ where\ comp = "Puccini"\%\}$$

2.1.3 Hypertext Transfer Protocol (HTTP)

HTTP [11] is a client server, content neutral, Internet communication protocol that runs over TCP/IP. A server URL is associated with each HTTP request. There are a number of HTTP request types including:

- *GET* To retrieve static or dynamic content which could be HTML documents, multimedia files, code fragments such as Java applets or ActiveX controls. Any input parameters are sent from the client to the server as part of the corresponding URL.
- *POST* To retrieve dynamically generated resources. Parameters are sent separately and not in the URL. Pages created through POST are not usually cached in intermediate caching servers.
- *PUT and DELETE* To add, replace or delete resources held on servers. Web standards do not describe the behavior of server systems should multiple users attempt to update resources concurrently, nor is the concept of a collection of updates (a transaction) defined.

One of HTTP's virtues is statelessness which means that servers are not required to associate consecutive requests from the same client and to maintain client context on the server side. Statelessness makes HTTP servers simpler to write, makes intermediate network caching of the results of an HTTP request more straightforward, makes independent client cache navigation possible, and is well suited to occasional access from humans when surfing or when running applications that bypass intermediaries such as human agents who use applications intensively.

Some JDBC drivers communicate with their corresponding JDBC server gateways over HTTP, however there is no agreement between products on the precise format.

In general, access to Web resources is permitted, unless explicitly forbidden. Web clients can filter documents received by checking the content or by referring to third party sites. Few tools are required to run and manage a simple low volume Website made up of one or more HTTP servers along with HTML documents and other Web resources. However, where a Website has static or dynamic content that changes frequently with high volume requirements, then more sophisticated tools and people are currently required.

2.2 Relational Databases

This section highlights some aspects of relational databases that influence Web integration. Relational database systems provide controlled computer program access to tabular data. Each table has distinct rows whose order is immaterial. There are a number of related set based data manipulation languages which are generally embedded in other programming languages. These languages have public descriptions such as SQL and SQLJ [13]. Metadata describing the structure of tables has to be supplied to the database system before tables can be populated and accessed. Additional tables can be defined in terms of views, e.g., by omitting portions of tables and by combining multiple tables. Database metadata and views are also expressed in terms of programming interfaces. Other than by using a programming interface, there is no consensus for many database features such as:

- how relational data and corresponding metadata are stored within a database system, making navigation via a database tool the only option
- how relational data is exchanged between heterogeneous systems such as dissimilar data management systems and with the Web
- how diverse relational database systems communicate across networks
- how relational database resources are named and how database query URLs are constructed

Security is given much emphasis in relational databases. Access is prevented unless specifically permitted and users who access databases are mostly identified in advance. Any routes through intermediate database systems traversed between clients and database servers are usually predefined. Rules governing concurrent database update attempts are defined and transactions are supported. Access to relational databases is stateful, and clients are expected to connect to a database system, perform a number of database requests using the same connections, and then disconnect. Thus, database systems maintain client context at the server side. Because of the elaborateness of database systems, the number of computer instructions to retrieve an item from a database is usually one or two orders of magnitude greater than that for retrieving a file from a typical Web server (excluding instructions that have to do with communications).

Tools are required to create and manage a database system, even one with few users. The effort of populating and managing a database is greater than that for a Website. There are many more people with the ability and tools to create HTML documents and related Web resources that include navigation elements for other people to follow, than there are people who are themselves able to navigate through existing relational databases.

3 Methods of Integration

In this section, we classify the current methods for Web database integration according to the format of information sent to clients (HTML documents or relational result sets).

3.1 HTML Documents (Server Side Processing)

Relational database information is delivered to clients across HTTP formatted in HTML, from Web database gateway software that acts as the interface between a Web server and a database system. HTML documents can be modified many times before they reach their ultimate human or software destination, e.g., via HTML template gateway software to resolve database references, via server and client side script processors, via server directive processors. By the time, an HTML document that includes relational data reaches a client, any relational table structure information is lost and is difficult to reconstruct.

3.1.1 HTML Template Gateways

A document author creates an extended HTML document, known as a template, that contains database access requests, result set formatting and programming constructs, often using product specific markup tags and programming notation.

Database template software acts as a gateway between an HTTP (Web) server and a relational database system, transforming templates into regular HTML documents before passing them on to a Web server for delivery to a client. Examples of database template tools include Cold Fusion and Net.Data. The most frequently cited benefit of template tools is is the ease with which database result sets are formatted. Features of template processing include:

Database constructs for HTML templates

- *Embedded SQL requests:* See the examples in the Web HTML section above.
- *Embedded result sets:* Templates contain support for defining the style and layout of relational result sets in HTML, without the template author making provision for processing each row individually.
- *Database query URLs:* Database query URLs can be used in hypertext links to generate database result sets. Examples were included in the URL section of this article.

General constructs for HTML templates

- *Programming constructs:* Template tools usually include their own programming constructs which means that *presentation services* code is combined with *business services* code that processes data.
- *Variable substitution:* Most tools make it possible for HTML forms input parameters and database content to be substituted in resulting database query and output documents. Variable substitution notation in use currently includes delimiters in the form of hash, dollar, brackets, ampersands, semicolons (depending on the product) such as $\$(input\_composer)$, #input_composer#, &input_composer; Here is a variable substitution example from an Adabas database template:

$$< sqlexec\, name ='' \, query1"$$

$$query = "\, select\, title, first\_perf\ from\ opera\, where\, comp\ = \ '\$input\%'" >$$

$$< /sqlexec>$$

- *Embedded HTML fragments:* Fragments of HTML can be incorporated into an HTML template by naming a file through a URL.
- *User Interaction Linkage:* Template tools have conventions for linking query and update documents with result sets documents, making it possible for users to navigate through a series of related documents. Generally each document contains references to the next appropriate document either through ACTION URLs in an HTML FORM, or in hypertext links. Some template tools make it possible to include a series of database queries and results documents in a single file with multiple entry points.
- *Context services:* Template tools include schemes to specify that user sessions be maintained across user requests, such as through HTML extensions for cookies.
- *File Name Extensions:* Template tools have their own usage for file extensions (.mac, .htx etc.) and the usual .htm or .html file extension is used at the time the page is transformed.

3.1.2 Server Side Script and Program Gateways

A programmer creates an application, in a scripting or programming language, that receives input parameters from clients across the network, via Web servers, issues database access requests, and produces an HTML document as output which is delivered to clients via Web servers.

Examples of server side scripting and programming include C, Perl and Rexx CGI programs [14], Java servlets [16], Active Server Pages [17] and LiveWire [18]. Where an interpretive scripting language is used, the difference between HTML templates and server side scripting becomes indistinct, but can be characterized by an emphasis on markup tags in a template environment and on script language constructs in a scripting environment. In general templates are easier to learn and use, but scripts are more flexible.

General HTML templates, in particular variable substitution notation, can be used in conjunction with regular server side programs. Thus, a server application substitutes variables in a template typically when it has completed its processing. Using templates makes it possible for separate individuals to design output documents (templates) while programmers write the code to populate the template.

3.2 Relational Result Sets (Client Side Processing)

A programmer writes a client application, e.g., a Java or ActiveX application, that communicates with a relational database across the network. An intermediate gateway is often used. Relational result sets are delivered to the client application, one row at a time, for further processing and formatting.

With Java, the client portion of an application can be pre-installed on a client system or downloaded from a Web server using HTTP, to run as an applet. An applet is a collection of Java classes that run within the context of another application such as a Web browser. For security reasons browsers prevent applets from performing a number of functions, such as:

- Communicating with software running on a system that is not the same as the one running the Web server from where the applet was downloaded
- Invoking non-Java code (native methods)
- Writing to disk

Some browsers have user options to override these restrictions in Intranet environments, but most browsers enforce the checks. Alternatively, signed applets can be used to overcome restrictions. Applets can access relational databases, subject to the security restrictions, through JDBC or SQLJ. Often, a separate JDBC gateway processes inbound JDBC requests and routes them to the appropriate database system.

4 Some System Integration Issues

This section describes system issues that affect the integration of relational database systems with the Web.

4.1 Thread and Process Management

Web servers provide ways of pooling resources across client requests each with its own performance characteristics. Here are just some of the ways operating system threads and processes are handled:

- *Single process per request:* such as the CGI, where one process is created and dismantled for each client request incurring performance overheads but which nevertheless works well in low volume environments.
- *Multiple long running processes:* such as Fast CGI [15] where a pool of processes is created, and optionally initialized with database connections when the server starts. Each process is used by multiple clients serially. Memory leaks do become apparent in this environment.
- *Single thead per request:* such as the mainframe CICS transaction processing system [19] when it responds to client requests (including HTTP).
- *Multiple long running threads:* such as Web server APIs (GWAPI, ISAPI, NSAPI) where a pool of threads is created and optionally initialized. Each thread is used by multiple clients serially. Side effects include memory overwrites and memory leaks becoming noticeable. (Java servlets can run in single-threaded or multi-threaded processes.)

Web database gateway software often tries to hide the differences between these server environments from template and script authors, and to give the illusion that each template or script is running alone. Database management systems typically manage their own thread and process structures internally, maintaining user contexts for as long as possible while minimizing server resource usage, to reduce request, session initiation and termination overheads. Some Web database gateway software also provides similar long term threading and process management structures. Long lasting resource pooling mechanisms, through Web servers and gateways, can have unexpected effects on database management system internal structures that are tuned for inbound user contexts that endure for minutes rather than days.

4.2 Security

- *Authentication:* There is no single way for performing authentication on the Web. By the time a Web request reaches a database system, the request has usually been authenticated.
- *Access Control:* It is often impossible to predict who all the users of a Web application will be in advance, and the numbers of occasional users may be very large, Usually database access control in Web database applications is performed with general purpose user identifiers rather than specific user identifiers associated with individuals. Most of the time, database gateway connection pools described in the next section require general purpose user identifiers to run efficiently.
- *Encryption:* Communication flows between clients and servers can be encrypted. SSL (Secure Sockets Layer) is the most commonly used encryption method on the Web.
- *Firewalls:* Firewalls are one or more computer systems that monitor and prevent certain kinds of inbound and outbound communication requests, usually between Intranets and the Internet. Few firewalls support database communication protocols because the protocols are not well understood outside database products. Thus, JDBC applets are suitable for Intranets, but they cannot be used easily by people behind firewalls to access servers outside firewalls unless the applets use regular Internet protocols to communicate with JDBC gateways.

4.3 Context or State Management

A Web database gateway has two interfaces, one for the Web client (often the human user) through a Web server, and one for the database system. Maintaining context on behalf of a client is called *session management*, and maintaining context on behalf of a database system is called *connection management*.

- *Session Management:* Users expect session support and do not want to type in the same information repeatedly such as their names. In order to support sessions, gateway software has to be able to identify when a session starts and when it ends, how to transmit a session identifier between a client and server, where to store any session state (client or server), and how to handle unexpected session errors. A number of types of session identifiers are widely used on the Web, such as URL extensions, cookies, hidden fields in HTML forms. Often gateway software provides explicit support for sessions.
- *Connection Management:* Database systems assume that consecutive requests from a single thread or process, within one database connection and with the same user identifier, are issued on behalf of one individual, and hence they bypass certain pieces of processing. Web database gateways often take advantage of this assumption by creating a database connection pool with threads or processes that are connected to the database system for the lifetime of the gateway or Web server. An element in the pool (a thread or process) is usually associated with one user session and cannot be accessed by other users. Where Fast CGI or Web server APIs are used, the Web server resource pool can become the database connection resource pool by automatically connecting the Web server processes or threads to the database system at Web server startup.

4.4 Transactions (Unit of Work)

Transactions with ACID (Atomicity, Consistency, Integrity and Durability) characteristics are a feature of database management systems. Transactions ensure that only consistent data is externally visible, even when multiple users are updating the same data concurrently, when errors occur in the system and when multiple databases at multiple sites are involved. Because of the concurrency control mechanisms for transactions and the resources they consume, and because of the nature of many applications, it is advisable to keep transactions short and in particular not to extend the duration of a transaction beyond one user interaction. The same considerations prevent ACID transactions from being used across organisations and companies. In Web database applications, it is usual for a transaction to be initiated and terminated by the Web database gateway. Typically, one user session is made up of multiple transactions. There have been attempts to introduce transaction identifiers into HTTP, making it possible to extend the notion of a transaction to resources on the Web, but HTTP transaction identifiers are not in general use yet. In Oracle, database query URLs can include transaction initiation and termination information.

4.5 Caching

The primary purpose of caching on the Web is to avoid transmitting Web resources unnecessarily across networks. Caching can occur at many points and they include:

- At a client: A Web browser cache that is typically associated with a single client
- Near a group of clients: A Web proxy cache which is typically associated with and placed near a group of clients accessing many servers
- Near a group of servers: A Web reverse proxy cache which is typically associated with and placed near a group of servers being accessed by many clients
- At a server: This can be similar to database caching where the main reason for caching is to avoid disk accesses. Dynamic content can also be cached to eliminate processing time needed for repeated generation.

LRU (Least Recently Used) algorithms are often used in proxy caches in combination with heuristics such as if a resource was updated recently, then it is more likely to be updated again in a short time interval, or prefetching according to previous access patterns. It is also possible for clients to refresh pages automatically at intervals. Web caching uses information transmitted as part of the HTTP protocol to determine whether to cache the resource and how long to cache it for. However, proxies are also being used to augment, modify or filter the information transmitted between clients and servers transparently. For example, the Webcosm [21] link manager can be configured as a proxy (known as an *intermediary* or a *facade*), to incoporate links into existing documents without changing the documents and taking into account users' interests. Currently, there are few systems that try to provide consistency between relational database content and with the content of Web caches.

5 XML and Relational Databases

XML (Extensible Markup Language [9]) is a modification of SGML [8] intended to make it straightforward to define domain specific markup tags (vocabularies) and to develop programs that process documents containing these tags. Processing programs can understand the specific tags or can be general purpose. HTML is an SGML (not XML) application with its own vocabulary of tags that is useful in a wide range of documents, as well as features such as FORMS that are used in Web applications. Fragments of XML and HTML are likely to appear in a single document. Some important aspects of XML are being able to:

- define and use application-specific markup tags (vocabularies) in XML documents
- create new or use existing document type declarations (DTDs) allowing software to validate the format of corresponding XML documents
- create or use existing stylesheets making it possible to present the same information in different ways
- specify a variety of link types between documents
- combine multiple vocabularies from different sources in a single document through appropriate naming (namespaces)
- refer to DTDs, stylesheets, namespaces etc. in XML documents through URLs

Examples of XML vocabularies are Chemical Markup Language, Information and Content Exchange (for exchanging information between Websites) and Web Interface Definition Language (analogous to IDL). Vocabularies for relational databases include some of the constructs in HTML templates for databases, such as database query and result set tags. The availability of XML formatted result sets makes the database templates approach (server side programming) and result sets approach (client side programming) described earlier more alike as both can manipulate XML fragments. It also makes it easier to exchange relational data with other systems in unplanned ways.

There are application design considerations when database updates arise some time, minutes say, after an XML document was generated from a database. Update decisions can be made based on information which may have already changed in the corresponding database. In the absence of check-in checkout and related constructs in database systems, applications have to detect and recover from the inconsistencies. Some XML formatted result sets persist for long periods, weeks or months say, such as in catalog applications, and the XML formatted information is refreshed or updated intermittently. The generated XML can then be stored in a cache or database for further manipulation and transformation [20].

XML style sheets describe the rules for translating XML tagged information into another format such as into another XML vocabulary or into HTML for display. Initially intended for transformations into human display formats, style sheets or database view mechanisms can form the basis for transformations between XML vocabularies at servers, between servers, as well as at clients. Multiple transformations can take place before data reaches its destination.

Creating and following links between diverse pieces of electronic information, now known as hypertext and hypermedia, was described by Vannevar Bush in 1945 [2]. Links are what make the Web a Web. Relational databases incorporate the notion of links mainly through programs matching key and attribute values within databases of the same type, or through referential integrity. Federated database software supports similar notions across heterogeneous databases.

XML linking makes it possible for applications that produce and process XML tagged information to manage links externally from both source and target resources, to support many-to-many links, to materialise linked to resources in a variety of ways, at different times in accordance with context information, to name and construct multiple navigational views on a set of resources.

In effect indexes created for Internet search engines and for relational databases are collections of externally managed links that provide direct access to resources containing specific terms, with particular names, or with particular attributes. However, relational databases do not typically expose their indexes for general use nor do they have universal naming schemes. Although few database tools automatically generate hypertext links for database result sets, hypertext links can play a part in database applications through linking primary and foreign key values and drilling down into categories of information when data mining.

6 Conclusions

The success of the Web illustrates the advantages of agreed extensible protocols and naming conventions, general purpose software and independence from underlying data structures. Simplifying and adapting well established and widely used standards, together with decentralized collaborative development, are hallmarks of Web protocols and software [5].

Currently, programming effort is required to make relational data accessible on the Internet and on Intranets. Disparate database markup extensions, variable substitution symbols, naming schemes and database query URLs demonstrate that there is no general agreement for Web related database access. The output from database programming interfaces currently requires manipulation before it can be presented to humans and to heterogeneous systems. Reaching consensus on Web based input and output formats (and ways of defining the formats) for established database interfaces is a step towards making relational data more easily accessible from the Web in a generic way.

Increased emphasis is being placed on database systems that manage structured and semi-structured data and which participate on networks as Web resources. With unanimous support for data exchange in XML, there is considerable opportunity for improving the integration of database data with other Web resources in a universal way, including better compatibility between diverse Web integration methods through XML vocabularies for relational databases, and automated conversions to other vocabularies.

The length of time that multiple copies of the same information exist in electronic form outside the control of database management systems, continues to grow. Data is materialised in formats that are not necessarily related to the way it is stored in relational databases. Instead, it is transformed into self defining linked fragments making it more readily accessible to humans or to general purpose software often held in network caches where further transformations occur and where the beginnings of network based collaborative data management and exchange systems are taking place. Irrespective of the facilities incorporated into current database systems, the Web and related software will eventually assimilate their content, thereby providing new data management opportunities.

Apology

In an article on archiving the Internet [4], Brewster Kahle claimed that the average lifetime of a URL is just 44 days making it likely that some URLs listed below will not be available at the time you read this article. I regret any inconvenience caused.

References

[1] **http://www.allaire.com/** : Burke, J., Web Databases with Cold Fusion, McGraw Hill, 1997

[2] **http://www.isg.sfu.ca/%7educhier/misc/vbush** : Bush, V., As We May Think, Atlantic Monthly, pp 101-108, July 1945

[3] **http://www.mkp.com/books_catalog/books.htm** : Chamberlin, D., Using the New DB2, Morgan Kaufmann, 1996

[4] **http://www.sciam.com/0397issue/0397kahle.html** : Kahle, B., Preserving the Internet, Scientific American, March 1997

[5] **http://www7.conf.au** : Khare, R. and Rifkin, A., The origin of (document) species, Proceedings of the 7th International WWW Conference in Australia, April 1998

[6] **http://www.oracle.com/** : Papaj, R. and Burleson, D., Oracle Databases on the Web, Coriolis Books, 1997

[7] **http://www.w3.org/Addressing/URL** : The URL at the W3 Consortium

[8] **http://www.sil.org/sgml/** : The SGML pages at SIL

[9] **http://www.w3.org/XML** : eXtensible Markup Language at the W3 Consortium

[10] **http://lcweb.loc.gov/global/internet/html.html** : HTML at the Library of Congress

[11] **http://www.w3.org/Protocols/Overview.html** : HTTP at the W3 Consortium

[12] **http://java.sun.com/products/jdbc/** : JDBC at Sun

[13] **http://www.oracle.com/st/products/jdbc/sqlj/** : SQLJ at Oracle

[14] **http://hoohoo.ncsa.uiuc.edu/cgi/** : Common Gateway Interface at NCSA

[15] **http://www.fastcgi.com/** : Fast CGI

[16] **http://java.sun.com/products/jdk/1.2/docs/ext/servlet/** : Java servlets at Sun

[17] **http://www.microsoft.com/** : Microsoft Active Server Pages

[18] **http://www.netscape.com/** : Netscape Livewire

[19] **http://www.software.ibm.com/** : CICS, DB2 and Net.Data at IBM

[20] **http://www.darmstadt.gmd.de/oasys/projects/** : The OASYS multimedia projects

[21] **http://www.webcosm.com/** : Webcosm link management

Web-Site Management: The Strudel Approach

Mary Fernandez
AT&T Labs
mff@research.att.com

Daniela Florescu
INRIA Roquencourt
dana@rodin.inria.fr

Alon Levy
Univ. of Washington
alon@cs.washington.edu

Dan Suciu
AT&T Labs
suciu@research.att.com

1 Introduction

The World-Wide Web (WWW) has become a prime vehicle for disseminating information. As a result, the number of large Web sites with complex structure and that serve information derived from multiple data sources is increasing. Managing the *content* and the *structure* of such Web sites presents a novel data management problem, which has not been previously considered by the database community.

To understand the problem, consider a Web-site builder's tasks: (1) choosing and accessing the data that will be displayed at the site, (2) designing the site's structure, i.e., specifying the data contained within each page and the links between pages, and (3) designing the visual presentation of pages. In existing Web-site management tools, these tasks are, for the most part, interdependent. Without any site-creation tools, a site builder writes HTML files by hand or writes programs to produce them and must focus simultaneously on a page's content, its relationship to other pages, and its visual presentation. As a result, several important tasks, such as automatically updating a site, restructuring a site, or enforcing integrity constraints on a site's structure, are tedious to perform. To support these tasks naturally, we view the problem from a data management perspective.

We decided to study the web-site construction and management problem from a database perspective. We have developed the STRUDEL system [6, 7], which applies concepts from database management systems to Web-site creation and management. In particular, STRUDEL supports declarative specification of a Web site's content and structure and automatically generates a browsable Web site from a specification. STRUDEL's key idea is separating the management of a Web site's data, the management of the site's structure, and the visual presentation of the site's pages.

Broadly speaking, using STRUDEL, the site builder first creates an integrated view of the data that will be available at the site. The Web site's raw data resides either in external sources (e.g., databases, structured files) or in STRUDEL's internal data repository. In STRUDEL's mediator component, as in all of its other components, all external or internal data is modeled as a labeled directed graph, which is the model commonly used for semistructured data [1, 4]. A set of source-specific wrappers translates the external representation into the graph model. The integrated view of the data is called the *data graph*. Second, the site builder declaratively specifies the Web site's structure using a *site-definition query* in STRUQL, STRUDEL's query language. The result of evaluating the site-definition query on the data graph is a *site graph*, which models both the site's content and structure. Third, the builder specifies the visual presentation of pages in STRUDEL's HTML-template language. The HTML generator produces HTML text for every node in the site graph from a corresponding HTML template; the result is the browsable Web site.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

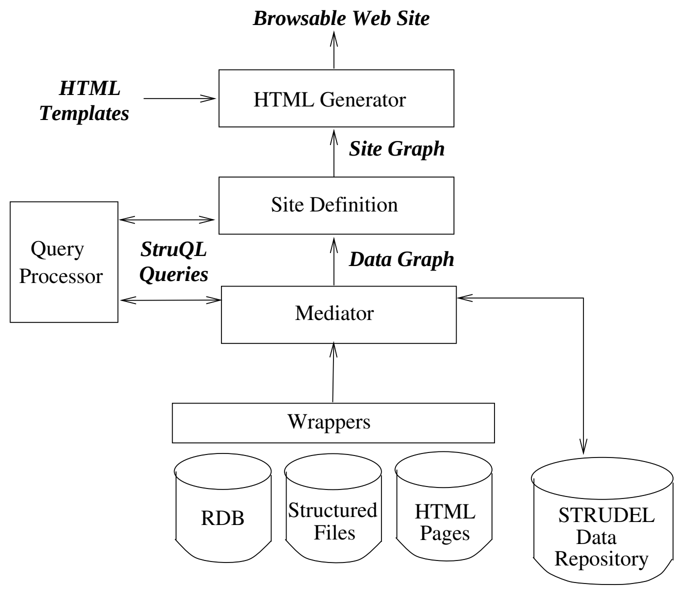

Figure 1: STRUDEL Architecture

STRUDEL is based on a semistructured data model of labeled, directed graphs. This model was introduced to manage *semistructured* data, which is characterized as having few type constraints, irregular structure, and rapidly evolving or missing schema [1, 4]. This data model was appealing for STRUDEL, because Web sites are graphs with irregular structure and non-traditional schemas. Furthermore, semistructured data facilitates integration of data from multiple, non-traditional sources.

As it stands, STRUDEL provides several benefits. Since a Web site's structure and content are defined declaratively by a query, not procedurally by a program, it is easy to create multiple *versions* of a site. For example, it is possible to build internal and external views of an organization's site or to build sites tailored to novice or expert users. Currently, creating multiple versions requires writing multiple sets of programs or manually creating different sets of HTML files. In STRUDEL, a site builder produces multiple sites by applying different site-definition queries to the same underlying data or by creating multiple visual presentations for the same site graph. STRUDEL's architecture also supports evolution of a Web site's structure. For example, to reorganize pages based on frequent usage patterns or to extend the site's content, we simply rewrite the site-definition query. Declarative specification of Web sites can offer other advantages. For example, it becomes possible to express and enforce integrity constraints on the site and to update a site incrementally when changes occur in the underlying data.

STRUDEL clearly separates the three tasks of building Web sites and is the first system that supports declarative specification of a site's content *and* structure. Other recent research prototypes have discussed the separation of the three tasks, but do not not support declarative specification of content or structure [3, 8]. Other research projects support declarative specification, but merge the tasks [2, 5]. Commercial tools such as Vignette's StoryServer and those provided by major database vendors separate the management of the underlying data from its visual presentation. Individual pages or sets of related pages are constructed dynamically by evaluating queries that are embedded in HTML templates; query results are merged into HTML templates to produce pages. Other products provide graphical user interfaces that support drag-and-drop editing of individual pages (e.g., Microsoft's FrontPage, NetObjects' Fusion) or of the structure between individual pages (e.g., Elemental's Drumbeat).

This intense activity in research and industry indicates that Web-site management is an important problem, and because its central issue is management of site content and structure, it should be of interest to the database community. Given this, we set out to gain experience quickly using STRUDEL so that we may understand which aspects of Web-site management benefit most from application of database concepts and identify the critical research issues we should focus on in this realm. In this paper, we briefly describe the STRUDEL system, the lessons we learned from applying the system, and outline the directions we consider important for future research.

2 The Strudel System

Strudel's architecture is depicted in Fig. 1; rectangles depict processes and emboldened terms specify the inputs and outputs of the processes.

We describe the architecture bottom up from the site builder's perspective. Using Strudel, the site builder first creates an integrated view of the data that will be available at the site. The Web site's raw data resides either in external sources (e.g., databases or structured files such as Word or Excel documents) or in Strudel's internal data repository. In Strudel, all external and internal data is modeled as a labeled directed graph. In this model, the database consists of objects connected by directed edges labeled with string-valued attribute names. Objects are either nodes, identified by a unique object identifier or are atomic values, such as integers, strings, and files. Objects are grouped into named collections, which are used in queries.

A set of source-specific wrappers translates the external representation into Strudel's graph model. The site builder can produce an integrated view of several sources by writing a query in StruQL, Strudel's query language. The integrated view of the data is called the data graph. Next, the site builder declaratively specifies the Web site's structure using a site-definition query also in StruQL. The result of evaluating the site-definition query on the data graph is a site graph, which models both the site's content and structure. Third, the builder specifies the visual presentation of pages in Strudel's HTML-template language. The HTML generator produces HTML text for every node in the site graph from a corresponding HTML template; the result is the browsable Web site.

Strudel provides several benefits. Since a Web site's structure and content are defined declaratively by a query, not procedurally by a program, it is easy to create multiple versions of a site. For example, it is possible to build internal and external views of an organization's site or to build sites tailored to novice or expert users. Currently, creating multiple versions requires writing multiple sets of programs or manually creating different sets of HTML files. In Strudel, a site builder produces multiple sites by applying different site-definition queries to the same underlying data or by creating multiple HTML renderings of the same site graph. Strudel's architecture also supports evolution of a Web site's structure. For example, to reorganize pages based on frequent usage patterns or to extend the site's content, we simply rewrite the site-definition query.

2.1 Example Web Site

The following example shows how one author's homepage site is generated by Strudel. The main data source is the author's Bibtex bibliography file. The homepage site has four types of pages: the root page containing general information, a page containing all paper abstracts, and pages containing summaries of papers published in a particular year or category. We describe the first two steps of the site-definition process: creating the data graph from a Bibtex file and defining the site graph in StruQL.

Fig. 2 contains a fragment of the site's data graph and was generated by a Bibtex wrapper; the wrapper converts Bibtex files into a Strudel data graph. Both objects, pub1 and pub2, are members of the Publications collection. It is important to note here that data in Strudel's graph data model does not have a fixed schema, which facilitates integration of data from multiple sources; for example, objects in the same collection may have different representations as do pub1 and pub2 below.

The homepages' site graph is defined by the query in Fig. 3. The first clause creates two new objects called RootPage and AbstractsPage and creates a link between them. The second clause (lines 6-7) creates two new objects, AbstractPage(x) and PaperPresentation(x) for each member x of the Publications collection; these objects contain the publication's information that will appear in different parts of the site. For example, the expressions on lines 9-10 copy all of x's attributes and values into the new objects. The link clause also encodes inter-page

```
object pub1 in Publications {
  title "Specifying Representations..."
  author "Norman Ramsey"
  author "Mary Fernandez"
  year 1997
  month "May"
  journal "Transactions on Programming..."
  pub-type "article"
  abstract is text "abstracts/toplas97.txt"
  postscript is ps "papers/toplas97.ps.gz"
  volume "19 (3)"
  category "Architecture Specifications"
  category "Programming Languages"
}
```

```
object pub2 in Publications {
  title "Optimizing Regular..."
  author "Mary Fernandez"
  author "Dan Suciu"
  year 1998
  booktitle "Proc.  of ICDE"
  pub-type "inproceedings"
  abstract is text "abstracts/icde98.txt"
  postscript is ps "papers/icde98.ps.gz"
  category "Semistructured Data"
  category "Programming Languages"
}
```

Figure 2: Fragment of data graph for example site

```
// Create Root & Abstracts page and link them
CREATE RootPage(), AbstractsPage()
LINK RootPage()−>"AbstractsPage"−>AbstractsPage()

// Create a presentation for every publication x
WHERE Publications(x), x−>l−>v
CREATE PaperPresentation(x), AbstractPage(x)
LINK
  AbstractPage(x) −> l −> v,
  PaperPresentation(x) −> l −> v,
  PaperPresentation(x)−>"Abstract"−>AbstractPage(x),
  AbstractsPage() −>"Abstract" −> AbstractPage(x)

{ // Create a page for every year
  WHERE l = "year"
  CREATE YearPage(v)
  LINK
    YearPage(v) −> "Year" −> v
    YearPage(v)−>"Paper"−>PaperPresentation(x),

    // Link root page to each year page
    RootPage() −> "YearPage" −> YearPage(v)
}

{ // Create a page for every category
  WHERE l = "category"
  CREATE CategoryPage(v)
  LINK
    CategoryPage(v) −> "Name" −> v,
    CategoryPage(v)−>"Paper"−>PaperPresentation(x),

    // Link root page to each category page
    RootPage() −> "CategoryPage" −> CategoryPage(v)
}
```

Figure 3: Site definition query for example homepage site

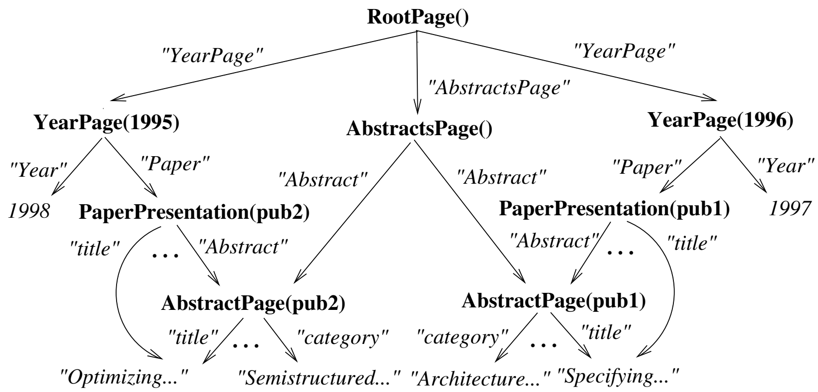

Figure 4: Fragment of site graph for example homepage site

structure. On line 12, the general abstracts page is linked to the abstract page of each publication (AbstractPage(x)). The nested where clause (lines 14-23) creates a page for each year associated with a publication; the link clause associates each PaperPresentation object with its corresponding YearPage. Lastly, the root page is linked to each year page. A similar clause creates a page for each publication category and links category pages to PaperPresentation objects. Note that the site-definition query only specifies what information will be available in the site and what relationships exist between pages in the site; it does not specify anything about the site's visual presentation. The result of applying a site-definition query to a data graph is another graph in Strudel's data model; this permits us to create site by composing multiple queries.

Fig. 4 depicts a fragment of the generated site graph; for clarity, it excludes the result of the last nested clause that produces category pages. Note that the site graph encodes both the site's content and its structure. For example, the PaperPresentation objects have links to paper titles and to their associated abstract pages. All leaf objects contain page content, e.g., the titles of publications. Declarative specification of the site graph is powerful, because the site builder can specify its structure in any order he chooses. For example, he can define the pages "top down" from the root, or first define each group of related pages and then link them.

3 Experiences with Strudel

After implementing STRUDEL's first prototype, we wanted to evaluate STRUDEL's methodology and our choice of the semistructured data model. First, we considered whether our premise that the three tasks of Web-site creation can and should be separated holds in practice. Specifically,

- Is there always a clear separation between these tasks? If not, in which cases do their mutual dependencies make separating them counter productive?
- For what kinds of Web sites is STRUDEL most effective? How useful is the ability to explicitly and declaratively manage a Web site's structure?

We have had both practical and exploratory experiences with STRUDEL. In our practical experience, we used the STRUDEL prototype to create sites for individual users and for two organizations (AT&T Research being the largest one), and to create a version of the CNN Web site for demonstration purposes. In our exploratory experience, we described our methodology and demonstrated our prototype to several potential STRUDEL users.

We learned several important lessons concerning STRUDEL's methodology from our study. Separating the management of the underlying data from other Web-management tasks is the basis for several commercial products, e.g., most commercial relational and object-oriented databases provide Web interfaces to their systems.

STRUDEL provides two other important features: the abilities to integrate data from *multiple* sources and to incorporate unstructured sources (e.g., structured files). The AT&T Research site, for example, integrated five data sources.

Isolating the management of a site's structure was also important. For example, CNN's Web-site group is building a specialized tool for managing site structure. We also found that building complex Web sites is an iterative process in which the site structure evolves over time. For example, creating the AT&T and Rodin sites required several iterations. Declarative specification of the site's structure enables easy changes to a site. Finally, STRUDEL is most effective when multiple versions of a site are built from the same underlying data. For instance, once we built AT&T's internal research site, building the external version was trivial.

Separating management of the site's structure and its visual presentation is more subtle. This separation simplifies creating multiple versions of a site especially when the site's structure is the same in all versions, but its visual presentations differ. In this case, all versions share one site graph, but each version has its own HTML templates. It is not always clear, however, which aspects of a site should be encoded as structure or as visual presentation. For example, the AT&T external site is derived from the internal site by excluding the attributes of some objects in the generated pages; in this case, it is easier to create HTML templates that omit these attributes than it is to create a new site graph that explicitly excludes those attributes. Consider the order of articles or the placement of images in a page at the CNN site. Such information could be encoded in the visual presentation or in the site's structure. For CNN, managing this information is crucial, because they consider these editorial elements a primary value of their site.

To characterize the sites for which STRUDEL is most useful, we consider two criteria: the amount of data they contain and their structural complexity. Measuring the amount of data in a site is straightforward. One possible measure of structural complexity is the number of `link` clauses in the site-definition query. In current practice, an analagous measure of site complexity is the number of CGI-BIN scripts required to generate a site.

We observed that STRUDEL is most useful for sites that have complex structure and whose structure is dependent on the underlying data. For example, the CNN Web site contains a large number of articles. Although the disposition of an article in a site is complex (i.e., it appears in several formats on different pages and is linked to many other pages), the structure is uniform for all articles in the site. This uniformity also applies to all people in the AT&T site and all publications in the example homepage sites. In summary, when a site has simple structure and little data WSYWIG tools such as Microsoft FrontPage or NetObjects Fusion are appropriate choices. When a site contains large amounts of data, but has simple structure, then a tool that provides a Web-based interface to a database is appropriate. However, when there is a lot of data and the site structure is complex, STRUDEL is most appropriate.

4 Future Research

Our experience helped us identify research problems of practical and theoretical interest, which we outline here. They address issues of STRUDEL's applicability to dynamically generated sites, its scalability to larger sites, its usability as an end-user tool, and its interoperability with existing tools.

Incremental generation of the Web site: In STRUDEL's prototype, we precompute a Web site by completely materializing its site graph. Many Web sites, however, cannot be precomputed, because they depend on user input that is not available statically or because the underlying data sources are too large. Currently, STRUDEL does not support dynamically generated sites. In practice, dynamic generation is supported by often large groups of loosely related CGI programs. Supporting dynamic evaluation would eliminate writing such programs by hand.

Although we can decompose a site-definition query into multiple, dynamic queries, and we have theoretical techniques for optimizing these queries, implementing dynamic evaluation requires significant systems-design effort. For example, our optimization techniques cache query results to reduce "click time" for future queries;

these results essentially encode state required by the STRUDEL query processor. An open problem is how and where this state should be stored: in a client-side browser and/or a server-side query processor.

Graphical interface to the query language: Not surprisingly, many potential users of STRUDEL asked whether we can provide a friendly visual interface for specifying queries, instead of having to write STRUQL queries by hand. Clearly, a better interface is needed, probably in the spirit of Query By Example.

Integration with the programming environment: Many commercial tools exist for Web-site creation and management. We do not presume that STRUDEL will replace *all* of them, therefore an important practical issue is how to integrate STRUDEL with existing tools. In particular, developing the appropriate API to STRUDEL may be the best way to incorporate it into tools that Web-site builders currently use.

References

[1] S. Abiteboul. Querying semi-structured data. In *Proceedings of the ICDT*, 1997.

[2] G. Arocena and A. Mendelzon. Weboql: Restructuring documents, database and webs. In *Proceedings of International Conference on Data Engineering*, pages 24–33, 1998.

[3] P. Atzeni, G. Mecca, and P. Merialdo. To weave the web. In *Proceedings of VLDB*, 1997.

[4] P. Buneman. Semistructured data. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona*, pages 117–121, 1997.

[5] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your mediators need data conversion. In *Proceedings of SIGMOD*, 1998.

[6] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. System demonstration - strudel: A web-site management system. In *ACM SIGMOD Conference on Management of Data*, 1997.

[7] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. Catching the boat with strudel: experience with a web-site management system. In *Proceedings of SIGMOD*, 1998.

[8] P. Paolini and P. Fraternali. A conceptual model and a tool environment for developing more scalable, dynamic, and customizable web applications. In *Proceedings of EDBT Conference*, Valencia, Spain, 1998.

Connecting Diverse Web Search Facilities

Udi Manber Peter A. Bigot
Department of Computer Science
University of Arizona
Tucson, AZ 85721
{udi,pab}@cs.arizona.edu

Abstract

The World Wide Web is now providing hundreds of powerful search facilities, often for free, enabling people to access an enormous amount of information. The most successful search facilities to date are the global flat databases that attempt to put everything in one place. We believe that this approach will not scale. This paper describes techniques and software that we developed to connect together many different diverse search facilities, allowing people to focus and customize their search much better and increase the precision of the results.

1 Introduction

With the explosion of available information, search is becoming one of the most important computer activities. Getting the right information at the right time with minimal effort is the main competitive edge in many businesses. The most commonly-used search facilities on the web are the global search engines such as Altavista, Infoseek, Excite, HotBot, and Lycos. They collect as much of the web as they can into one large database and provide keyword-based search. They have become amazingly powerful, but they are still lacking. Taking the whole universe as one flat data space and searching it with keywords has inherent limitations. Another even more successful approach is that of Yahoo, which collects and classifies web pages manually with the help of librarians. The noise generated by this approach is much smaller but the coverage is smaller too, and it is still often time consuming to find things. The challenge is to provide users with ways to focus and customize their search better without making it too difficult or too inefficient.

The first part of the paper describes a method of conducting search on the web that is based on a two-level search idea. It strikes a balance between flat global search and specialized databases by connecting together many diverse search facilities into one common interface. It also strikes a balance between automated blind collection and manual collection of quality information. We have implemented this method and provide it as a service called the Search Broker [9]. The second part of the paper describes the next generation of this approach, called the "Universal Search Interface" (USI), which is currently work in progress.

Our research emphasizes simplicity suitable for non-specialist users, rather than complexity and generality. We want to provide access through one interface, in many different customizable ways, to hundreds of existing search facilities. Although this problem can be viewed as an instance of database integration, which has been studied extensively, we are not attempting to integrate all the Web's different databases *per se*: for example,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

we are not attempting to support queries like the join of two separate databases. Instead, we facilitate querying many databases together, either in parallel or sequentially. By limiting ourselves to issues related to straightforward search tasks rather than complex database queries, we believe we can design a system that will be powerful (although not as powerful as an integrated database), and more important, easy to use.

2 The Search Broker

The weaknesses of the global web search engines are most glaring when one looks for answers to specific reference questions, which are hard to answer based on keywords and flat search. For example,

1. How much fat is there in a pepperoni pizza?
2. How do you say "search" in Latin?
3. How do you delete a directory in UNIX?
4. Give me a list of hotels in Phoenix.

The Search Broker is based on the idea of a *two-level search*. Instead of always searching the same all-encompassing database, imagine having specific databases for specific topics. The search will consist of two phases: In the first phase, the search is after the right database, and in the second phase the desired information is sought within this database. This is not a new approach, of course. It is similar to using the library subject card catalog to find the right shelf, or using Yahoo to find the right category. The novelty of the Search Broker is that it combines the two phases into one regular search in a way that makes the process very easy and very powerful for users. The resulting tool provides search features that are not otherwise available in any one place on the web.

Consider again the four questions listed above. The first one has to do with nutrition, the second with Latin, the third with UNIX, and the fourth with hotels. These are the most important characteristics of these questions. The person who asks the question can usually pinpoint its subject; perhaps not precisely, but often closely. For example, the questions above could be answered more precisely if they were of the form

1. "Subject: nutrition; Query: pizza", or
2. "Subject: english-to-latin; Query: search", or
3. "Subject: unix; Query: delete directory", or
4. "Subject: hotels; Query: Phoenix".

Knowing the right subject may be tricky. Some users may input "calories" instead of "nutrition", or "accommodations" instead of "hotels". We only ask that some information about the subject be included in the query. We also replace the rather complex syntax above with a very simple query, as we will show shortly.

The Search Broker works as follows. We collected over 400 different search providers that we judged to have reasonable general appeal. (This is an on-going process, of course; we expect a fully-operational system to have thousands of servers.) Each such search server covers a certain subject or category (such as "nutrition," "latin," "unix," or "hotels"). Each such category is identified by one or two words, and it is also associated with a list of *aliases* that people may think about when searching for that subject. So "nutrition" can be associated with "calories," and "hotels" with "motels," "lodging," and "accommodations." The collection of search engines and the assignment of the words and aliases that identify them are done manually by a librarian. This is a part that we intentionally do not wish to automate. The role of editors, reviewers, interpreters, and librarians has been rather limited in the web, mainly because of the scale. Finding paradigms that will allow significant librarian input while supporting the scale of the web is increasingly important. The two-level approach is promising because the number of subjects does not grow too fast (as opposed to the number of web pages or the number of web sites).

A user query is made of two parts corresponding to the two phases of the search. In the current implementation both parts are combined into one simple box. To answer the questions above you type “nutrition pizza,” “latin search,” “unix delete directory,” or “hotel Phoenix,” and you get direct results from the appropriate search services. Given a query like “hotel phoenix”, the Search Broker performs the following steps:

1. It searches its own database for subjects and aliases and finds the search engine corresponding to “hotel”. In the current implementation, the subject must be the first word in the query, mainly because we want users to identify the subject and think about it. We could easily select any word in the query that matches a subject and try it (or all of them).

2. After identifying the particular search engine, the rest of the query is reformatted to the form expected by that search engine. This step can sometimes be quite complicated; see [9] for details.

3. An HTTP request is sent to the search engine with the appropriate fields that match the query.

4. The results of the query are sent back to the user.

This simple minded approach turns out to be extremely powerful. The proliferation of search software, often for free, made it easy to provide search capabilities on many web sites. (We are proud to be partially responsible for that with our Glimpse [11], GlimpseHTTP, WebGlimpse [10], and Harvest [2] systems.) Within the last year thousands of search servers have been added. Most of them deal with very limited specific information (e.g., they search the content of one site), but many provide professional content in areas of general interest. The trend to connect existing databases to the web will continue. There are already so many high quality search facilities that people cannot keep track of them through bookmarks and favorite lists.

The list of currently available subjects is included in the home page of the Search Broker, and there are also facilities to search the Search Broker’s own database. Let’s see some examples of queries to demonstrate the power of the approach:

stocks ibm gives the current value of IBM’s stock plus links to corporate information and news.

patent object oriented gives abstracts of all patents (from 1971 to present) with these keywords.

howto buy a car gives practical advice about buying used and new cars

fly sfo jfk gives all scheduled flights between San Francisco and New York JFK.

polish-english wonderful tells you that “cudowny”, “zadziwiajacy”, and “godny podziwu” match the adjective “wonderful”.

nba-salaries michael gives the salary of Michael Curry (and all other Michaels playing in the NBA).

car-price 1994 Geo, Prizm gives the blue book value for this model (“,” is used as a delimiter).

email bill gates gives email addresses for that name (yes, it includes the one you are thinking of).

travel fiji gives a lot of useful information about travel in Fiji.

expert computer algorithm gives a list of experts who put “computer algorithms” in their areas of specialty.

demographics health food gives several articles on issues of demographics (and marketing) related to health food.

The Search Broker approach is not a magic bullet, and we do not expect it to replace any of the existing search mechanisms. But we believe that it complements them very well. It explores the middle ground between completely automated search systems on the one hand and manual collection of information on the other. It opens the door to a more significant involvement of experts in the organization of search. The web searching problem is too big to be solved by one tool or even one model. If one is not sure what one is looking for, browsing works best, and Yahoo presents the right approach. If one is looking for unusual words or names or wants everything known about something, then the spider-based engines cannot be beat. But most of the time, queries fall somewhere in between, and the Search Broker can help save time and focus the results. In a sense, it presents a very large live encyclopedia.

The first version of the Search Broker has been operational since October 1996. It was opened for public use on the web in July 1997. It can be found at `http://sb.cs.arizona.edu/sb/index.html`.

3 The Universal Search Interface

The Universal Search Interface (USI) is work in progress, which extends the Search Broker work in two directions. First, it is designed as a client rather than a server tool, allowing users to pick the search facilities they want to use and to customize them. We believe this to be a crucial usability feature: in the course of our daily use of the web, we found ourselves creating personal Search Broker databases that contained the sites of special interest to us, with our own aliases to identify them. Second, it provides tools to connect several search engines together, either in parallel or sequentially, to pipe results between searches, and to link web and local searches. The goal is to allow users to combine all the search facilities available to them —whether they search their own machine, their local network, or the Internet—in a powerful and customizable way via one common interface. The design will allow even inexperienced users to construct "search scenarios"—customized combinations of several search facilities—based on their preferences and needs. Scenarios may also be built by professionals and distributed like any other software. We will provide ways to "publish" search scenarios.

3.1 Examples of search Scenarios

Search for experts: Suppose that you are an editor looking for a referee to a paper about XYZ. You may first search for XYZ in relevant databases (e.g., The Collection of Computer Science Bibliographies at `http://liinwww.ira.uka.de/bibliography/index.html`, which uses our Glimpse, or the DBLP bibliography at `http://sunsite.ust.hk/dblp/db/index.html`) to find who has done work in that area. You then collect all authors' names and search for each one of them in either a special local file (e.g., of previous referees), or through all your files (e.g., using Glimpse). This is an example of piping results of one query into another after appropriate filtering.

Search for citations: Suppose that you want to find articles on the web that cite a given article. You first search one of the global web search engines, possibly asking for more than the usual number of first hits (e.g., 100 instead of 10) by combining together several requests. You then filter the results by extracting the URL from each hit. You fetch each page, and run agrep to find whether the citation is really there and extract the title and URL of that page. This is an example of several pipes involving web search and local search.

These examples involve common tasks that people currently have to do by hand with great effort. A USI will allow users to define these search tasks and then automate them.

3.2 Overall Design

The key to a successful implementation of a USI is the ability to handle three distinct issues:

Generality: A USI must be able to accommodate most search facilities, without changing them or even having access to their source code. The only assumption is that the user has permission and ability to perform the search through some mechanism.

Customizability: A USI must allow users to set their own preferences and customize their own search processes. Such customization must be easy to accomplish; in particular, it should not require any programming or any special knowledge of the search facilities (besides how to use them).

Ease of integration: Adding or extending search facilities should be relatively easy.

As is often the case, it is not likely that all three issues can be solved together perfectly, so making the right tradeoffs is crucial. Our approach to the design, to paraphrase an old saying attributed to Einstein, is to "make it as simple as we can, but not simpler than that." We will provide expansion room for generality and complexity, but the starting point will be as simple as possible. A good test of the simplicity of a USI is that it can be used to maintain hot lists (or bookmarks or favorite lists) as conveniently as they are maintained by the current browsers, but with more features. Next, users will learn to use a USI to build their own Search Broker (by extending items of a hot list to be "active" and trigger a search), and then they will be to able to construct complex scenarios. By concentrating on the most common type of searching interfaces, we hope to make the USI accessible to most users. Our challenge, in a sense, is to resist complexity by emphasizing the special flavor of search interfaces.

The USI is an umbrella encompassing several concepts and tools. One of the most important concepts is that of a "Search Object." Each search engine will have a corresponding search object which will encompass its interface, options, and formatting of results. Search objects will have associated input and output schemas, which will usually be very simple (e.g., a string in and a string out). There will also be schema-converting objects which extract, filter, and reformat the results of intermediate searches. Users can collect search objects, customize them, combine several of them into complex search objects, and build their own "super" interface that will give them easy access to all the scenarios they use in the way they want. We will build tools to allow users to construct search objects easily from given interfaces. In particular, for web-based CGI searches, users will be able to input the URL and fill in default options and/or fields, and the corresponding search object will be built for them (the Search Broker already provides a similar facility for its maintainers).

Selecting search objects and combining them to form scenarios will be done through a GUI which will be the main customization facility for users of a USI. Users will be able easily to pick a scenario from a USI and activate or modify it. The GUI will also provide type checking to guarantee consistent schemas, search capabilities on its own content, and tools for easy modification, import/export, and organization.

3.3 Putting Search Objects Together

The ability to compose search objects into scenarios is one of the most important features of a USI. In many cases, a user may wish to query multiple servers and combine the results, or to use the results of one query (e.g., to a web indexer) as the input to another (e.g., automatically retrieve the pages suggested by the indexer). A user may wish to examine intermediate results before passing them on to another stage, or may allow the system to run automatically. This kind of interaction extends the usual web browsing to include personal web "actions" triggered by the users. By treating stages consistently as implementations of a search object, we can support both arbitrary result reformatting, and complex searches where one output is the input of the next stage. Consistent use of the interface allows complex scenarios of search objects to be encapsulated and treated as search objects in their own right, providing a functionality similar to libraries of subroutines in programming languages.

To facilitate the combination of search objects, we use "translator objects" and "filtering objects". The distinction between search, translator, and filter objects is only for descriptive purposes—they are all objects with specific input and output schemas. A translator object translates between different schemas. For example, a translator may take the results of a particular search engine in HTML and extracts from it a set of fields (e.g., URLs,

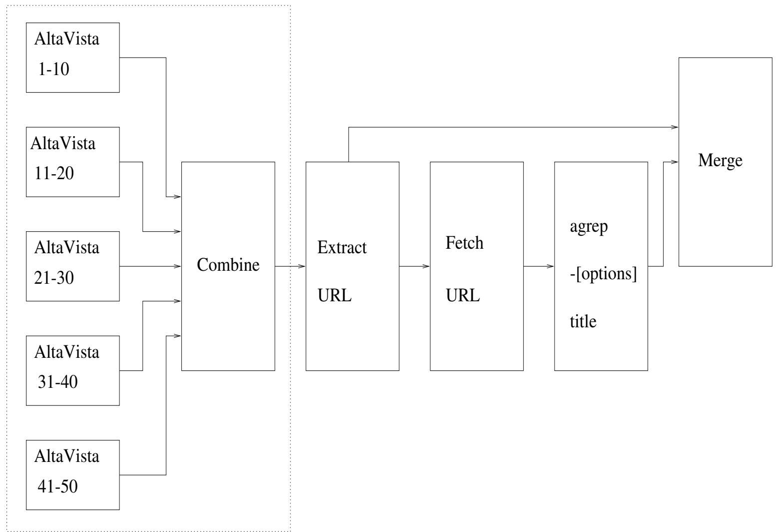

Figure 1: Putting together search objects to find citations

titles, abstracts, dates, scores). We expect this type of translator to be the most common one, and we will implement such translators for all popular search engines. We will also provide tools to allow users to build such translators for other interfaces. Translating from one complex database schema to another is very difficult in general, but we expect most of our search applications to require only simple translations, which is why this approach is feasible.

A filter object extracts some specific information. For example, a URL filter takes as input an HTML page and outputs the list of all URLs contained in that page. Other filters take a list of URLs and remove from it all those that the user has seen before (e.g., by consulting the history.db file), or take only those that appear in the user's bookmark list, or merge two URL lists together. The example of finding citations discussed earlier is shown as a combination of different objects in Figure 1. The area inside the dashed square is an example of part of a scenario that will most likely be encapsulated into a stand-alone search object.

The forms of customization we are discussing are instances of a black-box design [6], where the user is familiar with the interface to and capabilities of each USI object, but is unaware of and unconcerned with the internal implementation by which the object works. This is appropriate to meet the goals of a system used by people who are unfamiliar with programming, but who still want to customize searches to their own areas of interest. The configuration mechanism described above allows for specification of USI object options and supports complex scenarios by linking existing objects that perform searches and filter results. However, it does not provide for the ability to change the implementation of the interface methods. Nor does it allow users to define a search object that is not related to an existing one. Building from scratch a search object for Glimpse, for example, can be done only by a programmer. Selecting the right options can be done by anyone. This is a compromise we believe we must make.

4 Related Work

This work touches on many areas in computer science, including databases, user interfaces, networks, and algorithm design. For lack of space, we'll mention only a few related systems.

The seed of the Search Broker grew out of the Harvest project [2], where we attempted something similar, but ended up concentrating on the actual collection of data rather than the selection of servers. The closest existing search facilities to the Search Broker are the lists of search engines, such as The Internet Sleuth (http://www.isleuth.com/) and C/Net Search.com (http://www.search.com/). We believe that our approach is an improvement, and has the potential, especially with personal customization, to be a significant step forward. The USI can be thought of as an example of a *mediator*, a concept introduced by Wiederhold [13], and our search objects are similar to database *wrappers*. The Garlic system from IBM [4] and the use of mediators in TSIMMIS [5] are good examples. The main difference is in our emphasis on simple searches rather than on general database queries. As a result, our system is much simpler, and we can hope to allow non-experts to build specific scenarios, and easily customize them. Levy et al [8] developed tools (e.g., the Information Manifold) to allow complex queries across different databases. Their most pressing problem is how to negotiate with complex database schemas, a problem we don't have (yet).

Software agents, especially web agents, promise some of the features we want a USI to have, and address some of the same issues. Their emphasis, however, is on learning, and search is just one task in the context of many other tasks users perform. Our goal is to build the right infrastructure specifically for search. The two agent projects that are most relevant to this work are WBI from IBM Almaden [3] and LiveAgent [7] from AgentSoft Inc. Both work through HTTP proxies and serve as intermediaries between users and the web. WIDL from WebMethods [1], although not directly a system for end users, offers some of the scripting mechanisms we provide for the USI, and can be used by specialists to build similar search objects for scenarios of interest to businesses and organizations. Another very successful related work in the agents area is the "meta-search" [12] approach, which queries several search engines and combines the results. We believe that a USI will be instrumental in evaluating new meta-search techniques. We expect that a USI will be useful for agents, and hope that some of the agent software will be useful for us.

References

[1] C. A. Allen, "Automating the Web with WIDL," in "XML: Principles, Tools, and Techniques", O'Reilly, 1997.

[2] Bowman C. M., P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz, "The Harvest Information Discovery and Access System," *Computer Networks and ISDN Systems* **28** (1995) pp. 119-125.

[3] Barrett R., P. P. Maglio, and D. C. Kellem, "How to Personalize the Web," *Proceedings of the 1997 SIGCHI Conference on Human Factors in Computer Systems*, Atlanta, Georgia (March 1997).

[4] Carey M.J. et al, "Towards Heterogeneous Multimedia Information Systems: The Garlic Approach," *Fifth International Workshop on Research Issues in Data Engineering* Taipei, Taiwan (March 1995), pp. 124-131.

[5] Garcia-Molina H., Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom, "The TSIMMIS approach to mediation: Data models and Languages," *Journal of Intelligent Information Systems,* **8**(2) (1997), pp. 117-132.

[6] Johnson R.E., and B. Foote, "Designing Reusable Classes," *The Journal of Object-Oriented Programming* **1**(2) (June/July 1988), pp. 22-35.

[7] Krulwich B., "Automating the Internet – Agents as User Surrogates," *IEEE Internet Computing*, **1** (4) (July 1997).

[8] Alon Y. Levy, Anand Rajaraman and Joann J. Ordille, "Querying Heterogeneous Information Sources Using Source Descriptions," *Proceedings of the 22nd International Conference on Very Large Databases, VLDB-96,* Bombay, India, September, 1996.

[9] Manber U., and P. Bigot, "The Search Broker," *First Usenix Symp. on Internet Technologies and Systems,* Monterey, CA (December 1997), pp. 231-240. http://sb.cs.arizona.edu/sb

[10] Manber U., M. Smith, and B. Gopal, "WebGlimpse – Combining Browsing and Searching," *Usenix 1997 Annual Technical Conference,* Anaheim, CA (January 1997), pp. 195-206.

[11] Manber U., and S. Wu, "GLIMPSE: A Tool to Search Through Entire File Systems," *Usenix Winter 1994 Technical Conference,* (best paper award) San Francisco (January 1994), pp. 23-32. (ftp://ftp.cs.arizona.edu/reports/1993/TR93-34.ps.Z)

[12] E. Selberg, and O. Etzioni, "Multi-Service Search and Comparison Using the MetaCrawler," *Proceedings of the 4th International World Wide Web Conference* Boston, MA (December 1995).

[13] G. Wiederhold, "Mediators in the Architecture of Future Information Systems," *IEEE Computer*, **25** (March 1992), pp. 38-49.

Mediating and Metasearching on the Internet

Luis Gravano
Computer Science Department
Columbia University
www.cs.columbia.edu/ gravano

Yannis Papakonstantinou
Computer Science and Engineering Department
University of California, San Diego
www.cs.ucsd.edu/ yannis

1 Introduction

The Internet emerges as the largest database. Increasingly, users want to issue complex queries across Internet sources to obtain the data they require. However, finding relevant information sources and querying them manually is problematic: there are numerous sources, and they vary in the type of information objects they contain and in the interface they present to their users. Some sources contain text documents and support simple query models where a query is just a list of keywords. Other sources contain more structured data and provide query interfaces in the style of relational query languages. Furthermore, users have to manually fuse the query results by merging information, removing redundancies, ranking the answer objects in the appropriate order, and so on.

Since it is tedious to contact several heterogeneous sources, users can benefit from *metasearchers* and *mediators*, which are services that provide users with a virtual integrated view of the heterogeneous sources. Users access the view using a unified query interface that offers *location*, *model*, and *interface transparency*, i.e., users have the illusion of a single database and do not have to be aware of the location and interface of the sources. Although users and applications might access data directly through wrappers, mediators and metasearchers offer an integrated view of the world, where information related to the same entity has been fused together, redundancies have been eliminated, and inconsistencies have been removed.

The architecture of metasearchers and mediators are virtually identical (Figure 1). Wrappers export a common data model view of each source's data. Wrappers also provide a common query interface. After receiving a query, a wrapper translates it into a source-specific query or command, hence giving interface transparency to the user. Then, the wrapper translates the query results from the underlying source into the common data model or format.

To evaluate a user query over multiple heterogeneous databases, both metasearchers and mediators will typically perform three main tasks:[1]

- **Database Selection:** Choose the databases that have data relevant to the user query.
- **Query Translation:** Find the query fragment to be evaluated at each of the databases chosen in the previous step, translate these fragments so that they can be executed at their corresponding databases, and retrieve the query results from the databases.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] Note that this query processing strategy assumes that the integrated view is not materialized. An alternative would be to materialize the integrated view and evaluate queries on it directly. The query processing simplicity of the materialized view approach, together with the steep decline in disk storage prices has made it the predominant approach followed by data warehouses that integrate and consolidate corporate information.

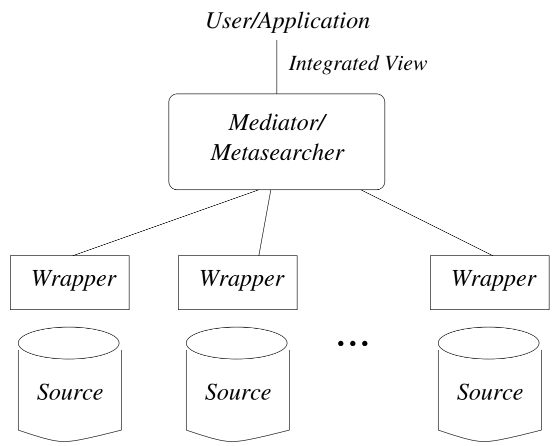

Figure 1: Both metasearchers and mediators use wrappers around the underlying databases. These wrappers provide a relatively uniform view of these databases.

- **Result Merging**: Combine the query results from the above databases into the final query answer.

In spite of their shared goals and architecture, the focus of the research on metasearchers has been quite different from that of the research on mediators. Two key issues explain this difference:

- **View Complexity:** Metasearchers typically operate on top of document databases, and the view that metasearchers export to users is generally some kind of union of the underlying databases. On the other hand, mediators usually integrate multiple relations or objects with complementary information. Thus, fusion of objects from several databases is not uncommon when defining mediator views. The higher complexity of the mediator views requires powerful view definition languages, together with powerful languages to query the integrated view.

- **Query Matches and Result Completeness:** The interaction of a user with a mediator is very similar to the interaction of a user with a relational database system; the user sends a query and the mediator typically returns the *complete answer* to the query. In effect, mediators generally operate over databases where query results are well defined sets of objects, as in the relational model. Metasearchers, however, usually deal with collections of unstructured text documents that return document ranks as the answer to a query, where the ranks are computed using undisclosed algorithms. The matches between queries and documents are "fuzzy." For example, *vector-space* databases compute how "similar" a document and a query are, where this similarity is a number between 0 and 1 [23]. Furthermore, these sources might return *only the best matches* for the query. Hence, metasearchers have to handle query results that have been computed using unknown matching algorithms. Also, metasearchers are aware that *partial answers* to queries are usually acceptable on the Internet, thus abandoning the goal of producing complete answers. This decision has strong implications on the query processing strategies that metasearchers might use.

Next, we describe how metasearchers (Section 2) and mediators (Section 3) address the three tasks that we discussed above to provide a unified view of the underlying heterogeneous databases.

2 Metasearchers

Web indexes like AltaVista (`http://www.altavista.digital.com`) are centralized full-text indexes of HTML documents. Unfortunately, the current web indexes do not manage to index every HTML document there is. Hence, if a user's web index of choice does not return satisfactory results for a query, the user might be forced to contact other indexes until the right documents are found. To complicate matters further, the contents of many text sources are hidden behind search interfaces (e.g., Knight-Ridder's Dialog information service, or the NC-STRL sources [2]). Web crawlers will typically not index the contents of such sources, since these sources' documents are only available in response to queries, not by following hypertext links. Hence, to access the contents of these *search-only sources*, users typically have to contact the sources themselves one by one. In either case, users need to query several autonomous, potentially heterogeneous sources.

Building metasearchers over Internet document sources is difficult because, in general, sources are too numerous. Therefore, finding the best sources for a query is a challenging task. Also, even if we know the document ranking algorithms that sources use, extracting the best objects for a query according to the metasearcher might be an expensive operation, since the sources' ranking algorithms might differ radically from that of the metasearcher's. Building metasearchers is also difficult because different sources are largely incompatible. In effect, the interfaces and query models of these sources vary from source to source. Even individual organizations use *search engines* from different vendors to index their internal document collections. In general, text search engines use different query languages, rank documents in the query results using secret algorithms, and do not export information about the sources in a standard form.

Several metasearchers already exist on the Internet for querying multiple web indexes. However, not all of them completely support the three major metasearch tasks described above. Examples include MetaCrawler [24] (`http://www.metacrawler.com`), SavvySearch (`http://guaraldi.cs.colostate.edu:2000/`}, and ProFusion [7].

STARTS, the *Stanford Protocol Proposal for Internet Retrieval and Search* [9], is an emerging protocol whose goal is to facilitate the three metasearching tasks above. *STARTS* has been developed in a unique way. It is not a standard, but a group effort involving 11 companies and organizations, coordinated by the Digital Library project at Stanford University.

Next, we discuss the three metasearching tasks in more detail, together with some related work. In particular, we summarize the main *STARTS* components and how they facilitate these tasks. For a detailed description of *STARTS*, please refer to [9].

2.1 Database Selection

A metasearcher might have thousands of sources available for querying. Some of these sources might charge for their use. Some of the sources might have long response times. Therefore, it becomes crucial that the metasearcher only contact sources that might contain useful documents for a given query, for which the metasearcher needs information about each source's contents.

To characterize the sources, we could manually write descriptions of their contents. For example, the Information Manifold system [15, 17] relies on declarative descriptions of the sources' contents and capabilities, written in a version of description logic. These descriptions are useful to prune the search space for evaluating user queries efficiently.

Alternatively, metasearchers might rely on automatically extracted summaries of the sources' contents. The *GlOSS* system [12, 10] uses source summaries that include the document frequency for each word in the source's vocabulary. [1] has applied inference networks (from information retrieval) to the database selection problem. Their approach summarizes databases using the same type of information that *GlOSS* keeps, together with the "inverse collection frequency" of the different terms. An inference network then uses this information to rank the databases for a given query.

To extract content summaries automatically from the sources, metasearchers require cooperation from the sources. If a source exports all of its contents (e.g., many web sites), then it is not as critical to have it describe

[2]The NCSTRL sources constitute an emerging library of computer science technical reports (`http://www.ncstrl.org`).

its collection to the metasearchers. After all, the metasearchers can just grab all of the sources' contents and summarize them any way they want. This is what the crawlers of web indexes like AltaVista do. In practice, some sources freely deliver their entire document collection, but others do not. Often, those sources that have for-pay information are of the second type. Morever, for performance reasons, it may still be useful to require that sources export a more succinct description of themselves. If a source "hides" its information (e.g., through a search interface), then it is even more important that the source can describe its contents. Otherwise, if a source does not export any kind of content summary, it becomes hard for a metasearcher to assess automatically what kind of information the source covers.

The *STARTS* protocol specifies that sources should export partial data about its contents [9]. This data is automatically generated, is orders of magnitude smaller than the original contents, and has proven helpful in distinguishing the more useful from the less useful sources for a given query [12, 10]. The *STARTS* summary for a source includes a list of all the words that appear at documents in the source, possibly with various tags, like the position in the documents where each word occurs, and some associated statistics that are easily computed from the source's index structures. For example, a source S might report that the English word *algorithm* appears in the title of 53 documents in S, while the Spanish word *datos* appears in the title of 12 documents. The summary might also tell us that there are 892 documents in the source. A metasearcher can use this information to decide whether a given query is likely to have good matches in source S.

2.2 Query Translation

A metasearcher submits queries over multiple sources. But the interfaces and capabilities of these sources may vary dramatically. Even the basic query model that the sources support may vary. Some search engines (e.g., Glimpse) only support the *Boolean retrieval model* [23]. In this model, a query is a condition that documents either do or do not satisfy. The query result is then a *set* of documents. For example, a query *distributed and systems* returns all documents that contain both the words *distributed* and *systems* in them.

Alternatively, most commercial search engines also support a variation of the *vector-space retrieval model* [23]. In this model, a query is a list of terms, and documents are assigned a score according to how *similar* they are to the query. The query result is then a *rank* of documents. For example, a query *"distributed systems"* returns a rank of documents that is typically based on the number of occurrences of the words *distributed* and *systems* in them. [3] A document in the query result might contain the word *distributed* but not the word *systems*, for example, or vice versa, unlike in the Boolean model above.

Even if two sources support a Boolean retrieval model, their query syntax often differ. More serious problems appear if different attributes are available for searching at different sources. For example, a source might support queries like *abstract "databases"* that ask for documents that have the word *databases* in their abstract, whereas some other sources might not support the *abstract* attribute for querying.

Another complication results from different stemming algorithms or stop-word lists being implicit in the query model of each source. (Stemming is used to make a query on *systems* also retrieve documents on *system*, for example. Stop words are used to not process very frequent words like *the* in the queries.) If a user wants documents about the rock group *The Who*, knowing about the stop-word behavior of the sources would allow a metasearcher to know whether it is possible to disallow the elimination of stop words from queries at each source.

As a result of all this heterogeneity, a metasearcher would have to translate the original query to adjust it to each source's syntax. To do this translation, the metasearcher needs to know the characteristics of each source. The work in [3, 4] illustrates the complexities involved in query translation. Querying multiple sources is much easier if the sources share a common query language.

The *STARTS* protocol specifies a flexible query language for sources. Even if support for most of this language is optional, query translation is much simpler if sources reveal what portions of the language they support. Thus,

[3] These ranks also typically depend on other factors, like the number of documents in the source that contain the query words.

STARTS asks that sources export detailed information on their searching capabilities. This information includes, for example, what attributes are supported for searching at each source (e.g., *author*, *title*, *body of text*).

2.3 Result Merging

A source that supports the vector-space retrieval model ranks its documents according to how "similar" the documents are to a given query. In practice, there are many ways to compute these similarities. To make matters more complicated, the ranking algorithms are usually proprietary to the search engine vendors, and their details are not publicly available.

Merging query results from sources that use different and unknown ranking algorithms is hard. For example, source S_1 might report that document d_1 has a *score* of 0.3 for some query, while source S_2 might report that document d_2 has a score of 1,000 for the same query. If we want to merge the results from S_1 and S_2 into a single document rank, should we rank d_1 higher than d_2, or vice versa? (Some search engines are designed so that the top document for a query always has a score of, say, 1,000.)

It is even hard to merge query results from sources that use the same ranking algorithm, even if we know this algorithm. The reason is that the algorithm might rank documents differently based on the collection where the document appears. For example, if a source S_1 specializes in computer science, the word *databases* might appear in many of its documents. Then, this word will tend to have a low associated weight in S_1 (e.g., if S_1 uses the *tf·idf* formula for computing weights [23]). The word *databases*, on the other hand, might have a high associated weight in a source S_2 that is totally unrelated to computer science and contains very few documents with that word. Consequently, S_1 might assign its documents a low score for a query containing the word *databases*, while S_2 assigns a few documents a high score for that query. Therefore, it is possible for two similar documents d_1 and d_2 to receive very different scores for a given query, if d_1 appears in S_1 and d_2 appears in S_2.

The problem of merging document ranks from multiple sources has been studied in the information retrieval field, where it is often referred to as the *collection fusion* problem. Given a query, the goal is to extract as many of the *relevant* documents as possible from the underlying document collections, where relevance is a subjective notion. Key decisions include how far "down" each document rank to explore, and how to translate the scores computed by the sources into the metasearcher's scores. An approach to address these problems is to learn from the results of training queries. Given a new query, the closest training queries are used to determine how many documents to extract from each available collection, and how to interleave them into a single document rank [29, 30]. Another approach is to calibrate the document scores from each collection using statistics about the word distribution in the collections [1].

The *STARTS* protocol asks sources to report term statistics in their query results. For example, the entry for a document d in the result for a query containing the word *database* might report the number of words in d, and that the query word *database* occurs 15 times in d, among other statistics on the document and its source. This way, metasearchers can try and merge query results in meaningful ways without having to retrieve the entire documents.

3 Mediators

Mediator systems [31] provide users with an integrated view of multiple heterogeneous information sources. These sources are not necessarily structured databases like relational and object oriented databases. In particular, the sources might contain semistructured data such as HTML (and very soon XML) documents, chemical abstracts, genome data, biology "metadata" (e.g., annotations of raw data), and bibliographic entries. Furthermore, the sources provide different and typically limited query interfaces to the data. Next, we describe how mediators handle the database selection and query translation tasks. We do not address the result merging task here, which is much simpler for mediators than it is for metasearchers. (See Section 2.3.)

3.1 Database Selection

When a mediator receives a query it first performs a *query decomposition* step where it computes the sources that are relevant to the query and decides what data is needed from each source [15, 13, 19, 18]. Mediators always use human-provided descriptions of the relationship between the integrated view and the sources in order to decompose queries. Assuming a significant amount of abstraction, we categorize the view/source relationship specifications in the following two broad classes and we provide a high level comparison of the virtues and disadvantages of the two approaches. A detailed technical comparison of the two approaches can be found in [26]. In practice, a hybrid of the two approaches is both desirable and possible.

The *view definition approach*, which is usually favored by the database community, describes the integrated view collections as views of the underlying source collections. For example, we may join a `salary` relation from the `payroll` source with an `employer` relation from the `human resources` source to produce an integrated view `complete_emp_info` that a mediator will export. Then, given a query, the mediator will expand the references to the integrated view with its definition, hence producing a rewritten query that has references to source collections only. Notice that query processing is similar to conventional database query processing and hence it can be based on existing query processors.

The view definition approach is not particular to the relational model. Indeed, recent mediator projects have preferred object oriented or semistructured models [6] for this purpose and they have also tuned the view definition language to the particular needs of integration.

The main advantage of the view definition approach is that source information can be integrated and transformed in complex ways. [4] On the other hand, this approach lacks modularity when it comes to adding new sources. For example, consider a view that is the union of collections of car advertisements. We will need to update the view specification every time a new car advertisement source becomes available. Furthermore, this approach misses a feature that is very desirable for query optimization: there is no direct description of the contribution of a source to the integrated view. For example, there is no way to express in the view definitions that one source covers only Honda's, while another source covers only BMW's. Hence, a mediator will not be able to direct a query on Honda's to the first source only. The source definition approach addresses this problem.

The *source definition approach*, which originated in the artificial intelligence and description logic communities, assumes the existence of global predicates and collections – say, a collection `car_ads` – and then defines the source contents with respect to the global predicates. For example, the Honda source above is defined to contain `car_ads` tuples with `make = Honda`. However, this statement should not be taken as a view definition of the source with respect to the global integrated view; the definition does not imply that the Honda source has all Honda `car_ads` tuples that appear in the integrated view. It only specifies that cars found in this source have to be Honda's.

3.2 Query Translation

Once the mediator computes what data is needed from each source, an optimizer develops an efficient plan that specifies what queries must be sent to the sources, and how the results will be combined. Note that the mediator has to retrieve the data required from a wrapper using only queries that the wrapper can translate into source specific queries or commands. For example, a mediator should not send a join query to a document retrieval source that can do selections only. Instead, it must decompose the join query into simple selection queries that can be handled by the source. At the same time, the capabilities of the most sophisticated sources must be exploited. This precludes the use of a simplistic "lowest common denominator" approach. For example, assuming that only one-condition selection queries are supported by the sources will also lead to inefficient plans because the mediator will not exploit the query processing abilities of sources that can process joins and multiple conditions. Indeed, in most cases, it is beneficial to produce *algebraically optimal* plans, i.e., plans that push as much work

[4] In theory, the only limit is the computational complexity of the view definition language.

as possible down to the sources. Focusing on select-project-join queries, a plan P is algebraically optimal [21] if there is no other plan P' such that for every query w sent to a wrapper by P there is a corresponding query w' of P' such that the sets of relations and conditions of w' are a superset of the corresponding sets of w and the set of exported attributes of w is a superset of the set of exported attributes of w'.

Several projects [22, 16, 25, 21, 20, 27] propose and discuss query processors that, given some description of the capabilities of the participating sources, adapt to the different, limited capabilities of the sources. An elegant approach to describing capabilities was introduced in [22] where the supported query interfaces were described by a finite number of parameterized views. For example, the following describes that the source `bib` supports substring conditions on the `title` field of the collection `reports`.

```
SELECT *
FROM bib.reports
WHERE reports.title LIKE $X
```

If we use these descriptions, finding a plan that consists of supported queries only is similar to finding a plan to answer queries using views. [20, 21, 27] provide languages for the specification of an infinite number of parameterized views and show that it is still possible to find all the potentially optimal plans. Finally note that other projects [14] tackle the rewriting issue as one describing acceptable plans that can be passed to the wrappers. However, the central ideas remain unchanged.

As we described in Section 2.2, the capabilities-based rewriting problem has also been addressed for text document collections [3, 4]. The system described in [3, 4] tends to produce efficient plans because it knows the semantic relationship between its own relations and access methods, and the source relations and access methods. For example, assume that the user query requests papers by *Knuth* where the title contains the word *complexity*. A source may not support a `contains word` predicate while, instead, it supports a `substring` predicate. By making the mediator aware that any document that satisfies the former predicate will also satisfy the latter one, we can rewrite the user query into one that uses `substring` and does the additional filtering at the mediator. In effect, such a mediator performs a form of semantic optimization [2].

4 Conclusion

Metasearchers and mediators provide uniform views over large numbers of heterogeneous databases. The problems that research in both areas has addressed, though, tend to differ significantly. However, some recent work has started to bridge the gap between these two areas:

- **Semistructured Data:** Metasearchers initially focused on text sources, while mediators were used for querying structured databases. However, both areas are converging on the study of sources with semistructured data. Such data has more structure than free text, but this structure is not as rigid, regular, or complete as that of relational schemas. Bibliographic entries, chemical abstracts, genome data, and a large number of HTML pages are typical examples of semistructured data. The emerging XML standard highlights the importance of an information exchange model for semistructured data.

  Database and mediator research has modeled semistructured data as graphs where the nodes carry semantic *labels* (i.e., the labels are essentially the metadata). Appropriate query and view definition languages have been designed for querying and transforming the graph data. However, the proposed query languages have logical underpinnings, and do not capture relevance ranking. Hence, they are still not well suited to many Internet querying scenarios where the query language is not logic-based.

- **Ranked Query Results:** Internet sources tend to overlap in arbitrary ways. Furthermore, users are often not interested in receiving "complete" answers to their queries. Instead, users sometimes prefer to receive the "best matches" for their queries. These characteristics of the data sources and the user expectations present both problems (we should avoid retrieving duplicate data), and opportunities (we can access fewer sources during query processing). To address these problems and opportunities several recent papers on mediators have proposed ways to define source overlap and optimization algorithms that produce efficient query execution plans [5, 28]. This work exploits the fact that users might be satisfied with efficiently computed partial answers to their queries, and produce incremental query plans for answering their queries, just like some metasearching systems already do for text documents [12, 10]. There has also been some initial work on producing ranked query results from sources of structured or semistructured data. (See [8] and DataSpot (`http://www.dataspot.com`).) The work in [11] addresses the problem of querying over multiple structured data sources that rank query results using different algorithms. One such source could be a real-estate agent that receives queries from users, and ranks the available houses according to how well they match the users' specification, for example.

Acknowledgments

We thank Roy Goldman and Vasilis Vassalos for their useful comments on the paper.

References

[1] J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of the Eighteenth ACM International Conference on Research and Development in Information Retrieval (SIGIR'95)*, July 1995.

[2] U. Chakravarthy, J. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Transactions on Database Systems*, pages 162–207, 1990.

[3] C.-C. K. Chang, H. García-Molina, and A. Paepcke. Boolean query mapping across heterogeneous information sources. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):515–521, Aug. 1996.

[4] C.-C. K. Chang, H. García-Molina, and A. Paepcke. Predicate rewriting for translating Boolean queries in a heterogeneous information system. Technical Report SIDL-WP-1996-0028, Stanford University, 1996. Accessible at `http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0028`.

[5] D. Florescu, D. Koller, and A. Levy. Using probabilistic information in data integration. In *Proceedings of the Twenty-third International Conference on Very Large Databases (VLDB'97)*, Aug. 1997.

[6] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: data models and languages. *Journal of Intelligent Information Systems*, 8:117–132, 1997.

[7] S. Gauch and G. Wang. Information fusion with ProFusion. In *Proceedings of the World Conference of the Web Society (WebNet'96)*, Oct. 1996.

[8] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity searching in databases. In *Proceedings of the Twenty-fourth International Conference on Very Large Databases (VLDB'98)*, Aug. 1998.

[9] L. Gravano, C.-C. K. Chang, H. García-Molina, and A. Paepcke. *STARTS*: Stanford proposal for Internet metasearching. In *Proceedings of the 1997 ACM International Conference on Management of Data (SIGMOD'97)*, May 1997.

[10] L. Gravano and H. García-Molina. Generalizing *GlOSS* for vector-space databases and broker hierarchies. In *Proceedings of the Twenty-first International Conference on Very Large Databases (VLDB'95)*, pages 78–89, Sept. 1995.

[11] L. Gravano and H. García-Molina. Merging ranks from heterogeneous Internet sources. In *Proceedings of the Twenty-third International Conference on Very Large Databases (VLDB'97)*, Aug. 1997.

[12] L. Gravano, H. García-Molina, and A. Tomasic. The effectiveness of *GlOSS* for the text-database discovery problem. In *Proceedings of the 1994 ACM International Conference on Management of Data (SIGMOD'94)*, May 1994.

[13] L. Haas, D. Kossman, E. Wimmers, and J. Yang. An optimizer for heterogeneous systems with non-standard data and search capabilities. *Special Issue on Query Processing for Non-Standard Data, IEEE Data Engineering Bulletin*, 19:37–43, Dec. 1996.

[14] L. Haas, D. Kossman, E. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proceedings of the Twenty-third International Conference on Very Large Databases (VLDB'97)*, Aug. 1997.

[15] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava. The Information Manifold. In *Proceedings of the AAAI Spring Symposium Series*, Mar. 1995.

[16] A. Levy, A. Rajaraman, and J. Ullman. Answering queries using limited external processors. In *Proceedings of the Fifteenth ACM Symposium on Principles of Database Systems (PODS'96)*, pages 227–237, June 1996.

[17] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases (VLDB'96)*, Sept. 1996.

[18] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of the Twenty-second International Conference on Very Large Databases (VLDB'96)*, Sept. 1996.

[19] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the 1996 ICDE Conference*, pages 132–41, 1996.

[20] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman. A query translation scheme for the rapid implementation of wrappers. In *Proceedings of the 1995 DOOD Conference*, pages 161–86, 1995.

[21] Y. Papakonstantinou, A. Gupta, and L. Haas. Capabilities-based query rewriting in mediator systems. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems (PDIS'96)*, June 1996.

[22] A. Rajaraman, Y. Sagiv, and J. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the Fourteenth ACM Symposium on Principles of Database Systems (PODS'95)*, pages 105–112, May 1995.

[23] G. Salton. *Automatic Text Processing: The transformation, analysis, and retrieval of information by computer*. Addison-Wesley, 1989.

[24] E. Selberg and O. Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the Fourth International WWW Conference*, Dec. 1995.

[25] A. Tomasic, L. Raschid, and P. Valduriez. Scaling heterogeneous databases and the design of DISCO. Technical report, INRIA, 1995.

[26] J. Ullman. Information integration using logical views. In *Proceedings of the 1997 ICDT Conference*, pages 19–40, 1997.

[27] V. Vassalos and Y. Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In *Proceedings of the Twenty-third International Conference on Very Large Databases (VLDB'97)*, pages 256–266, Aug. 1997.

[28] V. Vassalos and Y. Papakonstantinou. Using knowledge of redundancy for query optimization in mediators. Technical report, Computer Science Department, Stanford University, 1998.

[29] E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird. The collection fusion problem. In *Proceedings of the Third Text Retrieval Conference (TREC-3)*, Mar. 1995.

[30] E. M. Voorhees and R. M. Tong. Multiple search engines in database merging. In *Proceedings of the Second ACM International Conference on Digital Libraries (DL'97)*, July 1997.

[31] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25:38–49, 1992.

What can you do with a Web in your Pocket?

Sergey Brin [¶] Rajeev Motwani [‖] Lawrence Page [**] Terry Winograd [††]

Abstract

The amount of information available online has grown enormously over the past decade. Fortunately, computing power, disk capacity, and network bandwidth have also increased dramatically. It is currently possible for a university research project to store and process the entire World Wide Web. Since there is a limit on how much text humans can generate, it is plausible that within a few decades one will be able to store and process all the human-generated text on the Web in a shirt pocket.

The Web is a very rich and interesting data source. In this paper, we describe the Stanford WebBase, a local repository of a significant portion of the Web. Furthermore, we describe a number of recent experiments that leverage the size and the diversity of the WebBase. First, we have largely automated the process of extracting a sizable relation of books (title, author pairs) from hundreds of data sources spread across the World Wide Web using a technique we call Dual Iterative Pattern Relation Extraction. Second, we have developed a global ranking of Web pages called PageRank based on the link structure of the Web that has properties that are useful for search and navigation. Third, we have used PageRank to develop a novel search engine called Google, which also makes heavy use of anchor text. All of these experiments rely significantly on the size and diversity of the WebBase.

1 Introduction

The Web is a very diverse data source combining highly structured HTML, fully general natural language contained within the HTML, and embedded images. On top of this data is a reasonably well defined link structure, which is a directed graph over all the Web pages, with labels on the links (the text of the anchors).

We have found many applications for this data, some of which we will describe here.

- The extraction of structured data from many unstructured pieces spread throughout the Web

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[¶]Partially supported by the Community Management Staff's Massive Digital Data Systems Program, NSF grant IRI–96–31952, and grants of IBM and Hitachi Corp.

[‖]Supported by an NSF Young Investigator Award CCR–9357849, with matching funds from IBM, Mitsubishi, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

[**]Supported by the Stanford Integrated Digital Library Project (see below).

[††]Supported by the Stanford Integrated Digital Library Project. This project is supported by NSF Cooperative Agreement IRI-9411306. Funding for this cooperative agreement is also provided by DARPA and NASA, and by Interval Research, and the industrial partners of the Stanford Digital Libraries Project.

If such a process is sufficiently accurate, it can provide a comprehensive and queryable source of information. We present a technique and some initial results for automatically extracting relations from the Web in Section 3.

- Enhanced information retrieval.

  This area has already been explored by a number of researchers and search engine companies. However, there is plenty of room for improvement in information retrieval on the Web, and we present some novel techniques in Sections 4 and 5.

In the work presented in this paper we take advantage of one central idea: the Web provides its own metadata through its link structure, anchor text, and partially redundant content. This is because a substantial portion *of* the Web is *about* the Web. To take full advantage of this data would require human intelligence or more. However, simple techniques that focus on a small subset of the potentially useful data can succeed due to the scale of the web. A technique that might capture only one percent of the available information might still be very useful. Because size is so crucial to these techniques, and the amount of publicly available information is likely to grow rapidly, it is important to maintain a large Web repository like our Stanford WebBase.

2 The Stanford WebBase

The size of the World Wide Web is an elastic number. There are many automatically generated infinite Web spaces, there is a large amount of duplication, and the Web changes every day. Nonetheless, a recent estimate has put its size at about 200 million web pages in November 1997 [BB98].

In 1994, one of the first web search engines, the World Wide Web Worm [McB94], had an index of 110,000 web pages and web accessible documents. As of March, 1998, the largest search engines claim to index from 2 to 110 million web documents [Sul]. It is foreseeable that by the year 2000, a comprehensive index of the Web will contain over a billion documents.

The Stanford WebBase is designed to store a significant subset of the text content of the Web for research purposes. It currently holds roughly 25 million web pages and will soon expand to approximately double that size. The repository itself is roughly 150 GB of HTML (stored compressed in 50 GB). Additionally, we keep an inverted index of the text, which includes word position and font information, occupying an additional 50 GB. Finally, various metadata including URL's and link structure occupy another 10 GB .

By traditional standards, the repository contains a huge amount of information. However, since the acquisition and indexing can be almost fully automated, the repository can be collected, processed, and maintained by a university research project. As disk space, network bandwidth, and computing power continue to improve and fall in price, eventually a school child will be able to store a comparable collection on a portable computer.

3 Extraction of Relations

The World Wide Web provides a vast resource for information. At the same time it is extremely distributed. A particular type of information such as restaurant lists may be scattered across thousands of independent information sources in many different formats. If these chunks of information could be extracted from the World Wide Web and integrated into a structured form, they would form an unprecedented source of data, such as a combined international directory of people, the largest and most diverse databases of products, and the broadest bibliography of academic works.

There has been considerable work on integrating multiple information sources using specially coded wrappers or filters [Tsi, MOS97]. However, these can be time-consuming to create and maintain and are usually used for tens, not thousands of sources. In this section, we address the problem of extracting a relation from the thousands of sources that may hold pieces of the relation on the World Wide Web. Our goal is to discover information sources and to extract the relevant information from them either entirely automatically, or with very minimal human intervention.

3.1 Dual Iterative Pattern Relation Expansion (DIPRE)

Our test problem is to extract a relation of books – (author,title) pairs from the Web. Intuitively, our solution works as follows. We begin with a small seed set of (author, title) pairs (in tests we used a set of just five pairs). Then we find all occurrences of those books on the Web (an occurrence of a book is the appearance of the title and author in close proximity on the same Web page). From these occurrences we recognize patterns for the citations of books. Then we search the Web for these patterns and find new books. We can then take these books, find all their occurrences, and from those generate more patterns. We can use these new patterns to find more books, and so forth. Eventually, we will obtain a large list of books and patterns for finding them. We call this technique DIPRE - Dual Iterative Pattern Relation Expansion. In the process of applying DIPRE, it is much more important to be precise (not generate too much garbage) than it is to be complete, because the effect of adding false patterns is a proliferation of false entries.

Thus far we have been vague about the concepts of an occurrence and a pattern. They can be formalized in a number of ways. We chose a very simple approach to demonstrate the general technique:

An occurrence of an *(author,title)* pair is represented as a seven-tuple: *(author, title, order, url, prefix, middle, suffix)*. The *order* is a boolean value that determines whether the author appears before the title or the title appears before the author. The *url* is the URL of the document they occurred on. The *prefix* consists of the m characters (in tests m was 10) preceding the author (or title if the title was first). The *middle* is the text between the author and title, and the *suffix* consists of the m characters following the title (or author).

A pattern is a five-tuple: *(order, urlprefix, prefix, middle, suffix)*. The $order$ is boolean and the other attributes are strings. If *order* is true, an *(author,title)* pair matches the pattern if there is a document in the collection (the WWW) with a URL that matches *urlprefix*$*$ and which contains text that matches the regular expression: $*$*prefix, $<$author$>$, middle, $<$title$>$, suffix*$*$. If order is false, then the author and title are reversed.

Let R be the relation we are extracting, P be the set of patterns we are extracting, and let O be a set of occurrences. DIPRE works as follows:

1. $R \leftarrow$ Sample
   Initialize R with a small seed of the relation we are trying to find. In our tests it was just a list of five author,title pairs.

2. $O \leftarrow$ FindOccurrencesA(R)
   Find all occurrences of tuples of R on the Web.

3. $P \leftarrow$ GenPatterns(O)
   Generate patterns from the set of occurrences.

4. $O \leftarrow$ FindOccurrencesB(P)
   Find all occurrences of the patterns on the Web.

5. $R \leftarrow R \cup$ Tuples(O)
   Add the newly found author,title pairs to R.

6. If R is large enough, terminate. Else, go to step 2.

There are three important operations above: FindOccurrencesA, FindOccurrencesB, and GenPatterns (Tuples is just a trivial project). FindOccurences A and B find occurrences of books and patterns respectively in the Web repository. These are fairly similar operations but can be challenging to compute efficiently, since there are tens

Isaac Asimov	The Robots of Dawn
David Brin[1]	Startide Rising
James Gleick	Chaos: Making a New Science
Charles Dickens	Great Expectations
William Shakespeare	The Comedy of Errors

Figure 1: Initial sample of books.

<table>
<tr><th>URL Pattern</th><th>Text Pattern</th></tr>
<tr><td>www.sff.net/locus/c.*</td><td><LI><B>title</B> by author (</td></tr>
<tr><td>dns.city-net.com/ lmann/awards/hugos/1984.html</td><td><i>title</i> by author (</td></tr>
<tr><td>dolphin.upenn.edu/ dcummins/texts/sf-award.htm</td><td>author || title || (</td></tr>
</table>

Figure 2: Patterns found in first iteration.

of thousands of books (or more) and hundreds of patterns. For the purposes of our experiment, the equivalent of a *grep* over the WebBase was used. Much of the experiment was limited to just portions of the WebBase because this operation is so time-consuming.

The third operation, GenPatterns, is the trickiest. The key is to avoid generating overly general patterns. We avoid such patterns by requiring that any patterns generated provide sufficiently long urlprefixes, prefixes, middles, and suffixes. We refer to [Bri] for the details of this process.

3.2 The Experiment

We started the experiment with just 5 books (see Figure 1). These produced 199 occurrences and generated 3 patterns (see Figure 2). Interestingly, only the first two of the five books produced the patterns because they were both science fiction books. A run of these patterns over matching URL's produced 4047 unique (author,title) pairs. They were mostly science fiction but there were some exceptions.

A search through roughly 5 million web pages found 3972 occurrences of these books. This number was something of a disappointment since it was not as large a blowup as had happened in the first iteration. However, it would have taken at least a couple of days to run over the entire repository so we did not attempt to generate more.

These occurrences produced 105 patterns, 24 of which had url prefixes that were not complete URLs. A pass over roughly 2 million URLs produced 9369 unique (author, title) pairs. Unfortunately, there were some bogus books among these. In particular, 242 of them were legitimate titles but had an author of "Conclusion". We removed these from the list. This was the only manual intervention through the whole process. In future experiments, it would be interesting to see whether leaving these in would produce an unacceptable amount of junk.

For the final iteration, we chose to use the subset of the repository that contained the word "books." This subset consisted of roughly 156,000 documents. Scanning for the 9127 remaining books produced 9938 occurrences. These in turn generated 346 patterns. Scanning over the same set of documents produced 15257 unique books with very little bogus data.

The quality of the results was surprisingly good. A random sample of the proposed books revealed that roughly 95% were legitimate books. The remainder were mostly articles and other media. Of these, roughly 25% were not found on Amazon (which incidentally is not a part of the WebBase since they are not accessible to robots) or several other major sites but were legitimate books with copyrights and publishers. Some were out of print, some were published electronically, and still others were missing from these online book stores for no apparent reason. Therefore, the union of hundreds of small sources of book citations of the Web is very useful even in the presence of the huge catalogs available online. For more details about this experiment see [Bri].

4 PageRank

4.1 Link Structure of the Web

While estimates vary, the current graph of the crawlable Web has roughly 300 million nodes (pages) and 3 billion edges (links). Every page has some number of forward links (outedges) and backlinks (inedges) (see Figure 3). We can never know whether we have found all the backlinks of a particular page, but if we have downloaded it, we know all of its forward links at that time.

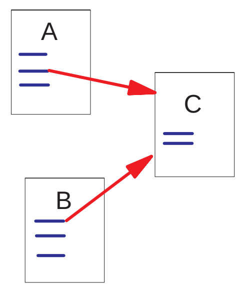

Figure 3: A and B are Backlinks of C

Web pages vary greatly in terms of the number of backlinks they have. For example, the Netscape home page has 62,804 backlinks in our current database while most pages have just a few backlinks. Generally, highly linked pages are more "important" than pages with few links. Simple citation counting has been used for many things including speculating on the future winners of the Nobel Prize [San95]. PageRank -[PBMW] provides a more sophisticated method for doing citation counting.

The reason that PageRank is interesting is that there are many cases where simple citation counting does not correspond to our common-sense notion of importance. For example, if a web page has a link from the Yahoo home page, it may be just one link but it is a very important one. This page should be ranked higher than other pages with more links but only from obscure places. PageRank is an attempt to see how good an approximation to "importance" can be obtained from just the link structure.

4.2 Propagation of Ranking Through Links

Based on the discussion above, we give the following intuitive description of PageRank: a page has high rank if the sum of the ranks of its backlinks is high. This covers both the case when a page has many backlinks and when a page has a few highly ranked backlinks.

4.3 Definition of PageRank

Let u be a web page. Then let F_u be the set of pages u points to and B_u be the set of pages that point to u. Let $N_u = |F_u|$ be the number of links from u and let c be a factor used for normalization (so that the total rank of all web pages is constant).

We begin by defining a simple ranking, R which is a slightly simplified version of PageRank:

$$R(u) = \frac{1}{c} \sum_{v \in B_u} \frac{R(v)}{N_v}$$

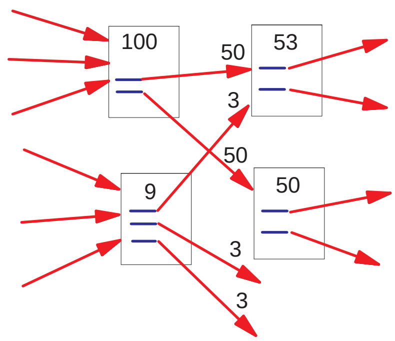

Figure 4: Simplified PageRank Calculation

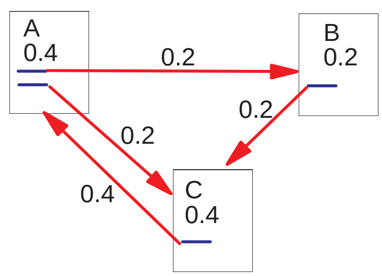

Figure 5: Steady State Solution to the PageRank Calculation

This equation formalizes the intuition in the previous section. Note that the rank of a page is divided among its forward links evenly to contribute to the ranks of the pages they point to. Note that $c < 1$ because there are a number of pages with no forward links and their weight is lost from the system. The equation is recursive but it may be computed by starting with any set of ranks and iterating the computation until it converges. Figure 4 demonstrates the propagation of rank from one pair of pages to another. Figure 5 shows a consistent steady state solution for a set of pages.

Stated another way, let A be a square matrix with the rows and columns corresponding to web pages. Let $A_{u,v} = 1/N_u$ if there is an edge from u to v and $A_{u,v} = 0$ if not. If we treat R as a vector over web pages, then we have $cR = AR$. So R is an eigenvector of A with eigenvalue c. In fact, we want the dominant eigenvector of A. It may be computed by repeatedly applying A to any nondegenerate initial rank vector.

There is a small problem with this simplified ranking function. Consider two web pages that point to each other but to no other page, and suppose there is some web page that points to one of them. This situation is shown in Figure 6. During iteration, this loop will accumulate rank but never distribute any rank (since there are no outedges). The loop forms a sort of trap which we call a rank sink.

To overcome this problem of rank sinks, we introduce a rank source:

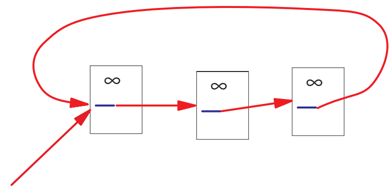

Figure 6: Loop that Acts as a Rank Sink

Definition 1: Let $E(u)$ be some vector over the Web pages that corresponds to a source of rank. Then, the PageRank of a set of Web pages is an assignment, R', to the Web pages which satisfies

$$R'(u) = \frac{1}{c}(\sum_{v \in B_u} \frac{R'(v)}{N_v} + E(u)) \tag{1}$$

such that c is maximized and $||R'||_1 = 1$ ($||R'||_1$ denotes the sum of the components of R').

$E(u)$ is some vector over the web pages that corresponds to a source of rank. Note that if E is all positive, c must be increased to balance the equation. Therefore, this technique corresponds to a decay factor. In matrix notation we have $cR' = AR' + E$. Since $||R'||_1 = 1$, we can rewrite this as $cR' = A + E \times \mathbf{1}R'$ where $\mathbf{1}$ is the vector consisting of all ones. So, R' is an eigenvector of $(A + E \times \mathbf{1})$.

4.4 Random Surfer Model

The definition of PageRank above has another intuitive basis in random walks on graphs. The simplified version corresponds to the standing probability distribution of a random walk on the graph of the Web. Intuitively, this can be thought of as modeling the behavior of a "random surfer". The "random surfer" simply keeps clicking on successive links at random. However, if a real Web surfer ever gets into a small loop of web pages, it is unlikely that the surfer will continue in the loop forever. Instead, the surfer will jump to some other page. The additional factor E can be viewed as a way of modeling this behavior: the surfer periodically "gets bored" and jumps to a random page chosen based on the distribution in E.

4.5 Personalization and Malicious Manipulation (Spam)

So far we have left E as a user defined parameter. In most tests we let E be uniform over all web pages with value α. However, E can be skewed to a particular user, say to weigh their bookmarks or homepage higher. We did some preliminary experiments with E set only to one particular Stanford professor's home page. This seemed to result in higher ranking for things relating loosely to that professor. Also, with this type of personalized E it is nearly impossible to mislead the ranking algorithm. There is no way to inflate the importance of a page, without convincing other pages to give you part of their importance. Even if I create a million web pages trying to mislead the system, these million pages will have the total importance only of the sum of the links into them. So if there is only one link from the outside web into my million pages, the million pages will have only the ranking from that one link to distribute amongst themselves. This type of resistance is very important in the commercial space, because there is a great deal of economic incentive to move pages to the top of popular search results.

4.6 Computing PageRank

The computation of PageRank is fairly straightforward if we ignore the issues of scale. Let S be almost any vector over Web pages (for example E). Then PageRank may be computed as follows:

$$\begin{array}{rcl} & R_0 & \leftarrow & S \\ \text{loop :} & & & \\ & R_{i+1} & \leftarrow & AR_i \\ & d & \leftarrow & ||R_i||_1 - ||R_{i+1}||_1 \\ & R_{i+1} & \leftarrow & R_{i+1} + dE \\ & \delta & \leftarrow & ||R_{i+1} - R_i||_1 \\ \text{while } \delta > \epsilon & & & \end{array}$$

Note that the d factor increases the rate of convergence and maintains $||R||_1$. An alternative normalization is to multiply R by the appropriate factor. The use of d may have a small impact on the influence of E.

We should also note that there are a large number of nodes that have no outgoing edges when we do not have a complete version of the web. Because of the presence of these nodes with unknown outedges, there is no completely satisfactory solution as to where to distribute their weight. If we do nothing, we have found that these nodes with no outgoing edges can cause the d factor to become larger than seems to work well. As a solution, we often remove nodes with no outgoing edges during the computation of PageRank, then add them back in after the weights have stabilized. This results in a slight boost to some nodes due to the change in normalization, but seems to be preferable to the alternatives.

5 The Google Search Engine

A major application of PageRank is searching. We have implemented two search engines which use PageRank. The first is a simple title-based search engine. The second is a full text search engine called Google [BPa] - [BP98]. Google utilizes a number of factors to rank search results, including standard IR measures, proximity, anchor text (text of links pointing to web pages), and PageRank. We encourage readers to try out Google at http://google.stanford.edu/ which allows searching the full text of 24 million web pages.

5.1 Use of PageRank

The benefits of PageRank are the greatest for underspecified queries. For example, on a conventional search engine a query for "Stanford University" may return any number of web pages which mention Stanford (such as publication lists. Using PageRank, the university home page is listed first.

An even more general query – "University" – is completely hopeless on conventional search engines, at best returning the home pages of random universities and at worst returning random pages at random universities. However, because of PageRank, the top ten results are the home pages of ten major universities – UIUC, Stanford, Michigan, and so forth.

5.2 Anchor Text

Let us consider the query "Stanford University" again. It so happens that there are nearly 6000 links to the Stanford home page, and for the overwhelming majority of them the text of the link reads "Stanford University". When there are several thousand links that match the query exactly and point to the same Web page (like the

Stanford example), they should be a pretty good indication to a search engine that that page may be a good result to return for that query.

Even in cases when there are just one or several anchors pointing to a page that match a query, these anchors are very useful. First, anchors are often better descriptions of Web pages than the pages themselves. They frequently state precisely what is significant about the Web page. Second, they are often written by people other than the author of the Web page, so they are more resistant to malicious tampering to move a page to the top of the search results for commercial gain. In fact, Google distinguishes between on-site, off-site, and off-domain anchors to improve the resistance to malicious tampering. Finally, anchors allow a search engine to return a Web page even without crawling it. So even though the WebBase contains roughly 25 million Web pages, Google can return any of roughly 60 million URL's in search results, including images and email addresses.

5.3 Proximity

Another important component of Google is heavy reliance on word proximity. That is, if two query words occur close together in a document, then the document is weighted much more heavily than if they appear far apart. Use of proximity makes searching considerably more computationally expensive. However, in practice our unoptimized system can answer the vast majority of queries in a matter of seconds.

6 Other Experiments

The WebBase has also been used for several other experiments, which we mention only briefly here. These range from duplicate detection to data mining to queryless search.

The Web can be a very useful testbed for experimenting with systems that may be applied more broadly. SCAM is a project at Stanford to detect duplicated or nearly duplicated documents. The WebBase turns out to be a very good test set for that purpose, since it is a large corpus with a large amount of duplication with many variations in how texts are duplicated. The results of duplicate analysis ([SGM]) show that roughly 22% of the documents in the WebBase are exact duplicates, and even more have approximate duplicates.

Another application for which the Web makes a good test data set is market basket mining. We consider the Web pages to be the baskets and the words that appear on them are the items. This data set is particularly challenging for traditional algorithms. There are over 10 million distinct words of which tens of thousands still occur in significant frequency. A number of the words occur very often, and there are many extremely strong correlations. A recent project on dynamic data mining considers a market basket algorithm when it is not possible to exhaustively explore all possible rules [BPb].

A research direction for searching is what we call "background surfing" or "queryless search". Our contributions in Google thus far have made underspecified queries such as one or two word queries work well. However, now we consider the case of zero word queries. Our goal is to build a system that listens to conversation going on around it and produce relevant Web pages. We feel that such a system can have a very substantial impact on the way people work, because it makes them aware of things they did not even know to look for. A current very early prototype scans through email and retrieves relevant Web pages. The overall goal is a difficult task because it must include and combine high quality continuous speech recognition over a large vocabulary with high quality search where it is not feasible to display 10 results.

6.1 Summarization

An issue we wish to explore is the summarization of a collection of documents. We are currently developing the following process. Suppose that we wish to summarize the contents of the pages returned for a query to a web search engine. We fetch the pages whose URLs form the response to the query from a local repository, identify a family of important keywords (via proximity to search terms or using traditional IR techniques such as inter-document frequency), identify phrases containing these keywords, and finally output a set of phrases that occur most frequently across the documents (under some notion of approximate phrase equality). Our thesis is that this set of phrases will read like a summary of the web documents.

A critical component of this approach is the identification of interesting phrases in collections of documents. It is important to avoid generating phrases that are merely common language constructs or are sporadic juxtapositions of terms. We do not restrict ourselves to grammatical or English phrases; indeed, we would like to generate more general kinds of phrases including dates, email addresses, phone numbers, names, etc. It is our expectation, supported by some preliminary experimental results, that the resulting sets of phrases provide a good sense of the contents and the topic of a document collection. We have developed an approach for quickly identifying interesting phrases. For instance, we define a two-word phrase as a pair of words that have a strong affinity for each other and appear together in a particular order far more often than could be explained by language constructs or pure chance. Our technique scans text in linear time and produces a ranked list of phrases without prior knowledge of the content of a document or its relationship with the document collection. While performing experiments in phrase detection, we noticed that when our technique is applied to a collection of documents returned by a search engine on a query, the detected phrases are very descriptive of the quality of the pages returned by the search engine.

Another way to use this summarization is to visualize clusters of documents. The clusters may be formed out of the responses to a query or we may just have a clustering of some part of the web. In particular, suppose we have a hierarchical clustering of a collection of web pages. We can provide the user with a summary of the document sub-collection associated with any point in the hierarchy as well as the differences in the set of characterizing phrases between two adjacent points in the hierarchy. This summarization will enable the user to visualize the clustered documents while navigating or browsing the hierarchy.

7 Conclusion

A repository of Web pages such as the WebBase is an excellent research tool, enabling experiments that would otherwise be impossible to perform efficiently. And, of course, it can be crucial to the development of better search engines.

An important lesson we have learned from these experiments is that *size does matter*. The extraction experiment would likely have failed if the WebBase had been one third of its current size.

Furthermore, the hardware cost of a large WebBase is quite reasonable and trends in disk capacity and computing power make it very likely that many more applications involving a local Web repository will become practical in the near future.

In analyzing the Web, we have found that it is important to look beyond just the text. The extraction experiment made heavy use of formatting and URL's. PageRank takes advantage of the link structure. Google makes use of anchor text and font information. Much of the information on the Web is not in the plain text and many applications can achieve great gains by leveraging it.

8 Acknowledgements

Scott Hassan and Alan Steremberg have been critical to the development of Google. Their talented contributions are irreplaceable, and the authors owe them much gratitude. Finally we would like to recognize the generous support of our equipment donors IBM, Intel, and Sun and our funders.

References

[BB98] Krishna Bharat and Andrei Broder. A technique for measuring the relative size and overlap of public web search engines. In *Proceedings of the Seventh International World Wide Web Conference*, 1998.

[BPa] Sergey Brin and Larry Page. Google search engine. `http://google.stanford.edu`.

[BPb] Sergey Brin and Lawrence Page. Dynamic data mining: Exploring large rule spaces by sampling. `http://www-db.stanford.edu/ sergey/ddm.ps`.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International World Wide Web Conference*, 1998.

[Bri] Sergey Brin. Extracting patterns and relations from the world wide web. In *Proceedings of the WebDB Workshop at EDBT 1998*. `http://www-db.stanford.edu/ sergey/extract.ps`.

[McB94] Oliver A. McBryan. GENVL and WWWW: Tools for Taming the Web. In O. Nierstarsz, editor, *Proceedings of the first International World Wide Web Conference*, page 15, CERN, Geneva, May 1994.

[MOS97] Workshop on management of semistructured data. `http://www.research.att.com/ suciu/workshop-papers.html`, May 1997.

[PBMW] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. `http://google.stanford.edu/google_papers.html`.

[San95] Neeraja Sankaran. Speculation in the biomedical community abounds over likely candidates for nobel. *The Scientist*, 9(19), Oct 1995.

[SGM] Narayanan Shivakumar and Hector Garcia-Molina. Finding near-replicas of documents on the web. In *Proceedings of the WebDB Workshop at EDBT 1998*.

[Sul] Danny Sullivan. Search engine watch. `http://www.searchenginewatch.com`.

[Tsi] Tsimmis home page. `http://www-db.stanford.edu/tsimmis/tsimmis.html`.

Virtual Database Technology, XML, and the Evolution of the Web

STS Prasad and Anand Rajaraman†

Junglee Corporation
1250 Oakmead Parkway, Suite 310
Sunnyvale, CA 94086-4027
http://www.junglee.com

Abstract

We describe Junglee's Virtual Database (VDB) technology, which makes the World Wide Web and other external data sources behave as an extension of an enterprise's relational database (RDBMS) system. We provide examples of powerful applications enabled by the technology. We then consider XML, a new markup language standard; we conjecture how XML will transform the web, and the role that will be played by Virtual Database technology in this transformation.

1 Introduction

Virtual database (VDB) technology makes the World Wide Web and other external data sources behave as an extension of an enterprise's relational database (RDBMS) system. According to some estimates, as much as 90% of the world's data is outside of relational database systems. Vital data is scattered across web sites, file systems, database systems, and legacy applications. These data sources differ in the way they organize the data, in the vocabulary they use, and in their data-access mechanisms. Many of them do not even support native query operations. Writing applications that combine data from these sources is a complex, often impossible, task because of the heterogeneity involved.

Junglee's patent-pending VDB technology can fundamentally transform enterprise computing and the World-Wide Web by providing a solution to this data scatter problem. VDB technology lets applications ask powerful queries of data that is scattered over a variety of data sources. The VDB gathers, structures and integrates the data from these disparate data sources and provides the application programmer with the appearance of a single, unified relational database system. VDB technology enables the development of an exciting new breed of applications that use all the data.

As an illustration of the applications enabled by VDB technology, consider job hunting on the Web. In order to make a meaningful career choice, a job seeker needs information on available opportunities as well as related data — such as information on housing, school districts, and crime statistics in the job area. Information on job openings is scattered across thousands of different web sites — company home pages and several aggregate sites,

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

†Phone (408) 522-9494; Fax (408) 522-9470. For more information contact Anand Rajaraman at anand@junglee.com

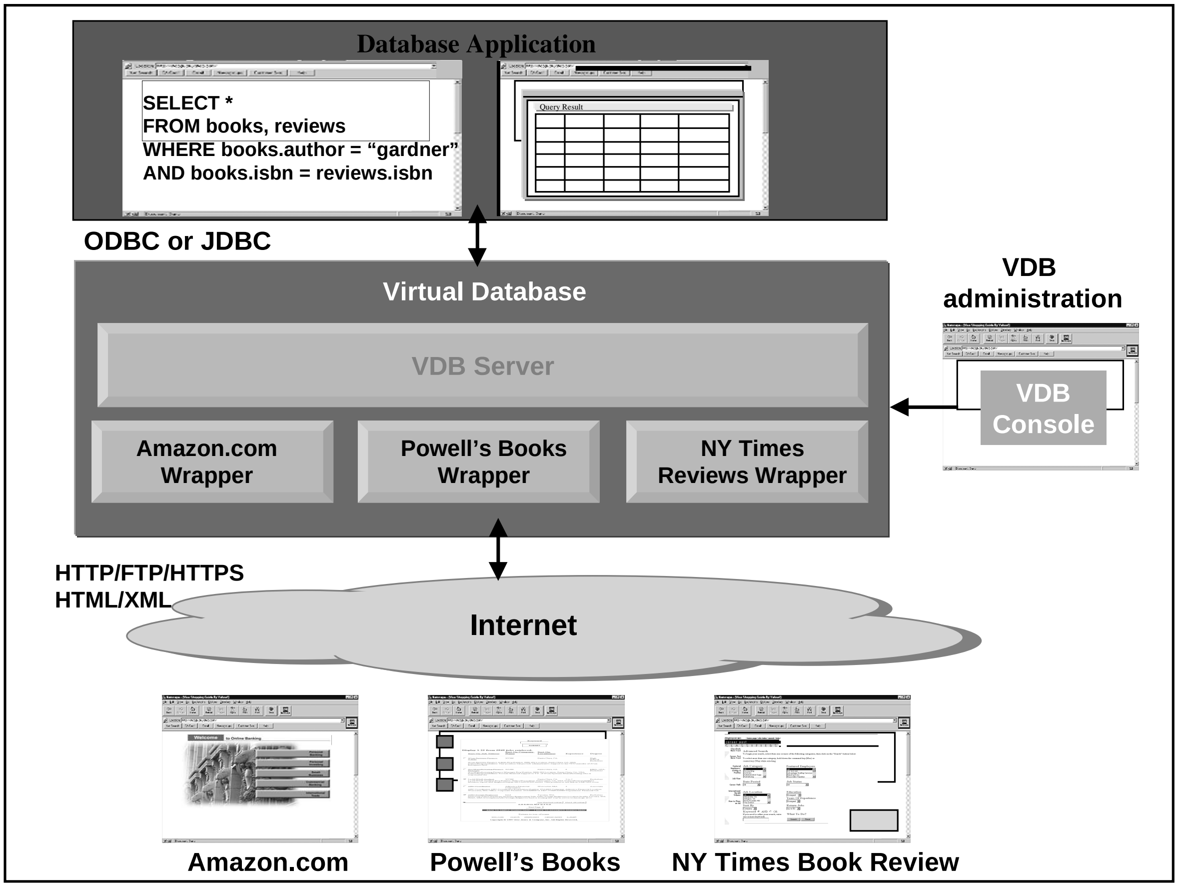

Figure 1: The Books Virtual Database

such as newspaper classifieds sites. Keyword search capabilities on words appearing in the job listing are the only available search choice.

VDB technology converts all these data sources into a single virtual relational database. Using an application based on VDB technology, the job seeker can now obtain answers to the following query posed to the Web, “find marketing manager positions in a company that is within 15 miles of San Francisco and whose stock price has been growing at a rate of at least 25% per year over the last three years.” This single query would span the Web employment listings of many corporations, in addition to web sites that have geographical mapping information and websites that contain historical records of corporate equity prices. The query would also return, for each position, related information including statistics on housing prices, school districts, and crime statistics. Section 3 provides details on this and other VDB applications that are deployed on several high-traffic web sites, including those of Yahoo!, The Wall Street Journal, The Washington Post, and San Jose Mercury News.

2 Technology Architecture

Figure 1 is a run-time view of a simple Virtual Database (VDB), which we’ll call the Books VDB for future reference. This VDB integrates the contents of two bookstores (Amazon.com and Powell’s Books) and the New York Times Book Reviews and presents a unified schema with two tables, books and reviews. The database application operates on this unified schema, issuing SQL queries through the JDBC or ODBC API; the application itelf can be built using standard RAD tools such as Delphi, PowerBuilder, Visual Basic, or similar Java toolkits.

The VDB is accessed through the VDB Server, and is administered through the browser-based VDB Console. The VDB also contains, for each external data source, a wrapper that interfaces the data source to the VDB server. A wrapper makes an arbitrary external data source, such as a web site, behave like an RDBMS, while the VDB Server integrates these separate relational databases into a unified Virtual Database (VDB).

A wrapper interfaces with a web site, typically using HTTP and HTML or XML. It handles HTTP protocol-related issues such as forms, cookies, and authentication. The wrapper is accessed via the JDBC API, through which clients can issue SQL queries. A SQL query issued to the wrapper might result in the wrapper filling out a HTML form on the Amazon.com web site, navigating and parsing the resulting HTML pages, and transforming the data into rows in a relational table. The wrapper uses extraction rules to apply sophisticated linguistic processing to extract attributes from the web pages, uses data transformation rules to transform and format the data to fit the schema, and uses the data validation rules to ensure data integrity.

Lightweight Java applications that interact with one (or a few) data sources can interface directly with wrappers. The application sees each data source as a separate JDBC source with its own schema, and must connect to each source separately and combine the data as needed.

Sophisticated applications that use more than a few data sources use the full functionality of the VDBMS, as shown in Figure 1. The VDBMS exposes tables in multiple data sources as virtual tables in a single Virtual Database (VDB), and supports full RDBMS functionality over virtual tables including view definitions and query processing across sources. In the example of Figure 3, the VDB defines the view books as the union of the amazon and powell's virtual tables. When the VDBMS receives the query shown in the figure, the query processor component decomposes the query, determines the fragments to be sent down to the individual data sources, and combines their results. The query result cache caches results from data sources for performance. In addition, the publishing system can be set up to periodically create physical snapshots of virtual tables in a local relational data store in order to speed up data access.

3 The VDB At Work: Real-World Applications

Junglee has applied VDB technology in several key domains: Employment Classifieds, Consumer Shopping, Real Estate, and Apartment listings. We describe below two such applications.

3.1 Online Recruitment

The JobCanopy VDB application integrates job listings from over 700 data sources, including employer web sites, flat files, and legacy data feeds. The schema for this VDB includes 31 attributes of interest to employers and jobseekers, including job title, job category, job location, and contact information. These data sources are scoured each week to ensure that the information is always fresh. Listings from different employers are normalized to have the same set of fields and the same vocabulary. The JobCanopy product is accessible from the web sites of several major newspapers and online media companies, including The Wall Street Journal Interactive Edition, The Washington Post, The San Jose Mercury News, Classifieds2000, and Westech Virtual Job Fair.

3.2 Web Commerce

The ShopCanopy VDB application allows comparison shopping over 40 merchants in 8 categories, including Books, Music, Computer Hardware, and Consumer Electronics. ShopCanopy is deployed on the Yahoo! Visa Shopping Guide web site at http://shopguide.yahoo.com.

The ShopCanopy application brings together buyers and sellers online to create marketplaces on the Web. ShopCanopy allows consumers to easily access and compare product and pricing information from merchants simultaneously, and then link to a specific merchant's site to make a purchase. VDB technology reduces the time spent looking for specific items by searching through affiliated online merchants and compiling a single list of all the vendors that offer the specified item, plus availability, shipping, pricing and other information helpful for making product choices.

4 The Three Phases of XML

The ever-increasing reach of the Web is based to a large extent on the simplicity of HTML, which enables authors to distribute documents at low cost and with ease. Most documents on the Web today are stored and transmitted in HTML. HTML is adequate for handling the presentation aspects of small and simple documents. Going beyond presentation of data and documents, HTML is used today to represent the form-based query capabilities and transactional functionality of Web sites. For example, a bookstore would allow shoppers to search for books by author or title. This query capability is presented through an HTML form. A transaction, e.g., the ability to buy a book and order its shipment, is also presented through HTML.

Virtual database technology transforms the Web into a database, using adapters (called *wrappers*) that analyze the HTML from Web sites to present them as relational data sources. The limited descriptive capability of HTML necessitates manual analysis for the creation of adapters. XML will add the next level of automation. As data sources become self-describing, VDB technology will automatically create the adapters. Virtual databases of several hundred thousand data sources will occur. In fact, the entire Web could become one unified database, fulfilling Junglees vision. With the increasing complexity of documents and their inter-relationships, the Web is evolving from a collection of hyperlinked documents to dynamic content that is generated from relational databases, document libraries and other forms of organized content. The limitations of HTML — stemming from the fact that structure, content and presentation are intermingled in HTML — is becoming increasingly evident to Web developers. The need for extensibility, structure and validation is the basis for the evolution of the Web towards XML. Both HTML and XML are derived from SGML (Standard Generalized Markup Language, ISO 8879). SGML allows documents to be self-describing, through the specification of tag sets and the structural relationships between the tags. This specification is referred to as the Document Type Definition, or DTD. HTML is a small hard-wired set of about 70 tags and 50 attributes, which allows HTML users to skip the self-describing aspect from a document. XML, on the other hand, retains the key SGML advantage of self-description through DTDs, while avoiding the complexity of full-blown SGML

The XML specification was adopted as a standard by the W3C in February 1998. Since then, there has been a groundswell of support for XML from the development community. We believe that XML will become the dominant data interchange format on the web, and that this transition will happen in three phases outlined below.

4.1 Phase 1: Data in XML

The delivery of XML content to browsers from Web sites enables the distribution of a significant proportion of the processing load from the Web server to the Web client. Sorting, grouping and pagination can be performed locally, avoiding round-trips to the Web server. The highly structured delivery of data enables clients to present different views of the same data to different users through style sheets.

Phase 1 represents a gigantic leap forward towards structured data interchange between Web servers and browsers, and potentially between Web servers themselves.

4.2 Phase 2: Data and queries in XML

The growth in the number of query-based Web data sources presents a challenge and a business opportunity for Web search engines, Web portals and Web content aggregators that will push XML to the next level. In Phase 2, Web sites will describe their query capabilities through XML to facilitate integration of individual Web sites into structured search engines and content syndicates. For example, a bookstore's web site may state that it allows searches by author or by title, but not a search that will enumerate all the books in the store.

A key benefit from describing query capabilities in XML is the ability to present the query based on a style sheet, enabling custom search forms for individual users. In addition, it would enable Web clients to mediate between small collections of heterogeneous Web data sources.

In Phase 2, VDB technology will be the enabling technology for the next generation of search engines. This next generation will go beyond mere keyword searches, allowing domain-specific attribute-based searches. For example, a corporate procurement search engine would allow procurement analysts to search the entire collection of supplier catalogs on the Web by supplier location, product category, category-specific attributes and price range. Cross-domain searches will present new business opportunities. VDB technology will normalize the query capabilities across Web sites, presenting uniform query capabilities across vast collections of autonomous Web sites conceivably the entire Web.

4.3 Phase 3: Data, queries and transactions in XML

Phase 3 will accelerate the pace of virtualization of organizations, where each organization focuses on its core competencies, and increasingly use Web-based or extranet-based outsourcing of ancillary services. However, to the consumer, there is an illusion of a complete service-provider. There are signs of this trend even today in financial services, health care, retail, business-to-business suppliers and electronic marketplaces. VDB technology will form the basis for the virtual organization, leveraging its ability to pull together disparate transaction capabilities into a common model. Web-based transaction integration follows naturally from the Web data integration capabilities of the preceding phases. In other words, the VDB search engines of Phase 2 will naturally gravitate to full-service transaction facilities, to cement their position in the value chain to the consumer.

5 Conclusion

VDB technology enables rapid deployment of applications with at least one of the following characteristics:

- Large numbers of data sources
- Data sources are autonomous, there is no centralized control
- Data sources can have a mixture of structured and unstructured data

The World Wide Web, and most Intranets, have all of these characteristics. The emergence of XML and related standards, such as RDF, will accelerate the deployment of VDB technology since they have the potential to lower drastically the cost of incorporating a web site into a virtual database.

Acknowledgements

We thank Ashish Gupta for helpful discussions, and the entire Junglee team who have implemented everything described in this paper.

CALL FOR PAPERS

DATA ENGINEERING

15th International Conference on

Data Engineering

March 23-26, 1999
Millennium Hotel, Sydney, Australia
Sponsored by the IEEE Computer Society

IEEE

Scope

Data Engineering deals with the use of engineering techniques and methodologies in the design, development and assessment of information systems for different computing platforms and application environments. The 15th International Conference on Data Engineering will continue in its tradition of being a premier forum for presentation of research results and advanced data-intensive applications and discussion of issues on data and knowledge engineering. The mission of the conference is to share research solutions to problems of today's information society and to identify new issues and directions for future research and development work.

Organizing Committee

General Co-chairs:	John Hiller (UNSW, Australia) Robert Meersman (Frije Univ Brussels, Belgium)
Program Co-chairs:	Mike Papazoglou (Tilburg Univ, Netherlands) Calton Pu (OGI, USA) Masaru Kitsuregawa (Univ of Tokyo, Japan)
Panel Program Chair:	Bhavani Thuraisingham (MITRE, USA)
Tutorial Program Chair:	Athman Bouguettaya (QUT, Australia)
Industrial Program Chair:	Meichun Hsu (HP Labs, USA)
Organizing Chairs:	Dorota Kieronska (Curtin Univ Tech, Australia) Anne H.H. Ngu (UNSW, Australia)
Publication Chair:	Leszek Maciaszek (Macquarie Univ, Australia)
Financial Chair:	Alan Fekete (Univ Sydney, Australia)

Topics of Interest

These include (but are not restricted to):

- Internet & WWW data management systems
- Information resource discovery
- Agent Technology
- Electronic commerce
- Digital libraries

- Distributed object management
- Cooperative Information Systems
- Business process modeling and reengineering
- Information and data modeling
- Metadata use and management

- Storage structures
- Performance and benchmarking
- Query processing
- Parallel and distributed databases
- Active databases
- Temporal databases

- Interoperability and heterogeneous systems
- Advanced transaction models and management
- Workflow modeling and management

- Legacy data access and management
- Uncertainty and information quality

- OLAP & data warehouses
- Data mining
- GIS and spatial databases
- Multimedia information systems
- Mobile Systems

Important Dates

Abstract submissions: Aug 21, 1998
Paper submissions: Aug 28, 1998
Panel/tutorial proposals: Aug 28, 1998
Acceptance notification: Oct 30, 1998
Camera-ready copies: Jan 11, 1999
Tutorials: Mar 23, 1999
Conference: Mar 24-26, 1999

Paper Submission

Six copies of original papers not exceeding 6000 words (25 double spaced pages) should be submitted to the program co-chair of the author's respective region by **August 28, 1998**. An abstract (no more than 300 words in ASCII text) of each intended submission should be sent via email to the author's regional program co-chair by **August 21, 1998** (*firm* deadline). The message should include the title of the paper, authors' names, the abstract, the one or two areas most relevant to the paper, and whether the paper is submitted to the research or industrial program. If the paper falls outside the specific areas handled by PC vice-chairs, indicate this fact and include keywords that characterize the paper for proper reviewing.

Program Co-Chairs

Europe:
Mike Papazoglou
INFOLAB, Tilburg University
P.O. Box 90153
5000 LE, Tilburg, The Netherlands
Phone: +31-13-466-23-49/30-20
Email: mikep@kub.nl

Americas:
Calton Pu
OGI/CSE
P.O. Box 91000
Portland, OR 97291-1000, USA
(for express mail only)
20000 NW Walker Road
Beaverton, OR 97006, USA
Phone +1-503-690-1112
Email: icde99@cse.ogi.edu

Asia/Oceania:
Masaru Kitsuregawa
Institute of Industrial Sciences,
University of Tokyo
7-22-1, Roppongi, Minato-ku,
TOKYO, JAPAN.
Phone: +81-3-3402-6231
Email: icde99asia@tkl.iis.u-tokyo.ac.jp

Conference Web Site

http://www.cse.unsw.edu.au/icde99/

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398