Interactive Query and Search in Semistructured Databases*

Roy Goldman, Jennifer Widom

Stanford University
{royg,widom}@cs.stanford.edu

Abstract

Semistructured graph-based databases have been proposed as well-suited stores for World-Wide Web data. Yet so far, languages for querying such data are too complex for casual Web users. Further, proposed query approaches do not take advantage of the interactivity of typical Web sessions—users are proficient at iteratively refining their Web explorations. In this paper we propose a new model for interactively querying and searching semistructured databases. Users can begin with a simple keyword search, dynamically browse the structure of the result, and then submit further refining queries. Enabling this model exposes new requirements of a semistructured database that are not apparent under traditional database uses. We demonstrate the importance of efficient keyword search, structural summaries of query results, and support for inverse pointers. We also describe some preliminary solutions to these technical issues.

1 Introduction

Querying the Web has understandably gathered much attention from both research and industry. For searching the entire Web, search engines are a well-proven, successful technology [Dig97, Ink96]. Search engines assume little about the semantics of a document, which works well for the conglomeration of disparate data sources that make up the Web. But for searching within a single Web site, a search engine may be too blunt a tool. Large Web sites, with thousands of pages, are attracting millions of users. The ESPN Sports site (espn.com), for example, has over 90,000 pages [Sta96] and several million page views a day [Sta97]. As large as some sites may be, they are fundamentally different from the Web as a whole since a single site usually has a controlled point of administration. Thus, it becomes possible to consistently assign and expose the site's semantic data relationships and thereby enable more expressive searches.

Consider any of the large commercial news Web sites, such as CNN (cnn.com), ABC News (abcnews.com), etc. Currently, users have very limited querying ability over the large amounts of data at these sites. A user can browse the hardcoded menu system, examine a hand-made subject index, or use a keyword-based search engine. When looking for specific data, traversing the menus may be far too time consuming, and of course a hand-made subject index will be of limited scope. Finally, while a keyword search engine may help locate relevant data, it doesn't take advantage of the conceptual data relationships known and maintained at the site. At any such Web site today, there is no convenient way to find:

- All photos of Bill Clinton in 1997
- All articles about snow written during the summer
- All basketball teams that won last night by more than 10 points

Such queries become possible if most or all of the site's data is stored in a database. Recently, numerous researchers have proposed *semistructured* data models, databases, and languages for modeling, storing, and querying World-Wide Web data [AQM+96, BDHS96, BDS95, FFLS97, MAG+97]. Such proposals argue that a graph-based semistructured database, without the requirement of an explicit schema, is better suited than traditional database systems for storing the varied, dynamic data of the Web. So far, however, there has been little discussion of who will query such data and what typical queries will look like. Given the domain, we feel that a large and important group of clients will simply be casual Web users, who will want to pose interesting queries over a site's data.

How would a typical Web user pose such queries? Asking casual users to type a query in any database language is unrealistic. It is possible to handle certain queries by having users fill in hardcoded forms, but this approach by nature limits query flexibility. Previous work on *DataGuides* [GW97] has proposed an interactive query tool that presents a dynamic structural summary of semistructured data and allows users to specify queries "by example." A DataGuide

* This work was supported by the Air Force Rome Laboratories and DARPA under Contracts F30602-95-C-0119 and F30602-96-1-031.

summarizes all paths through a database, starting from a root. While such dynamic summaries are an important basic technology for several reasons [GW97], presenting the user with a summary of paths from the root may force him to explore much unnecessary database structure. In this paper, targeting casual users, our strategy is to model and exploit two key techniques that Web users are intimately familiar with:

1. specifying a simple query to begin a search, usually with keywords
2. further exploring and refining the results

For the first technique, we want to support very simple queries that help "focus" the user on relevant data. The many search engines on the Web have shown that keyword search is an easy and effective technique for beginning a search. To enable the second technique, we want to expose and summarize the structure of the database "surrounding" any query result. To do this, we dynamically build and present a DataGuide that summarizes paths not from the database root, but instead from the objects returned in the query result. A user can then repeat the process by submitting a query from this "focused" DataGuide or specifying additional keywords, ultimately locating the desired results.

Our discussions are in the context of the *Lore* project [MAG+97], which uses the *OEM* graph-based data model [PGMW95] and the *Lorel* query language [AQM+96]. Our results are applicable to other similar graph-based data models, as well as the emerging XML standard for defining the semantic structure of HTML documents [Con97].

In the rest of the paper, we first provide background and context in Section 2. In Section 3, we present a simple motivating example to illustrate why new functionality is needed in a semistructured database system to support interactive query and search. Our session model is described in Section 4, followed by three sections covering the new required technology:

- Keyword search (Section 5): Efficient data structures and indexing techniques are needed for quickly finding objects that match keyword search criteria. While we may borrow heavily from well-proven information retrieval (IR) technology, the new context of a graph database is sufficiently different from a simple set of documents to warrant investigation.

- "Lazy" DataGuides (Section 6): Computing a DataGuide over each query result can be very expensive, so we have developed new algorithms for computing and presenting DataGuides piecewise, computing more on demand.

- Inverse pointers (Section 7): To fully expose the structural context of a query result, it is crucial to exploit inverse pointers when creating the DataGuide for the result, browsing the data, and submitting refining queries. While support for inverse pointers may seem straightforward, the major proposed models for semistructured data are based on directed graphs, and inverse pointers have not been considered in the proposed query languages [AQM+96, BDHS96, FFLS97].

2 Background

To set the stage for the rest of the paper, we briefly describe the OEM data model, introduce the Lorel query language, and summarize DataGuides. In OEM, each object contains an object identifier (oid) and a value. A value may be atomic or complex. Atomic values may be integers, reals, strings, images, or any other indivisible data. A complex OEM value is a collection of OEM subobjects, each linked to the parent via a descriptive textual label. An OEM database can be thought of as a rooted, directed graph. The left side of Figure 1 is a tiny subset of an OEM database that we use at Stanford to describe the Database Group, rooted at object r.

The Lorel query language, based on OQL [Cat94], evaluates queries based on *path expressions* describing traversals through the database. Special edges coming from the root are designated as *names*, which serve as entry points into the database. In Figure 1, DBGroup is the only name. As a very simple example, the Lorel query "Select DBGroup.Member.Publication.Title" returns a set containing object 11, with value "OLTP." More specifically, when a query returns a result, a new named Answer object is created in the database, and all objects in the result are made children of the Answer.[1] The Answer edge is available as a name for successive queries.

[1] Identifying labels are assigned to the edges connecting the Answer to each query result object, based on the last edge traversed to reach the object during query evaluation. In this example, the label is Title. Also, Lorel queries may create more complicated object structures as query results, but for simplicity we ignore such queries here; our work can easily be generalized.

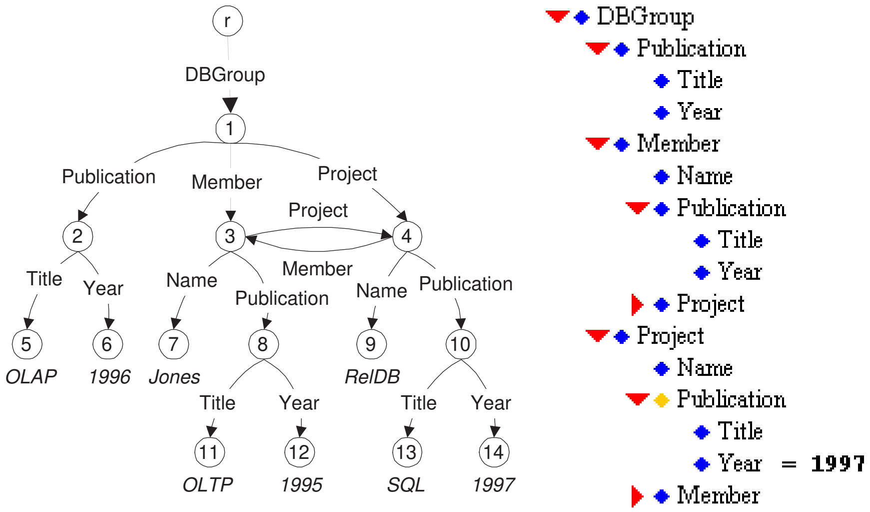

Figure 1: A sample OEM database and its Java DataGuide

A DataGuide is a dynamic structural summary of an OEM database. It is an OEM object G that summarizes the OEM database (object) D, such that every distinct label path from the root of D appears exactly once as a path from the root of G. Further, every path from the root of G corresponds to a path that exists in D. We have carefully chosen Figure 1 to be a DataGuide of itself (ignoring atomic values). For any given sequence of labels, there is only one corresponding path in the database. (In the real database, there are of course many Member objects under DBGroup, several Publication objects per Project, etc.) Through a Web-accessible Java interface, a DataGuide is presented as a hierarchical structure, and a user can interactively explore it. The right side of Figure 1 shows the Java DataGuide for our sample database. Clicking on an arrow expands or collapses complex objects. We have expanded most of the links, but because of the cycle we have not expanded the deepest Project or Member arrows.

Users can also specify queries directly from the Java DataGuide with two simple steps: 1) selecting paths for the query result, and 2) adding filtering conditions. Each diamond in the DataGuide corresponds to a labeled path through the database. By clicking on a diamond, a user can specify a condition for the path or select the path for the query result. Filtering conditions are rendered next to the label, and the diamonds for selected paths are highlighted. The Java DataGuide in Figure 1 shows the query to select all project publications from 1997. The DataGuide generates Lorel queries, which are sent to the Lore server to be evaluated. In our Web user interface, we format the query results hierarchically in HTML for easy browsing.

3 Motivating Example

In this section we trace a motivating example, using the sample database presented in Figure 1. Suppose a user wishes to find all publications from 1997, a seemingly simple query. (In the previous section, our sample query only found publications of projects.) It is possible to write a Lorel query to find this result, but a casual user will not want to enter a textual Lorel query. This example also illustrates some limitations of using the DataGuide to locate information. Even in this simple case, there are numerous paths to all of the publications; in a larger database the situation may be much worse. In short, while the DataGuide does a good job of summarizing paths from the root, a user may be interested in certain data independent of the particular topology of a database.

In this situation, a typical Web user would be comfortable entering keywords: “Publication,” “1997,” or both. Suppose for now the user types “Publication” to get started. (We will address the case where the user types “1997” momentarily, and we discuss the issue of multiple keywords in Section 5.) If the system generates a collection of all Publication objects, the answer is $\{2, 8, 10\}$, identified by the name Answer. While this initial search has helped focus our search, we really only wanted the Publications in 1997. One approach would be to browse all of the objects in

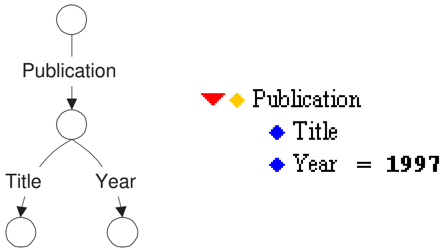

Figure 2: DataGuide constructed over result of finding all publications

the result, but again in a larger database this may be difficult. Rather, we dynamically generate a DataGuide over the answer, as shown in Figure 2. Notice now that even though Title and Year objects were reachable along numerous paths in the original DataGuide, they are consolidated in Figure 2. As shown in the Java DataGuide, the user can mark Publication for selection and enter a filtering condition for Year to retrieve all 1997 publications. Getting the same result in the original DataGuide would have required three selection/filtering condition pairs, one for each possible path to a Publication.

The above scenario motivates the need for efficient keyword search and efficient DataGuide creation over query results. Next, we show how these features essentially force a system to support inverse pointers as well. Suppose the user had typed "1997" rather than "Publication." This time, the answer in our sample database is the singleton set $\{14\}$, and the DataGuide over the result is empty since the result is just an atomic object. This example illustrates that what the user needs to see in general is the area "surrounding" the result objects, not just their subobject structure as encapsulated by the DataGuide. Presenting the "surrounding area" to the user clearly requires us to consider inverse pointers, such as pointers from a year to the publications from that year. We can then create a more descriptive DataGuide over our results.

4 Query and Search Session Model

We define a Lore *session* over an initial database D_0, with root r and initial DataGuide $G_0(r)$, as a sequence of *queries* $q_1, q_2, \ldots q_n$. A query can be a "by example" DataGuide query, a list of keywords, or, for advanced users, an arbitrary Lorel query. The objects returned by each query q_i are accessible via a complex object a_i with name $Answer_i$. After each query, we generate and present a DataGuide $G_i(a_i)$ over the result, and users can also browse the objects in each query result. Perhaps counterintuitive to the notion of narrowing a search, we do not restrict the database after each query. In fact, the database D will grow monotonically after each query q_i. After q_i, $D_i = D_{i-1} \cup a_i$. Essentially, each DataGuide helps focus the user's next query without restricting the available data. In the following three sections, we discuss three technologies that enable efficient realization of this model of interaction.

5 Keyword Search

In the IR arena, a keyword search typically returns a ranked list of documents containing the specified keywords. In a semistructured database, pertinent information is found both in atomic values and in labels on edges. Thus, it makes sense to identify both atomic objects matching the specified word(s) and objects with matching incoming labels. For example, if a user enters "Publication," we would like to return all objects pointed to by a "Publication" edge, along with any atomic objects with the word "Publication" in their data. This approach is similar in spirit to the way keyword searches are handled by Yahoo! (yahoo.com). There, search results contain both the *category* and *site* matches for the specified keywords.

While a keyword search over values and labels is expressible as a query in languages like Lorel and UnQl, the issue of efficient execution has not been addressed. In Lore, we have built two inverted-list indexes to handle this type of query. The first index maps words to atomic objects containing those words, with some limited IR capabilities such as *and*, *or*, *near*, etc. The second index maps words to edges with matching labels.

An interesting issue is how to handle multiple keywords. It is limiting to restrict our searches to finding multiple keywords within a single OEM object or label, since our model encourages decomposition into many small objects. Hence, we would like to efficiently identify objects and/or edges that contain the specified keywords and are also near each other in terms of link distance. We are currently investigating this issue. We are also exploring the topic of how to group or rank the results, an essential aspect of any search engine that may return large answer sets.

6 "Lazy" DataGuides

As described in the motivating example, we wish to build DataGuides over query results. For this section, let us ignore the issue of inverse pointers. As shown in [GW97], computing a DataGuide can be expensive: the worst case running time is exponential in the size of the database, and for a large database even linear running time would be too slow for an interactive session. We thus introduce two techniques to improve the running time.

First, we can exploit certain auxiliary data structures that are built to provide incremental DataGuide maintenance [GW97]. These structures guarantee that we never need to recompute a "sub-DataGuide" that has previously been constructed. In Figure 1, suppose a user searches for all "Projects," a query that would return the singleton set $\{4\}$. In this case, the DataGuide over $\{4\}$ is the same as the sub-DataGuide reachable along DBGroup.Project in the original DataGuide. We can dynamically determine this fact with a single hash table lookup, and no additional computation is needed.

Second, we observe that an interactive user will rarely need to explore the entire DataGuide. Our experience shows that even in the initial DataGuide, users rarely explore more than a few levels. Most likely, after a reasonable "focusing" query, users will want to browse the structure of objects near the objects in the query result. Hence, we have modified the original depth-first DataGuide construction algorithm to instead work breadth-first, and we have changed the algorithm to build the DataGuide "lazily," i.e., a piece at a time. From the user's perspective, the difference is transparent except with respect to speed. When a user clicks on an arrow for a region that hasn't yet been computed, behind the scenes we send a request to Lore to generate and return more of the DataGuide. Our maintenance structures make it easy to interrupt DataGuide computation and continue later with no redundant work.

7 Inverse Pointers

Directed graphs are a popular choice for modeling semistructured data, and the proposed query languages are closely tied to this model. While powerful regular expressions have been proposed for traversing forward links, essentially no language support has been given to the problem of directly traversing inverse pointers. As our motivating example demonstrates, a parent may be just as important as a child for locating specific data.

Adding inverse pointers affects many levels of a semistructured database system, including object storage, creation of DataGuides (or any other summarizing technique), query language design, and query execution. Physically, inverse pointers may be clustered with each object's data or stored in a separate index. Logically, we try to make access to inverse pointers as seamless as possible: for an object *O* with an incoming label "X" from another object *P*, we conceptually make *P* a child of *O* via the label "X Of." With this approach, inverse edges can be treated for the most part as additional forward edges. Next, we focus on how exposing inverse pointers affects DataGuides, graph query languages, and query execution.

DataGuide Creation: We wish to extend DataGuides to summarize a database in all directions, rather than only by following forward links. If the "Of" links described above are simply added to the database graph, then we need not even modify our DataGuide algorithms. Unfortunately, this approach can yield some unintuitive results. In OEM and most graph-based database models, objects are identified by their incoming labels. A "Publication," for example, is an object with an incoming Publication edge. This basic assumption is used by the DataGuide, which summarizes a database by grouping together objects with identical incoming labels. An "Of" link, however, does a poor job of identifying an object. For example, given an object *O* with an incoming TitleOf link, we have no way of knowing whether *O* is a publication, book, play, or song. Therefore, a DataGuide may group unrelated objects together. For example, suppose a user's initial search over a library database finds some Title objects. Figure 3(a) shows three atomic objects in the result (shaded in the figure), with dashed "Of" links to show their surrounding structure. Figure 3(b)

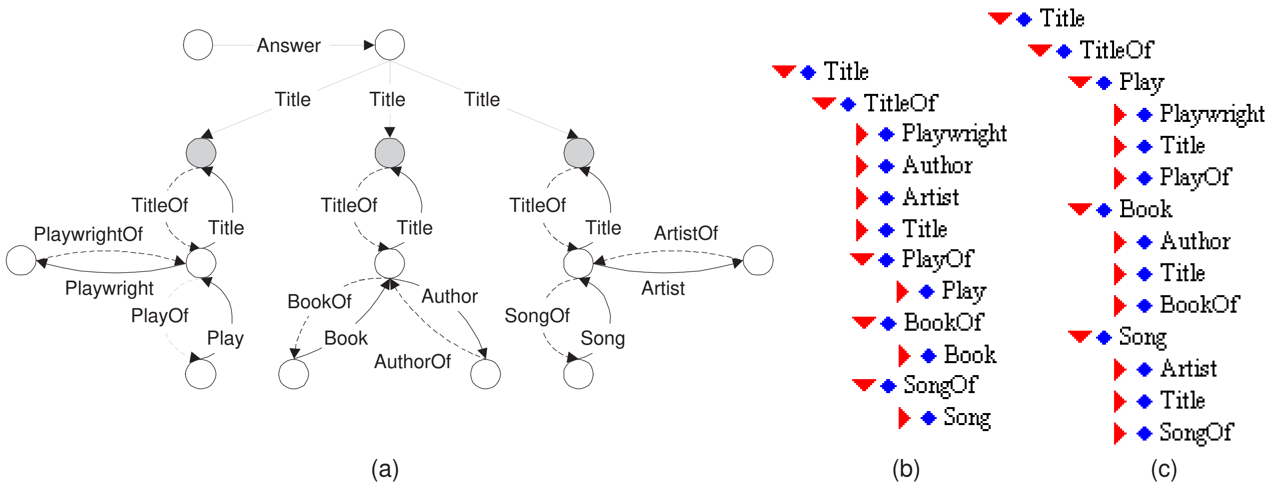

Figure 3: An OEM query result and two potential DataGuides

shows the standard DataGuide over this Answer. The problems with 3(b) should be clear: the labels shown under TitleOf are confusing, since the algorithm has grouped unrelated objects together. Further, the labels directly under TitleOf do not clearly indicate that our result includes titles of books, plays, and songs. To address the problem, we have modified the DataGuide algorithm slightly to further decompose all objects reachable along an "Of" link based on the non-"Of" links to those objects. Figure 3(c) shows the more intuitive result, which we refer to as a *Panoramic DataGuide*. Note that adding inverse pointers to DataGuide creation adds many more edges and objects than in the original DataGuide, making our new support for "lazy" DataGuides even more important.

Query Language & Execution: Just as users can specify queries "by example" with the original DataGuide, we would like to allow users to specify queries with Panoramic DataGuides as well. Suppose in Figure 3(c) a user selects Author to find the authors of all books having titles in the initial result. In Lorel, which currently does not support direct access to inverse pointers, the generated query is:

```
Select A
From Answer.Title T1, #.Book B, B.Title T2, B.Author A
Where T1 = T2
```

This query essentially performs a join between the titles in our answer and all book titles in the entire database, returning the authors of each such book. The # is a wildcard representing any path, and because of this wildcard a naive execution strategy could be very expensive. Efficient execution based on forward pointers alone depends on having an index that quickly returns all Book objects in the database, and we support such an index in Lore. If we store inverse pointers in the system, we might be able to train the optimizer to exploit them for such queries; rather than finding all Book objects and performing the join, the system could simply follow inverse and then forward pointers from each Title in the initial result. However, it could be difficult to recognize and optimize these cases. Another approach is to allow inverse links to be specified directly in path expressions in the language.

8 Implementation Status

Our interactive query and search model, along with the necessary supporting features discussed in this paper, are under development within the Lore project. We keep our online Lore demo up-to-date, reflecting new designs as they are completed. Please visit www-db.stanford.edu/lore.

References

[AQM+96] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel query language for semistructured data. *Journal of Digital Libraries*, 1(1), November 1996.

[BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 505–516, Montreal, Canada, June 1996.

[BDS95] P. Buneman, S. Davidson, and D. Suciu. Programming constructs for unstructured data. In *Proceedings of the 1995 International Workshop on Database Programming Languages (DBPL)*, 1995.

[Cat94] R.G.G. Cattell. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, San Francisco, California, 1994.

[Con97] World Wide Web Consortium. Extensible markup language (XML). http://www.w3.org/TR/WD-xml-lang-970331.html, December 1997. Proposed recommendation.

[Dig97] Digital Equipment Corp. About AltaVista: our technology. http://altavista.digital.com/av/content/about_our_technology.htm, 1997.

[FFLS97] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for a web-site management system. *SIGMOD Record*, 26(3), September 1997.

[GW97] R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured data. In *Proceedings of the Twenty-Third International Conference on Very Large Data Bases*, 1997.

[Ink96] Inktomi Corp. The technology behind HotBot. http://www.inktomi.com/Tech/CoupClustWhitePap.html, 1996. White paper.

[MAG+97] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3), September 1997.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 251–260, Taipei, Taiwan, March 1995.

[Sta96] Starwave Corp. About ESPN. http://www.starwave.com/starwave/about.espn.html, 1996.

[Sta97] Starwave Corp. ESPN SportsZone surpasses 2 billion page views. http://www.starwave.com/starwave/releases.sz.billion.html, December 1997. Press release.