Squeal: A Structured Query Language for the Web

Ellen Spertus
Mills College
spertus@mills.edu

Lynn Andrea Stein
MIT Artificial Intelligence Lab
las@ai.mit.edu

Abstract

The Web contains an abundance of useful semi-structured information that can and should be mined. Types of structure include hyperlinks between pages, structure within hypertext pages, and structure within URLs. We have implemented a programming language, Squeal, that facilitates structure-based queries. Specifically, the Squeal user can query the Web as if it were in a standard relational database. We describe Squeal and show the ease of writing structure-based information tools in Squeal.

Keywords: Web, structure, semi-structured information, hyperlinks, hypertext, relational databases, SQL, recommender systems

1. Introduction

The success of the Web is a testament to the importance of structure. Web pages consist not only of text but also of intra-document structure (via annotations indicating headers, lists, and formatting directives) and inter-document structure (hyperlinks). Even the uniform resource locator (URL) identifying a page is structured: typically a host name followed by a location in the file hierarchy or a set of query keywords and values. All of these types of information are used automatically by human readers (through browser software) but have been awkward for programmers to make use of in their search tools. Some examples of structure-based queries are:

- What pages are pointed to by both Yahoo and Netscape Netcenter?
- What are the titles of pages that point to my home page?
- What are the most linked-to pages containing the phrase "java development kit"?
- What pages have the same text as my home page but appear on a different server?

We designed Squeal to allow programmers to easily express these types of queries.

Rather than invent a new query language for structured information, we decided to build on the most popular query language for databases: Structured Query Language (SQL). Benefits of this decision include:

- Anyone who knows SQL can program in Squeal.
- Implementations of Squeal can (and do) build on existing SQL implementations.
- Users can combine references to the Web with references to their own relational databases.
- GUIs and other tools built for SQL can be used with Squeal.

In the next section, we will present Squeal. In the following section, we will demonstrate the ease of writing structural queries in Squeal. We will conclude with a discussion of the implications and a comparison to other database-inspired languages for querying the Web.

2. Squeal

Squeal provides users with the illusion that the Web is stored and organized in a relational database. Squeal consists of two parts: (1) a schema describing the Web and (2) an implementation that answers queries about the

schema (even though the entire Web is not really in a database).

2.1 Schema

A *schema* describes the structure of a relational database, i.e., the tables, fields, and the relationships between them. For example, the schema for an employee database might include a table **employee** with fields *first_name*, *last_name*, and *social_security_number*, where each employee has a distinct *social_security_number*, but different employees may have the same *first_name* or *last_name*.

The following description of the Squeal schema is a slight oversimplification. (The real tables are more normalized and have additional fields.) For full information on the Squeal schema, see [Spertus 1998].

The **page** table, which describes a Web page, has the following fields:

- *url*: the page's URL
- *contents*: the text on the page
- *bytes*: the size of the page
- *when*: the date and time when the page was last retrieved

The following are examples of legal queries (with SQL keywords in capital letters and comments preceded by slashes):

```
// What text is on the page "http://www9.org"?
SELECT contents
FROM page
WHERE url="http://www9.org";

// What pages contain "hypertext" and have fewer than 1000 bytes?
SELECT url
FROM page
WHERE contents LIKE '%hypertext%'
AND bytes < 1000;
```

The **tag** table, which describes each html tag on a page, has the following fields:

- *url*: the page's URL
- *tag_id*: a unique number identifying this tag instance
- *name*: the text of this tag string (e.g., "H1" or "IMG")
- *startOffset*: the offset within the page at which the tag begins
- *endOffset*: the offset within the page at which the tag ends

Some legal queries are:

```
// What pages contain the word "hypertext" and contain a picture?
SELECT url
FROM page p, tag t
WHERE p.contents LIKE "%hypertext%"
AND t.url = p.url
AND t.name = "IMG";

// What tags appear on the page "http://www9.org"?
SELECT name
FROM tag
WHERE url="http://www9.org";
```

The **att** table contains information about the attribute names and values associated with each tag. (For example, consider the hypertext **"<IMG SRC=foo.gif ALIGN=center>"**. The tag value is "IMG", the first attribute The fields of the **att** table are:

- *tag_id*: a reference to the corresponding **tag**
- *name*: the attribute name, e.g., "SRC"
- *value*: the attribute value, e.g., "foo.gif"

Some legal queries are:

// What are the values of the SRC attributes associated with IMG tags on "http://www9.org"?

```
SELECT a.value
FROM att a, tag t
WHERE t.url = "http://www9.org"
AND t.name = "IMG"
AND a.tag_id = t.tag_id
AND a.name = "SRC";
```

// What pages are pointed to by "http://www9.org"?

```
SELECT a.value
FROM att a, tag t
WHERE t.url = "http://www9.org"
AND t.name = "A"
AND a.tag_id = t.tag_id
AND a.name = "HREF";
```

While it is possible to inquire about hyperlinks with the **tag** and **att** table, we also provide a more convenient **link** table with the following fields:

- *source_url*: the page on which the link occurs
- *anchor*: the anchor text for the link
- *destination_url*: the target of the link
- *hstruct*: information about under what headers within the page the link appears (e.g., "after the second H1 header and a subsequent H2 header")
- *lstruct*: information about within what lists within the page the link appears (e.g., "in the second doubly-nested list")

Some legal queries are:

// What pages are pointed to by "http://www9.org"?

```
SELECT destination_url
FROM link
WHERE source_url = "http://www9.org";
```

// What pages point to "http://www9.org"?

```
SELECT source_url
FROM link
WHERE destination_url = "http://www9.org";
```

// What pages are pointed to via hyperlinks with anchor text "Web conference"?

```
SELECT destination_url
FROM link
WHERE anchor = "Web conference";
```

The utility of the *hstruct* and *lstruct* fields will be shown later.

The **parse** table describes the structure of a URL. It has the following fields:

- *url_value*: the URL string
- *component*, one of the following, specifying the type of data that appears in the *value* field:
  - "host": the host portion of the URL
  - "port": the port number of the URL
  - "path": the path components (directory names) and file name
  - "ref": the URL's reference field (the portion after a pound sign (#))

- *value*: the value of the item specified by the *component* field
- *depth*: for file names, 0; for directory names, how close to the file name

For example, the URL "**`http://www.ai.mit.edu:80/people/index.html#S`**" would appear in the parse table as follows:

component	value	depth
host	www.ai.mit.edu	-
port	80	-
path	index.html	1
path	people	2
ref	S	-

We are planning to also add support for parsing parameterized arguments.

Summary

We have presented the highlights of the Squeal schema and shown some SQL queries that can be made on it. Note that all of the queries shown are standard SQL, thus the user need only learn our schema (just as one would to access any database) and not a new query language.

Implementation

The Squeal interpreter provides the illusion that the Web is in a relational database, with certain limitations because it is impractical (if not impossible [Abiteboul and Vianu 1997]) to determine complete information about the Web. Squeal responds to user queries by querying search engines and fetching and parsing Web pages, which it puts into a small local off-the-shelf relational database. Once the interpreter has caused the relevant information to be placed in the local database, it passes through the user query and returns the response. Because information is put into the database only when demanded, we call this a "just-in-time database" [Spertus and Stein 1998].

Let us look at how one of our sample queries would be interpreted:

```
// What pages are pointed to by "http://www9.org"?
SELECT destination_url
FROM link
WHERE source_url = "http://www9.org";
```

The Squeal interpreter would respond as follows:

1. Fetch the page "http://www9.org" from the Web.
2. Insert information about the page and the URL into the **page** and **parse** tables in the small local database.
3. Parse the page and store information about it into the **tag**, **att**, and **link** tables.
4. Pass the original query (`SELECT destination_url FROM link WHERE source_url = "http://www9.org"`) to the local database server, which has a SQL interpreter. It returns a list of URLs, which the Squeal interpreter returns to the user.

Let us look at another query:

```
// What pages point to "http://www9.org"?
SELECT source_url
FROM link
WHERE destination_url = "http://www9.org";
```

The Squeal interpreter would respond as follows:

1. Ask a search engine (e.g., AltaVista) what pages point to "http://www9.org".

2. Fetch from the Web all of the pages returned by the search engine, entering information about each page into the **page**, **parse**, **tag**, and **att** tables in the local database.
3. Pass the original query (`SELECT source_url FROM link WHERE destination_url = "http://www9.org"`) to the local database server, which will return a list of URLs, which the Squeal interpreter returns to the user.

The interpreter is described in more detail elsewhere [Spertus 1998, Spertus and Stein 1998].

3. Applications

We call Squeal applications *ParaSites* because they exploit information on the Web in a manner unintended by the information's authors. Our goal is not to argue that these are the best possible applications but to show the variety of useful structural queries and the ease with which they can be implemented in Squeal. Full details about the applications, including evaluations, can be found in [Spertus 1998].

3.1 Recommender System

One useful class of information retrieval applications is recommender systems [Resnick and Varian 1997], where a program recommends new Web pages (or some other resource) judged likely to be of interest to a user, based on the user's initial set of seed pages P. The standard text-based technique for recommender systems, used by the Excite search service (www.excite.com), is extracting keywords that appear on the seed pages and returning pages that contain these keywords. Note that this technique is based purely on the text of a page, independent of any inter- or intra-document structure.

Another technique for making recommendations is collaborative filtering [Shardanand and Maes 1995], where pages are recommended that were liked by other people who liked P. This is based on the observation that items thought valuable/similar by one user are likely to by another user. As collaborative filtering was currently practiced, users explicitly rated pages to indicate their recommendations. This inconvenient and expensive step can be eliminated through data mining by interpreting the act of creating hyperlinks to a page as being an implicit recommendation. In other words, if a person links to pages Q and R, we can guess that people who like Q may like R, especially if the links to Q and R appear near each other on the referencing page (such as within the same list). This mines intra-document structural information.

Accordingly, if a user requests a page similar to a set of pages {*P1, …, Pn*}, the system can find (e.g., through AltaVista) pages R that point to a maximal subset of these pages and then return to the user what other pages are referenced by R. Note that the ParaSite does not have to understand what the pages have in common. It just needs to find a list that includes the pages and can infer that whatever trait they have in common is also exemplified by other pages they point to.

For example, the first page returned from AltaVista that pointed to both Electronic Privacy Information Center ("www.epic.org") and Computer Professionals for Social Responsibility ("www.cpsr.org/home.html") was a list of organizations fighting the Communications Decency Act; links included the Electronic Frontier Foundation ("www.eff.org") and other related organizations.

We are not the only ones to use this technique, which derives from citation indexing [Kessler 1963, Small 1973], and many people have applied this technique to the Web [Kleinberg 1998, Dean and Henzinger 1999, Rousseau 1997]. What is novel is the ease with which such heuristics can be written and tested.

We use the following algorithm to find pages similar to P1 and P2:

1. Generate a list of pages R that point to P1 and P2.
2. List the pages most commonly pointed to by pages within R in order of number of occurrences.

The corresponding Squeal code is:

```
SELECT link3.destination_url, COUNT(*)
FROM link link1, link2, link3
WHERE link1.destination_url = p1
```

```
AND link2.destination_url = p2
AND link1.source_url = link2.source_url
AND link2.source_url = link3.source_url
GROUP BY link3.destination_url
ORDER BY COUNT(*) DESC;
```

Some heuristics for improving precision are:

1. Only return target pages that include a keyword specified by the user.
2. Return the names of hosts frequently referenced.
3. Only return target pages that point to one or both of P1 and P2.
4. Only follow links that appear in the same list and under the same header as the links to P1 and P2.

This last heuristic was motivated by the observation that some pages contains hundreds or thousands of links and that the most similar pairs of links are likely to be within the same list or under the same header. It makes use of the *hstruct* and *lstruct* fields described above.

The Squeal code for the recommender system, including the requirement that links occur within the same list, is:

```
SELECT link3.destination_url, COUNT(*)
FROM link link1, link2, link3
WHERE link1.destination_url = p1
AND link2.destination_url = p2
AND link1.source_url = link2.source_url
AND link2.source_url = link3.source_url
AND link1.lstruct = link2.lstruct
AND link2.lstruct = link3.lstruct
GROUP BY link3.destination_url
ORDER BY COUNT(*) DESC;
```

While the syntax may be confusing to people not familiar with SQL, note the relative size of the code and the English description. We know of no other computer language where this query of the Web can be expressed more concisely.

When we ran this program with seed pages www.now.org (National Organization for Women) and www.feminist.org (The Feminist Majority Foundation), we got the following results:

URL	Count	Description
www.aauw.org	4	American Assoc. of University Women
www.aclu.org	4	American Civil Liberties Union
www.feminist.org/gateway/womenorg.html#top	4	Directory of Women's Organizations
www.cs.cmu.edu/afs/cs.cmu.edu/user/mmbt/www/women/writers.html	3	A Celebration of Women Writers
a href="http://www.cs.utk.edu/~bartley/other/ISA.html	3	Incest Survivors Anonymous
www.democrats.org	3	Democratic National Committee
www.igc.org/igc/womensnet/	3	WomensNet
www.pfaw.org	3	People for the American Way

3.2 Home Page Finder

A new type of application made necessary by the Web is a tool to find users' personal home pages, given their name and perhaps an affiliation. Like many information classification tasks, determining whether a given page is a specific person's home page is an easier problem for a person to solve than for a computer. Consequently, our ParaSite's primary strategy is not determining directly if a page "looks like" a home page but finding pages that human beings have labeled as being someone's home page. While there is no single stereotypical title for home pages, there is for the anchor text of hyperlinks to them: the author's name. This can be done in Squeal for the name "Pattie Maes" as follows:

```
// Create a table to store candidate pages
CREATE TABLE candidate (url VARCHAR(1024));

// Populate table with destinations of links with anchor text "Pattie Maes"
INSERT INTO candidate (url)
SELECT destination_url
FROM link
WHERE anchor = "Pattie Maes";
```

Observe that this takes advantage of inter-document structure. In contrast, the Ahoy! home page finder [Shakes et al 1997] generates candidates by searching for the name anywhere in a document.

Once we have the candidate pages, we can make use of intra-document information to rank them. For example, while it is promising if the name appears anywhere on the page, it is most promising if the name appears in the title field or a header field:

```
// Create a table to store ranked results
CREATE TABLE result (url VARCHAR(1024), score INT);
// Give a page 5 points if it contains the name anywhere
INSERT INTO result (url, score)
SELECT destination_url, 5
FROM candidate c, page p
WHERE p.url = c.url
AND p.contents LIKE '%Pattie Maes%';
// Give a page 10 points if it contains the name in the title
INSERT INTO result (url, score)
SELECT destination_url, 10
FROM candidate c, tag t, att a
WHERE t.url = c.url
AND t.name = "TITLE"
AND a.tag_id = t.tag_id
AND a.name = "anchor"
AND a.value LIKE '%Pattie Maes%';
```

The structure of the URL can also be used. The URL of Pattie Maes's home page is: "http://pattie.www.media.mit.edu/people/pattie/". This is easily recognized as a likely home page because:

1. The file name is the empty string. (Other stereotypical file names for home pages are "index.html" and "home.html".)
2. The final directory name is the user's email alias.
3. The penultimate directory name is "people". (Other common penultimate directory names for home pages are "home" and "homes".)

The last heuristic would be added to the code as follows:

```
// Give a page 10 points if the penultimate directory
// is "home[s]" or "people".
INSERT INTO result (url, score)
SELECT destination_url, 10
FROM candidate c, parse p
WHERE p.url_value = c.url
AND p.component = "path"
AND p.depth = 2
```

```
AND (p.value = "people" OR p.value = "homes" OR p.value = "home");
```

The Squeal code to display the resulting URLs and their total number of points is simply:

```
SELECT url, SUM(*)
FROM result
GROUP BY url ORDER BY SUM(*) DESC;
```

Other heuristics, with their Squeal encodings, are provided in [Spertus 1998]. The top results for the full version are:

url	SUM(*)
pattie.www.media.mit.edu/people/pattie/	33
lcs.www.media.mit.edu/people/pattie/	23
www.media.mit.edu/~pattie	16

We implemented a simple variant of the program encodes and returns reasons for each decision [Spertus 1998]. (The code is not shown here because we wish to emphasize what is unique about Squeal and not the power of SQL.) Some of the reasons the system provided for its top decision were:

score	reason
10	File name is the empty string
10	Penultimate directory is: "people"
2	Anchor from "http://lcs.www.media.mit.edu/groups/agents/papers.html" is the full name: "Pattie Maes"
2	Anchor from "http://nif.www.media.mit.edu/" is the full name: "Pattie Maes"
1	Anchor from "http://aries.www.media.mit.edu/people/aries/home-page.html" includes the full name: "Prof. Pattie Maes"

3.3 Moved Page Finder

Users frequently encounter "broken links" (obsolete URLs) while searching and browsing. In 1997, the Web Characterization Group found that 5-8% of file requests were for broken links [Pitkow 98]. We provide two structure-based techniques for tracking down moved pages.

Consider the following blurb, returned by HotBot (www.hotbot.com) in response to the query "Lenore Blum 1943":

Lenore Blum

Lenore Blum 1943 Written by Lisa Hayes, Class of 1998 (Agnes Scott College) Lenore Blum was a bright and artistic child who loved math, art, and music from her original introductions to them. Blum finished high school at the age of 16, after which...

http://www.scottlan.edu/lriddle/women/BLUM.HTM, 5359 bytes, 27Apr97

The goal of a moved-page finder is to find the new URL U_{new} given the information in an out-of-date blurb, i.e., the invalid URL U_{bad} and the title of the page. In this example, U_{bad} is "www.scottlan.edu/lriddle/women/BLUM.HTM", and the title is "Lenore Blum".

Technique 1: Climbing the directory hierarchy

We can create URL U_{base} by removing directory levels from U_{bad} until we obtain a valid URL. We can then crawl from U_{base} in search of a page with the given title. This is based on the intuition that someone who cared enough about the page to house it in the past is likely to at least link to the page now. In this example, the page was quickly found; its new name was "http://www.scottlan.edu/lriddle/women/blum.htm".

Technique 2: Checking with pages that referenced the old URL

People who pointed to a URL U_{bad} in the past are some of the most likely people to point to U_{new} now, either because they were informed of the page movement or took the trouble to find the new location themselves. Here is a heuristic based on that observation:

1. Find a set of pages *P* that pointed to U_{bad} at some point in the past.
2. Let *P0* be the elements of *P* that no longer point to U_{bad} anymore.
3. See if any of the pages pointed to from elements of *P0*is the page we are seeking.

A question is how to recognize when we've found the target page. We do this by looking for the known title text or letting the user specify a key phrase. The code to implement both of these techniques appears in [Spertus 1998].

4. Conclusions

4.1 Summary

Because the Web contains useful structural information, it is important to be able to make structure-based queries. We built such a system, Squeal, on top of the most popular structured query language, SQL. By making use of our schema and implementation, any person familiar with SQL can use Squeal to make powerful queries on the Web.

We described three applications that make use of the Web's structural information and showed or sketched their Squeal implementations. Our point was not to argue that these are the best possible such applications (although evaluations have been encouraging [Spertus 1998]) but to give an idea of the structure-based queries one might want to make and to show how easy it is to do so in Squeal.

4.2 Related Work

An extractor developed within the TSIMMIS project uses user-specified wrappers to convert web pages into database objects, which can then be queried [Hammer et al 1997]. Specifically, hypertext pages are treated as text, from which site-specific information (such as a table of weather information) is extracted in the form of a database object. This is in contrast to our system, where each page is converted into a set of database relations according to the same schema.

This work is influenced by WebSQL, a language that allows queries about hyperlink paths among Web pages, with limited access to the text and internal structure of pages and URLs [Mihaila 1996, Mendelzon 1997, Arocena 1997]. In the default configuration, hyperlinks are divided into three categories, internal links (within a page), local links (within a site), and global links. It is also possible to define new link types based on anchor text; for example, links with anchor text "next". All of these facilities can be implemented in our system, although WebSQL's syntax is more concise. While it is possible to access a region of a document based on text delimiters in WebSQL, one cannot do so on the basis of structure. Some queries we can express but not expressible in WebSQL are:

1. How many lists appear on a page?
2. What is the second item of each list?
3. Do any headings on a page consist of the same text as the title?

W3QL is another language for accessing the web as a database, treating web pages as the fundamental units [Konopnicki 1998]. Information one can obtain about web pages includes:

1. The hyperlink structure connecting web pages
2. The title, contents, and links on a page
3. Whether they are indices ("forms") and how to access them

For example, it is possible to request that a specific value be entered into a form and to follow all links that are returned, giving the user the titles of the pages. It is not possible for the user to specify forms in our system (or in WebSQL), access to a few search engines being hardcoded. Access to the internal structure of a page is more restricted than with our system. In W3QL, one cannot specify all hyperlinks originating within a list, for example.

An additional way in which Squeal differs from all of the other systems is in providing a data model guaranteeing that data is saved from one query to the next and (consequently) containing information about the time at which data was retrieved or interpreted. Because the data is written to a SQL database, it can be accessed by other applications. Another way our system is unique is in providing equal access to all tags and attributes, unlike WebSQL and W3QL, which can only refer to certain attributes of links and provide no access to attributes of other tags. Furthermore, Squeal is the only system that is built on genuine SQL, with the full power of that language.

Some powerful query systems have been built for Extensible Markup Language (XML), including Ozone [Lahiri et al 1998] and XML-QL [Deutsch et al 1999], both of which are influenced by relational and object-oriented databases. Because XML tags are customized and pages more structured than in SQL, these systems are able to support queries that focus more on semantics than syntax. Despite the different domain, the XML and Ozone work suggests that Squeal would benefit from support for object-oriented queries.

4.3 Status

We are in the process of creating:

- A public release of our Just-In-Time Database [Spertus and Stein 1998] implementation, written in Java.
- A pre-computed database containing the most popular sites on the Web in database form so it can be queried by any client.
- A forms-based text interface.
- A graphical user interface to make Squeal accessible to people who do not know SQL [Sengupta 1999].

For the latest information about our project, see http://parasite.mills.edu.

Acknowledgements

We were assisted in this research by Oren Etzioni, Keith Golden, Tom Knight, and Pattie Maes. We also benefited from interaction with Alberto Mendelzon's WebSQL group and with Alexa Internet. This paper was improved by comments from anonymous reviewers. Ellen Spertus is partially supported by a National Science Foundation Career Grant.

References

[Abiteboul and Vianu 1997]
Serge Abiteboul and Victor Vianu. Queries and Computation on the Web. In *The Sixth International Conference on Database Theory* (ICDT), Delphi, Greece, January 1997.

[Arocena et al 1997]
Gustavo O. Arocena, Alberto O. Mendelzon, and George A. Mihaila. Applications of a Web Query Language. In *Proceedings of the Sixth International World Wide Web Conference*, Santa Cruz, CA, April 1997.

[Dean and Henzinger 1999]
Jeffrey Dean and Monika Henzinger. Finding Related Web Pages in the World Wide Web. In *Eighth International World Wide Web Conference*. Elsevier Science B.V., May 1999.

[Deutsch et al 1999]
Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A Query Language for XML.

Eighth International Conference on the World-Wide Web. Elsevier Science B.V., May 1999.

[Hammer et al 1997]
J. Hammer, H. GarciaMolina, J. Cho, R. Aranha, and A. Crespo. Extracting semi-structured information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data,* Tucson, Arizona, May 1997.

[Kessler 1965]
M. M. Kessler. "Bibliographic Coupling between Scientific Papers." *American Documentation*, 14, 10-25.

[Kleinberg 1998]
J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[Konopnicki 1998]
David Konopnicki and Oded Shmueli. WWW Information Gathering: The W3QL Query Language and the W3QS System. *ACM Transactions on Database Systems*, September 1998.

[Lahiri et al 1998]
Tirthankar Lahiri, Serge Abiteboul, and Jennifer Widom. Ozone: Integrating Structured and Semistructured Data. Technical Report, October 1998.

[Mendelzon97]
Alberto Mendelzon, George Mihaila, and Tova Milo. Querying the World Wide Web. In *Journal of Digital Libraries* 1(1), pp. 68-88, 1997.

[Mihaila 1996]
George A. Mihaila. WebSQL — an SQLLike Query Language for the World Wide Web. Master's Thesis, University of Toronto, 1996.

[Pitkow 98]
James E. Pitkow. Summary of WWW Characterizations. World Wide Web, Vol. 2, No. 1-2, 1999.

[Resnick and Varian 1997]
Paul Resnick and Hal R. Varian. Recommender Systems (introduction to special section). *Communications of the ACM*, 40(3):56-58, March 1997.

[Rousseau 1997]
Ronald Rousseau. Sitations: an Exploratory Study. *Cybermetrics*, 1(1), 1997.

[Sengupta 1999]
Dyuti Sengupta. A Visual Interface for Internet Information Retrieval Via ParaSite. MA Thesis proposal, Department of Mathematics and Computer Science, Mills College, October 28, 1999.

[Shakes et al 1997]
Jonathan Shakes, Marc Langheinrich, and Oren Etzioni. Dynamic Reference Sifting: A case study in the homepage domain. In *Proceedings of the Sixth International World Wide Web Conference*, April 1997.

[Shardanand and Maes 1995]
Upendra Shardanand and Pattie Maes. Social information filtering: Algorithms for automating "word of mouth". In *ComputerHuman Interaction (CHI)*, 1995.

[Small 1973]
H. Small. Cocitation in the scientific literature: a new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24:265-269, 1973.

[Spertus 1997]
Ellen Spertus. ParaSite: Mining Structural Information on the Web. In *Proceedings of the Sixth International World Wide Web Conference*, April 1997.

[Spertus 1998]
Ellen Spertus. ParaSite: Mining the Structural Information on the World-Wide Web. PhD Thesis, Department of EECS, MIT, Cambridge, MA, February 1998.

[Spertus and Stein 1998]
Ellen Spertus and Lynn Andrea Stein. Just-In-Time Databases and the World-Wide Web. In *Proceedings of the Seventh International ACM Conference on Information and Knowledge Management*, November 1998.

Vitae

Ellen Spertus is an Assistant Professor in the Department of Mathematics and Computer Science at Mills College in Oakland, California. She received her S.B. (1990), S.M. (1992), and PhD (1998) in computer science from MIT, where she worked with Prof. Lynn Andrea Stein in information retrieval and Prof. William J. Dally in computer architecture. She has also worked at Microsoft Research and been a visiting scholar at the University of Washington.

Lynn Andrea Stein is an Associate Professor in the Department of Electrical Engineering and Computer Science and a member of the Artificial Intelligence Laboratory and the Laboratory for Computer Science at MIT. She received the A.B. degree in computer science from Harvard and Radcliffe Colleges in 1986 and the Sc.M. (1987) and Ph.D. (1990) from the Department of Computer Science at Brown University. Her research and teaching center on nontraditional computational architectures supporting interaction among programs and users.