Squeal: A Programming Interface to Structural Information on the Web

Ellen Spertus
Mills College
5000 MacArthur Blvd.
Oakland, CA 94613

spertus@mills.edu

Lynn Andrea Stein
MIT AI Lab
545 Technology Square
Cambridge, MA 02139

las@ai.mit.edu

Abstract

The World-Wide Web contains an abundance of semi-structured information, including hyperlinks between pages, structure within hypertext pages, and structure within the addresses of pages (URLs). Existing Web tools and applications, however, usually treat the Web as though it were a flat text collection. In earlier work, we described "Just-in-Time Databases" and the associated Squeal schema, which allow the Web's structure to be queried easily by providing the illusion that the entire Web is in a standard relational. In this paper, we describe a number of "ParaSites" built on top of Squeal to show the utility of structural information to a variety of applications and the ease with which such applications can be written. Specifically, we present Squeal implementations of a recommender system, a personal home page finder, and a moved page finder.

Introduction

The World-Wide Web contains hundreds of millions of pages of data. While many Web search tools treat the Web as though it were merely a distributed collection of flat documents, it can also be viewed as an enormous set of federated databases containing semi-structured information that can be automatically mined for knowledge discovery. Human beings browsing the Web take advantage of the following types of semi-structured information:

- structure within a page, such as its header and list structure (intra-document structure)
- hyperlinks connecting pairs of documents (inter-document structure)
- structural information in the uniform resource locators (URLs), the addresses of Web pages

Elsewhere, we have described a language, Squeal, and a technology, Just-In-Time Databases, that allow easy access to the Web's structural information [19][20]. In this paper, we discuss some of the applications that have been built on top of Squeal. Specifically, we describe a recommender system, a personal home page finder, and a moved page finder. We call our 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 ease with which structural algorithms can be implemented in Squeal and the power of even simple structure-based applications.

Background

Web

The World-Wide Web consists of pages of data, addressed *by uniform resource locators* (URLs) containing each page's host machine name and path. We focus on pages written in *hypertext markup language* (HTML). HTML includes tags and attributes to specify intra-document information, such as a page's logical structure and how it should be visually rendered. HTML also incorporates inter-document information, such as what pages are connected by hyperlinks. If page *P1* contains a hyperlink with destination *P2,* we say that *P1* points to *P2*.

Squeal and Just-in-Time Databases

The Squeal interpreter [19][20] allows Structured Query Language (SQL) queries on the

Web as though it were a relational database, with certain limitations because it is impractical (if not impossible [1]) to determine complete information about the Web. Squeal answers queries as completely as possible by querying search engines and fetching and parsing web pages in response to user queries. For example, consider the hyperlink relation, which consists of three fields:

- *source*, the URL of the page on which the hyperlink appears
- *anchor*, the text that the user selects in order to follow a link
- *destination*, the URL that is the target of a hyperlink

For example, a hyperlink from the ACM SIGIR upcoming events page to the Digital Libraries '99 page would consist of the following values:

- source: "www.acm.org/sigir/UpcomingEvents.html"
- anchor: "Digital Libraries '99"
- destination: "fox.cs.vt.edu/DL99/"

One permissible query would be:

```
SELECT destination
FROM hyperlink
WHERE
source="www.acm.org/sigir/UpcomingEvents.html"
```

The Squeal interpreter would respond by fetching the named page from the Web, parsing it, and returning a list of the destinations of the page's hyperlinks, i.e., the pages pointed to by "www.acm.org/sigir/UpcomingEvents.html". Another legal query would be:

```
SELECT source FROM hyperlink
WHERE destination="http://fox.cs.vt.edu/DL99/"
```

The Squeal interpreter would:

1. Ask a search engine (e.g., AltaVista) what pages point to the named page.
2. Fetch all of the pages returned by the search engine, verifying which ones still point to the named page.
3. Return to the user a list of the pages that point to "http://fox.cs.vt.edu/DL99/".

The Squeal interpreter thus provides the illusion that the Web is in a database that the user can query. In addition to allowing queries about hyperlinks, Squeal allows queries concerning the contents of a page, including its header and list structure; the structure of URLs, including the directory components of path names; and both verified and unverified claims by search services about the contents or hyperlinks of a page. (This description of Squeal is an over-simplification; a complete discussion can be found elsewhere [19].)

Applications

A Recommender System

One useful class of information retrieval applications is recommender systems [11], 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. A standard 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 [16], 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 is currently practiced, users explicitly rate 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 (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.

Note the similarity to *bibliometrics*, the statistical study of documents, which includes citation indexing [13]. *Co-citation* refers to when two papers are referenced by a common source [17] and is equivalent to the ParaSite's judging two web documents similar if they are both pointed to by the same page. The term "sitation" has been coined by Gerry McKiernan to describe the study of links between Web pages [12], and Jon Kleinberg has developed sophisticated algorithms for studying Web topology [6]. What is novel about this work is the ease with which heuristics can be written and tested.

Implementation

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.

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.

The Squeal code for the recommender system, including the last heuristic, is shown in Figure 1. While the syntax may be confusing to people not familiar with Squeal or SQL, note the relative size of the code and the comments. It shows that expressing an application in Squeal takes roughly the same amount of space as expressing it in English, something we also found to be true for our other applications.

```
// Create a table in which to store
// parent URLs
CREATE TABLE parent(url_id url_id,
hstruct BINARY(6), lstruct BINARY(6));

// Store the source pages of hyperlinks
// pointing to the seed page, along
// with information about where in the
// header and list structure of the
// page the link appeared (e.g., on
// list 2 under header 1).
INSERT INTO parent (url_id, hstruct,
lstruct)
SELECT DISTINCT source_url_id, hstruct,
lstruct
FROM link
WHERE dest_url_id = pageid;

// Show the destinations of the links
// specified in the "parent" table.
// List the most-frequently referenced
// pages first.
SELECT v.vcvalue, COUNT(*)
FROM link l, parent p, valstring v,
urls u
WHERE l.source_url_id = p.url_id
   AND l.dest_url_id = u.url_id
   AND u.value_id = v.value_id
   AND l.hstruct = p.hstruct
   AND l.lstruct = p.lstruct
GROUP BY v.vcvalue
HAVING COUNT(*) >= threshold
ORDER BY COUNT(*) DESC;
```

Figure 1: Squeal code for similar page finder. Lines beginning with slashes are comments.

Evaluation

To compare the structure-based and text-based approaches, we used the above ParaSite with the last heuristic and the Excite "more like this" feature. Because Excite can only find pages similar to a single page, not a set of them, we only provide a single URL to each system for each round of the test.

We had four human subjects submit a set of seed URLs that interested them. For thirteen URLs given, we provided users with the top 5 recommendations of each system, which users then rated on a scale of 0 to 4 for relevance, interestingness, and novelty. As subjects pointed out, a rating for novelty seems not to be applicable when a page was entirely irrelevant. For this reason, when "averaging" ratings, novelty was treated as zero when relevance was zero. Full details about the experiment appear elsewhere [19].

On average, the Excite pages were judged more relevant (1.84 vs. 1.36) and interesting (1.63 vs. 1.47) than the ParaSite pages, while the ParaSite pages were judged more novel (1.32 vs. 1.12). The results of the evaluation of each set of recommendations can be divided into three cases: those where all the ParaSite averages were higher (3), where the Excite averages were higher (4), and where the results are mixed (6). I discuss one example in each category.

<u>ParaSite superior: Austin weather</u>
The URLs returned by each system for the page entitled "The Weather Channel – Austin, TX" (www.weather.com/weather/us/cities/TX_Austin.html) are shown in Figure 2. Excite returned Weather Channel reports on other cities in the Southwest, while ParaSite generally returned information, not necessarily weather-related, about Austin or the whole of Texas. Three of the users, including the one who submitted the seed URL, preferred the ParaSite listings. The fourth reviewer thought that the weather information returned by Excite was more relevant. Quantitatively, the ParaSite pages were judged more relevant (1.4 vs. 95), interesting (1.8 vs. .95), and novel (1.37 vs. .25) than the Excite pages.

Description	**r**	**i**	**n**
TWC: Lamesa, TX	1	1	.25
TWC: Seminole, TX	1	1	.25
TWC: Pecos, TX	1	1	.25
TWC: Muleshoe, TX	1	1	.25
TWC: Hot Springs, AZ	.75	.75	.25
Excite averages	**.95**	**.95**	**.25**
Austin weather	2.75	2.75	2.25
Austin guide	2	2	1.75
Texas magazine	1.25	1.75	2
Jokes	.25	1.5	.25
Austin information	.75	1.25	.75
ParaSite averages	**1.40**	**1.85**	**1.40**

Figure 2: User Ratings of Austin Weather Recommendations. "The Weather Channel" is abbreviated "TWC". The letters "r", "i", and "n" stand for "relevance", "interestingness", and "novelty", respectively.

<u>Excite Superior: MapQuest</u>
The URLs returned by each system for the "MapQuest!" (www.mapquest.com) home page are shown in Figure 3. All 5 sites returned by Excite were highly relevant map-related sites. The 5 sites returned by ParaSite were all related to travel but much less directly, such as tourist information about San Diego. Excite was rated better for all measures.

Description	**r**	**i**	**n**
Geography & Maps	2.5	2.25	1.75
AOL NetFind	2.5	2.5	1.75
Maps on the Net	2.25	2	1.5
GeoSystems	2	1.5	1.5
MapQuest	1.75	1	.25
Excite averages	**2.20**	**1.85**	**1.35**
PCL Map Collection	1.75	1.75	1.125
Subway navigator	1	1.75	1.25
Xerox Map Viewer	1.25	1.75	1
San Diego information	.25	1	.25
More San Diego info	.25	.75	.25
ParaSite averages	**.90**	**1.40**	**.78**

Figure 3: User Ratings of MapQuest Recommendations

<u>Neither System Superior: Geek Site of the Day</u>
The URLs returned by each system for the "Geek Site of the Day" (www.owlnet.rice.edu/~indigo/gsotd/) are shown in Figure 4. Because ParaSite only made four recommendations, only the top four Excite recommendations are listed.

Two of the Excite recommendations were articles about GSotD, one was a review of GSotD and similar sites, and one was a GSotD archive. The ParaSite selections were more diverse: the first two were collections of cool/useless pages, the next was the home page of "CNET: The Computer Network", and the fourth was the Museum of Bad Art. The Excite pages were considered more relevant (1.94 vs. 1.71), while the ParaSite pages were considered more interesting (1.83 vs. 1.44) and novel (2.13 vs. .94). Users disagreed in their written comments as to which system was preferable:

"System A [Excite] came up with one good suggestion. System B [ParaSite] came up with several. System B [ParaSite] wins…"

"I assume the person wants sites that would be interesting or funny to the computer geek, such as things in poor taste. In this case I would choose system A [Excite]."

Description	**r**	**i**	**n**
WebCrawler review	2.25	1.75	1.25
PC Novice mention	1.5	.75	0
GSotD, Sep. 1995	2.25	1.75	.75
News Herald review	1.75	1.5	1.75
Excite averages	**1.94**	**1.44**	**.94**
Cool Site of the Day	2	2.25	2
Useless Pages	2.08	2.08	2.5
CNET.COM	1.75	1.75	2
Museum of Bad Art	1	1.25	2
ParaSite averages	**1.71**	**1.83**	**2.13**

Figure 4: User Ratings of Geek Site of the Day (GSotD) Recommendations.

Discussion

The ParaSite suggestions were judged more novel, while the Excite ratings were judged more relevant and interesting. Each system was markedly superior to the other in some cases. Some possible conclusions are:

1. The text-based approach is likelier than the structure-based approach to stay within the seed web site, yielding pages that users find more relevant but less novel.
2. Neither of the two approaches is always superior. Whether the text- or structure-based approach is better depends on the type of link and the user's purpose.
3. A superior system could be built by combining the two approaches.
4. The structure-based approach would have generated more useful results if more pages had been examined for each seed URL.

Further evaluation is planned.

At a higher level, we consider the following points highly significant:

- A nontrivial comparison between the best commercial text-based recommender system for Web pages and a simple structure-based system.
- Structure-based algorithms can be expressed quickly and simply in Squeal, allowing easy prototyping and testing.

Home Page Finder

A new type of application made necessary by the Web is a tool to find users' personal home ages, 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, 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. For example, an AltaVista search for pages containing the name "Nicholas Kushmerick" returned 27 links, none of them to his home page. In contrast, a search for hyperlinks with anchor text "Nicholas Kushmerick" returned three matches, two of which were links to the correct home page and one to his email address. This is an example of taking advantage of inter-document structure. In contrast, the Ahoy! home page finder [15] generates candidates by searching for the name anywhere in a document.

Another class of useful structural information is intra-document structure. For example, it is more significant if the name "Nicholas Kushmerick" appears in the title field of a page than in the body.

The structure of the URL can also be used. The URL of Kushmerick's home page is: "http://www.cs.washington.edu/homes/nick/". 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 "homes". (Another common penultimate directory for home pages is "people".)

Implementation

Our heuristics take advantage of:

- inter-document structure (hyperlinks)
- intra-document structure (headers)
- intra-URL structure

All of these types of structure can be queried in Squeal. A portion of the code is shown in Figure 5.

```
// Give ten points to pages named
// "index.htm[l]", "home.htm[l]", or
// <empty>.
INSERT INTO candidate (url_id, score)
FROM candidate c, urls u, parse p
WHERE u.url_id = c.url_id
AND p.url_value_id = u.value_id
AND p.depth = 1
AND (p.value LIKE 'home.htm%' OR
    p.value LIKE 'index.htm%' OR p.value
    LIKE '');

// Give 5 points to pages with the name
// in a title or header tag.
INSERT INTO candidate(url_id, score)
SELECT t.url_id, 5
FROM tag t, att a, valstring v
WHERE t.url_id IN (SELECT DISTINCT
   url_id FROM candidate)
AND (t.name='title' OR t.name LIKE
   'h_')
AND a.tag_id = t.tag_id
AND a.name = 'anchor'
AND v.value_id = a.value_id
AND v.vcvalue like fullnameExp;
```

Figure 5: Code from home page finder

Evaluation

Names were taken from David Aha's list of Machine Learning and Case-Based Reasoning Home Pages (www.aic.nrl.navy.mil/aha/people.html), which was chosen because it was used in the evaluation of Ahoy! [15]. Of the first fifteen hyperlinks appearing under the letter "A", twelve links still worked. We excluded the link for "David Aha", considering him a special case. Of the eleven remaining names, ParaSite successfully found nine home pages (82%) and failed in two cases (18%), as shown in Figure 6.

Name	# returned	# correct
Agnar Aamodt	2	2
Gennady Argre	0	0
Kamal Ali	1	1
Carolyn Allex	3	3
Lloyd Allison	5	1
Ethem Alpaydin	1	1
Rick Alterman	2	1
Klaus-Dieter Althoff	3	2 or 3
Tim Anderson	1	1
Bill Anderson	3	0
Chuck Anderson	4	1

Figure 6: Results of Home Page Finder on Names from Aha's List

In one successful case, ParaSite found a better link than the one on Aha's list: Aha's entry for Lloyd Allison pointed to a one-page profile (http://www.cs.monash.edu.au/people/profiles/lloyd.html) containing no links, while ParaSite returned a link to a more traditional home page (entitled "Lloyd Anderson - Home Page") that contained roughly two dozen links (www.cs.monash.edu.au:80/lloyd/index.html). ParaSite unequivocally failed in its search for Bill Anderson, returning pages relevant to other people with the name. In the case of Chuck Anderson, links were returned both to the intended Chuck Anderson and to others, clearly a hazard of only using names for searches.

Moved Page Finder

Search engines frequently return obsolete URLs. In 1995, Selberg and Etzioni found that 14.9% of the URLs returned by popular search engines no longer point to accessible pages [9]. With the growth and aging of the Web since their measurements, the percent of obsolete URLs returned may now be even higher. We provide two 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.

Conclusions

The three applications we have presented show that the structural information on the Web can be effectively mined. The Squeal source code for the programs, one of which was included here, show that it is possible to state useful queries concisely. We believe that we have shown that mining the Web's structural information is worthwhile and that Squeal provides a mean for easily doing so.

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 [5]. 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 [2][9][8]. 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 [7]. 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.

Future Work

Further evaluation should be done on these and similar applications, although our main research is in the Squeal programming system. We plan to make Squeal publicly available in a variety of formats:

- Our Just-In-Time Database [20] implementation.
- A user interface to make Squeal accessible to people who do not know SQL.
- A CD containing the most popular sites on the Web in database form so it can be queried by any client.

We look forward to seeing what people do with Squeal. We believe there are many possible applications of extending data mining to the Web.

References

[1] Serge Abiteboul and Victor Vianu. Queries and computation on the web. *In The Sixth International Conference on Database Theory* (ICDT), Delphi, Greece, January 1997.
[2] 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.
[3] Justin Boyan, Dayne Freitag, and Thorsten Joachims. A machine learning architecture for optimizing web search engines. In Franz [4].
[4] Alexander Franz, editor. The AAAI-96 Workshop on Internet-based Information Systems. AAAI, August 1996.
[5] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.
[6] J. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.
[7] David Knopnicki and Oded Shmueli. WWW Information Gathering : The W3QL Query Language and the W3QS System. *ACM Transactions on Database Systems*, September 1998.
[8] Alberto Mendelzon, George Mihaila, and Tova Milo. Querying the World Wide Web. In *Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems*, Miami, FL, 1996.
[9] George A. Mihaila. WebSQL — an SQL-like query language for the world wide web. Master's thesis, University of Toronto, 1996.

[10] Daniel E. O'Leary. The relationship between relevance and reliability in internet-based information and retrieval systems. In Franz [4].

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

[12] Ronald Rousseau. Sitations: an exploratory study. *Cybermetrics*, 1(1), 1997.

[13] Jacques Savoy. Citation schemes in hypertext information retrieval. In Maristella Agosti and Alan F. Smeaton, editors, *Information Retrieval and Hypertex*t, pages 99-120. Kluwer Academic Press, 1996.

[14] Erik Selberg and Oren Etzioni. Multi-service search and comparison using the metacrawler. In *Proceedings of the Fourth International World Wide Web Conference*, 1995.

[15] 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.

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

[17] H. Small. Co-citation 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.

[18] Ellen Spertus. ParaSite: Mining structural information on the web. In *Proceedings of the Sixth International World Wide Web Conference*, April 1997.

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

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