Chapter 6

Conclusions

6.1 Lessons Learned

In addition to producing several useful artifacts, most notably the Squeal interpreter, there were several conclusions reached that have broader applicability.

6.1.1 A Relational Database Model of the Web

User Interface

By providing an ontology (Chapter 2), I have shown that the relational database model is useful for representing information about the Web, at both the user and implementation levels. The power of the relatively short Squeal applications (Chapter 4) demonstrates that powerful queries on the Web can be stated concisely. We also know from the success of SQL that even more powerful statements can be constructed. A major limitation of SQL is that it cannot represent transitive closure, e.g., the question of whether a path exists from page A to page B. This limitation is well-suited to the Web, because such queries cannot necessarily be answered (as will be discussed in greater detail in section 6.2.2 below).

Of course, simplicity and expressiveness are often at odds. Some classes of queries can be expressed in a more concise and intuitive fashion in languages designed for the Web than in a more general-purpose language. For example, the query "What is in the third table on page X?" can be expressed much more naturally in WebL [19] than in SQL/Squeal.

Internal Representation

The Squeal system uses the relational database model not only in the user interface but also in its implementation (Chapter 5). This provides the user with the added benefit of being able to directly access the internal SQL tables by querying the SQL server instead of the Squeal interpreter. Although not discussed earlier in this document, it is a powerful option: Users could have the Squeal system build up the tables over the region of the Web they care about and then access them directly through SQL. This would allow the user to easily make queries such as finding pairs of pages [in the prefetched portions of the Web] point to each other.

Building the Squeal interpreter on top of a SQL server black box meant that a Squeal did not need to know how to interpret SQL statements, only how to produce them. This greatly expedited development and allowed the use of an industrial SQL server. On the other hand, there is a performance cost associated with having the Squeal-to-SQL translator (in the Squeal interpreter) separate from the SQL interpreter (in the SQL server), since the two associated optimizers are also separate.

6.1.2 Using SQL syntax to specify computation

In ordinary SQL systems, a SELECT query does not modify the database; it only returns information about the state of the database at the time the query is made. The Squeal system is novel in fetching and parsing Web pages in response to a query. This makes Squeal a far more powerful language than SQL.

Implementable

As the algorithms (Chapter 3) demonstrate, it is possible to implement this more powerful query-language semantics for certain domains, including the Web. Specifically, it can be implemented when each table has one or more defining columns (section 3.4) whose value can be used to determine entire lines in the table. In the Web domain, **url_id** was the most common defining column (Table 3.2, since it is easy to determine information about a specific page by fetching and parsing it or by using it as an input to a search engine. There is no reason this technique could not be used for other domains where one or two fields of a relation determine the values of the others.

Useful language for user

The Squeal extensions to the SQL semantics let users specify Web operations declaratively, i.e., indicating *what* information they want, not *how* it should be retrieved. Declarative programs are often easier for people to use and are more easily optimized by machines than imperative languages. While there is a learning curve associated with declarative languages, Squeal's similarity to a language as popular as SQL should improve its accessibility.

6.2 Comparisons to Related Work

6.2.1 Structure within and across web pages

Boyan et al. have observed that Web pages differ from general text in that they possess external and internal structure [5]. They use this information to propagate rewards from interesting pages to those that point to them (also done by LaMacchia [21]) and to more heavily weight words in titles, headers, etc., when considering document/keyword similarity, a technique used earlier in Lycos by Mauldin and Leavitt [22]. Mauldin has made use of link information by naming a page with the anchor text of hyperlinks to it. Iwazume et al. have preferentially expanded hyperlinks containing keywords relevant to the user's query [16], although O'Leary observes that anchor text information can be unreliable [28]. LaMacchia has implemented or proposed heuristics similar to some mentioned in this paper, such as making use of the information in directory hierarchies [21]. Frei and Stieger have discussed how knowledge of the adjacency of nodes via hyperlinks can be used to help a user navigate or find the answer to a query [11].

6.2.2 Theoretical analyses of the Web

Abiteboul and Vianu model the Web as an infinite semistructured set of objects and discuss what queries are theoretically or practically computable [2]. They conclude that first-order logic and Datalog¬ queries are problematic when they include negation, the use of which is restricted in Squeal. They also observe that some queries have possibly infinite answers, such as the set of pages referencing a given page P. For the same query, Squeal only returns the set of pages known about by a given search engine, after filtering out those that no longer link to P. The practicality is achieved at the expense of completeness.

Mendelzon and Milo consider the Web to be finite and discuss two differences between the Web and conventional databases: restricted access (e.g., one-way links) and a lack of concurrency control, which allows the Web to change during a query's computation. While they cache pages so they do not (appear to) change during queries, other concurrency problems remain. For example, while having a finite answer, a request for the set of pages reachable from a page P might never terminate,

if pages are added to the set faster than they can be read [24]. This query cannot be expressed in the Squeal core because, like SQL, it does not support recursion. It can be expressed as a recursive Squeal procedure, in which case the application would not terminate.

6.2.3 Database interfaces to the Web

An extractor developed within the TSIMMIS project uses user-specified wrappers to convert web pages into database objects, which can then be queried [14]. 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 ParaSite, where each page is converted into a set of database relations according to the same schema.

WebSQL allows queries about hyperlink paths among web pages, with limited access to the text and internal structure of web pages and URLs [4, 25, 23]. 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 ParaSite, although WebSQL's syntax is more concise. While it is possible to access a region of a document based on text delimiters, one cannot do so on the basis of structure. For example, consider this sample WebSQL definition and query:

```
DEFINE CONTEXT LEFT = HR, RIGHT = P;

 SELECT a.href
 FROM Anchor a SUCH THAT base =  "http://www.x.y.z/songs.html"
 WHERE file(a.href) CONTAINS ".wav"
 AND a.context CONTAINS "beatles";
```

For each link on page "http://www.x.y.z/songs.html" whose destination contains ".wav", the system will examine the region from the nearest "HR" tag to the left of the link to the nearest "P" tag to the right of the link, returning the destination if the region contains "beatles". This same query could be specified in ParaSite, although it would be more verbose. A query that can be stated in ParaSite but not WebSQL is: "Find all links appearing within the first top-level list on a page." This query relies on knowing not just that a link is within a list item (which can be determined by examining nearby text) but its position in the overall structure of a page; e.g., in the second list appearing within the first top-level list.

W3QL is another language for accessing the web as a database, treating web pages as the fundamental units [20]. Information one can obtain about web pages are:

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 ParaSite (or in WebSQL), access to a few search engines being hardcoded. Access to the internal structure of a page is more restricted than with ParaSite (or with WebSQL). In W3QL, one cannot even specify all hyperlinks originating within a list.

One major way in which ParaSite differs from both 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. Another way that ParaSite is unique is in providing equal access to all tags and attributes, unlike WebSQL and W3QL, which can only refer to certain attributes of the LINK tag and provide no access to attributes of other tags.

6.3 Future Work

6.3.1 Improvements to the System

The current system is a prototype and could be improved in a number of ways.

Integration into a SQL server

Squeal would be much more efficient and robust if it were integrated with a SQL database server. Currently, no guarantees are made as to atomicity, consistency, isolation, and durability. Compilation of Squeal queries would be more efficient than the current interpretation, as would be exposing Squeal queries to optimization.

Memory usage

Opportunities exist for more efficient memory usage. For example, text within nested tags is repeated in the database, instead of being referred to indirectly. A better approach might be to use offsets, which would also eliminate the redundancy of storing both the unparsed contents of a page in **page** and the parsed contents in other **tag**, **att**, and especially **vcvalue**.

Access to up-to-date information

Active database technology [15] could be used to update or invalidate pages in the database as they expire. We would also like to provide Squeal with direct access to a search engine's database, which would also minimize data transfer delays.

6.3.2 Further evaluation

It would be desirable to evaluate many structure-based heuristics, such as those that appear in this document, in depth. Some specific experiments that would be desirable are:

1. Do a larger test of the similar page finder, comparing it to both Excite and collaborative filtering.
2. Run the home page finder on a large test set. Measure not only the performance of the entire application but the effectiveness of each heuristic.
3. Measure the performance of the moved-page finder and each of its heuristics.

A broader experiment would be to make Squeal publicly available and see what people do with it. The best verification of my thesis would be for Squeal and applications built on it to be widely used.