Milestone 2: Literature Review and Design

Eliza Brock
Rose-Hulman Institute of Technology
brocker@rose-hulman.edu

1. ANALYSIS OF THE FIELD

1.1 XML Processing

The mainstream approaches to parsing and accessing XML data are DOM (Document Object Model) and SAX(Simple API for XML). DOM is a full featured in-memory representation of an XML document. DOM can be very cumbersome when working with large documents and even the best of the widely-used DOM parsers has trouble with large documents. SAX uses linear scanning to trigger SAX events as various datum are encountered.

The most interesting non-mainstream (or at least not mainstream to consumers of XML processing) of the data structures that can be used to index and store XML data for further processing are tree-to-table mapping, prefiltering, external-memory index structures and range labeling.

Prefiltering frameworks[11] can make processing large XML files more feasible by returning slices of a large XML document to a client that can then use a SAX or DOM parser to complete the query.

Range labeling is extremely convenient for processing containment queries[15].

1.2 XPath Processing

[20]

The majority of the papers that I read on XPath and XPath processing focused on a specialized subset of XPath referred to as "navigational XPath" or "core XPath" rather than the full XPath specification.

One of the most useful things I picked up was some differentiation on the various properties of XPath fragments, from Bendikt's "Structural Properties of XPath fragments"[14] which differentiates between downward vs. upward, recursive vs. nonrecursive, and qualifiedvs. non-qualified XPath fragments.

The two main optimizations that I have seen for XPath processing are query rewriting[12] and the advent of streaming algorithms[1].

Query rewriting focuses on preventing unnecessary document traversal by eliminating unnecessary steps. Perhaps the most interesting of the algorithms for query rewriting is the Xaos algorithm[1] due to its incorporation of backward-axes. There are several other algorithms that incorporate optimizations for only forward axes (e.g. XFilter, YFilter, XTrie, and TurboXPath).

Streaming algorithms work by processing the XPath expression as the document is scanned, thus bypassing the common requirement that an entire document by in memory before an expression can be evaluated.

1.2.1 XPath Satisfiability with Schemas

After reading about XPath satisfiability, I've come to the conclusion that it isn't a particularly interesting tangent to take for a pedagogical tool of this nature.

The basic approaches to satisfiability that I've read about thus far either transform queries and trees into specialized calculi [7], incorporate the use DTDs[2], or focus on XML Schemas[9].

The calculus approach has an interesting application to pedagogical tools, in that it can be used to "generate XML tree examples when the containment does not hold"[7], thus providing students with examples of *why* their query does not or may not returns any results.

2. PRELIMINARY RESEARCH RESULTS

2.1 Implementation Languages/Techniques

2.1.1 Implementation Language

I will be using Haskell to implement this research project. Haskell is a purely functional programming language that has good support for pattern matching and appears to have a large number of libraries available that are applicable to the project at hand.

2.1.2 Server/Web Framework

For this project, I need a server with support for the following usage: automatic recompilation upon source change, session state, and the AJAX paradigm. I considered the following approaches before finally settling on using HAppS[10].

If HAppS does not work out, I will try the other options available in the order below:

HAppS "Haskell Application Server" Features auto recompilation, optional state storage, and is designed for the "pattern of developing static web pages and using AJAX to populate them with dynamic content" [10]

WASH "Web Authoring System Haskell" Supports for session persistence, and has been used in a number of production web services, good support for database access.[19]

CGI libraries Simplest approach. However, using the CGI libraries directly would involve putting effort into work that isn't directly related to the project's end goals.

Others HWS, SHWF, and other lesser known web frameworks written for Haskell.

2.1.3 Javascript

I'm looking into javascript libraries in order to facilitate the usage of XMLHttpRequest and dynamic content generation. At this point I'm either going to use jQuery[16] or write all the necesary javascript by hand.

I'll be using a technique similar to the one used in the script section of the 4umi javascript code samples[**?**] to do some subquery-related highlighting.

2.1.4 HTML

I found several GPL'd HTML/javascript based org charts that look like they'll really fit the bill for displaying tree-based data. I plan on using one of R.P. Bouman's Org Chart scripts to generate much of my in-browser XML view [5, 4]. For my existing prototype, I am using version 3 of Bouman's scripts.

2.2 Data Collection

I have collected a number of XML files from various internet sources[18, 6, 17] to use while testing the project. However, few of them are properly licensed for the task at hand. Since my needs for small easy-to-understand yet somewhat complex XML documents, I will probably end up writing my own sample documents later, rather than trying to find appropriate publicly licensed samples.

2.3 Algorithms Under Consideration

2.3.1 XML

I'm particularly interested in creating a *light-weight* DOM-esque framework in conjunction with a tree-to-table mapping and range labeling.

DOM will cover direct ancestor and descendant relationships, while tree-to-table mapping with range labeling will coordinate descendant-or-self axis and some of the other more complicated axes.

The enhanced indexing in [15] and the well specified axis relationships defined therein are extremely interesting. I plan on investigating them more thoroughly, and, depending on my implementation I might choose to use Chen's indexing technique (modified to work in-memory rather than on-disk) in lieu of my original plan of using a tree-to-table mapping with range labeling.

2.3.2 XPath

The W3C has an excellent reference for parsing XPath expressions, "Building a Tokenizer for XPath of XQuery" [3], which will serve as an excellent guide for parsing strategies and writing a tokenizer for the XPath expressions.

I think that, based on the strategy I plan to take with partial parsing and monadic processing, that it may be possible to have a sort of pause-return-continue loop within the processing of XPath expressions, such that, at the end of each evaluation of a subexpression, the parser pauses to return the set of nodes that have been selected, and then waits for the input of an additional subexpression- at which point it will continue parsing from the point at which it left off, without repeating any unnecessary effort.

Streaming algorithms [1] sound really interesting. However, given the slightly-stateful, mostly-synchronous nature of the project's XPath processing I'm not sure that streaming algorithms work in this context.

I'm certain that the streaming algorithms that work with backward axes wouldn't work with the "sub-expression at a time" approach that I plan on taking. However, it would be very interesting to use forward-axis only streaming processing.

The concept of streaming processing might be better saved for future work on the project; perhaps with the inclusion of a feature in which the user can manually specify very large documents. Given the current restrictions I have in place for the sake of in-browser usability, the gains from streaming parsing would be negligible.

2.4 Updated Timeline

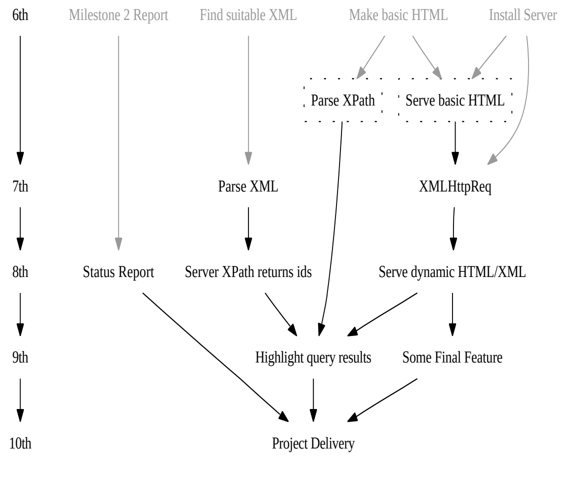

Figure 1: Schedule (Grey is completed, Dotted is in progress)

3. REFERENCES

[1] C. Barton, P. Charles, D. Goyal, M. Raghavachari, M. Fontoura, and V. Josifovski. Streaming XPath processing with forward and backward axes. *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 455–466, 5-8 March 2003.

[2] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *PODS '05: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 25–36, New York, NY, USA, 2005. ACM.

[3] S. Boag. Building a tokenizer for XPath or XQuery. Technical report, W3C, April 2005.

[4] R. Bouman. Dutch cabinet, April 2008. `http://xcdsql.org/Misc/orgchart/03/OrgChart.html`.

[5] R. Bouman. HTML organization chart, April 2008. `http://xcdsql.org/Misc/orgchart/02/orgchart.html`.

[6] R. Data. XML examples, April 2008. `http://www.w3schools.com/XML/xml_examples.asp`.

[7] P. Genevès and N. Layaïda. A system for the static analysis of XPath. *ACM Trans. Inf. Syst.*, 24(4):475–502, 2006.

[8] G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *PODS '03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 179–190, New York, NY, USA, 2003. ACM.

[9] J. Groppe and S. Groppe. Filtering unsatisfiable XPath queries. *Data I& Knowledge Engineering*, 64(1):134–169, January 2008.

[10] HAppS LLC. HAppS – the haskell application server, April 2008. `http://www.happs.org/`.

[11] C.-H. Huang, T.-R. Chuang, and H.-M. Lee. Prefiltering techniques for efficient XML document processing. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering*, pages 149–158, New York, NY, USA, 2005. ACM.

[12] C. Koch. Processing queries on tree-structured data efficiently. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 213–224, New York, NY, USA, 2006. ACM.

[13] M. Marx and M. de Rijke. Semantic characterizations of navigational XPath. *SIGMOD Rec.*, 34(2):41–46, 2005.

[14] W. F. Michael Benedikt and G. Kuper. Structural properties of XPath fragments. *Theoretical Computer Science*, 336(1):3–31, May 2005.

[15] K. O. Qun Chen, Andrew Lim and J. Tang. Indexing XML documents for XPath query processing in external memory. *Data & Knowledge Engineering*, 59(3):681–699, Decemeber 2006.

[16] J. Resig. jQuery; the write less, do more, javascript library, April 2008. `http://jquery.com/`.

[17] M. Technologies. Sample XML file (books.xml), April 2008. `http://msdn2.microsoft.com/en-us/library/ms762271(VS.85).aspx`.

[18] M. Technologies. Schematron xml sample documents, April 2008. `http://www.mulberrytech.com/papers/schematron-xml2004-hrefs.html`.

[19] P. Thiemann. Web authoring system haskell (WASH), April 2008. `http://www.informatik.uni-freiburg.de/~thiemann/haskell/WASH/`.

[20] W3C. *XML Path Language (XPath)*, 1.0 edition, November 1999.

[21] W3C. *Extensible Markup Language (XML) 1.0*, 4.0 edition, September 2006.