Visualizing Structural Patterns in Web Collections

M. S. Ali
University of Toronto
sali@cs.toronto.edu

Mariano P. Consens
University of Toronto
consens@cs.toronto.edu

Flavio Rizzolo
University of Toronto
flavio@cs.toronto.edu

Categories and Subject Descriptors

H.2.1 [**Information Systems**]: Database Management—*Schema and subschema*

General Terms

Design, Measurement

Keywords

XML, Structural Summaries, Visualization, XPath, RSS

1. VISUALIZING SUMMARIES

We present DescribeX, a tool for exploring and visualizing the structural patterns present in collections of XML documents. DescribeX can be employed by developers to interactively discover those XPath expressions that will actually return elements present in a collection of XML files.

The element structure of many collections of XML documents present in the Web can be fairly unpredictable. This is the case even when the documents are validated by a schema, and can happen for two main reasons. First, the documents may follow a schema that allows many elements to occur almost anywhere in the document (e.g., by extensive use of `<xsd:choice>` in XML schema). Second, the default namespace and corresponding schema can be extended by incorporating elements from other namespaces with corresponding schemas (e.g., by using the `<xsd:any>` XML Schema construct to allow open content models).

A collection of RSS files provides a good example of the situations described above. The RSS schema is fairly permissive of where the basic elements occur. Multiple feed schemas can be present in the collection because of multiple versions (*e.g.* RSS 0.91 or 2.0) or formats (*e.g.* Atom, or RSS 1.0/RDF). Finally, RSS encourages the use of extensions so elements from several namespaces will be present (*e.g.* Dublin core, iTunes podcast, Microsoft Simple List Extensions). Other web collections for which the XML element structure can be fairly unpredictable are traces generated by web services requests and also XML-ized versions of wikis (and Wikipedia, in particular).

A major challenge in working with these kinds of web collections is to understand enough about their structure to be able to pose meaningful queries employing XPath patterns.

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

Structural *summaries* are labeled graphs that describe the structural patterns present in document instances. Summary nodes are associated with partitions (called extents) formed by document elements that are indistinguishable with respect to a simple pattern query (*i.e.*, elements with the same path to the root, or elements with the same leafs, or both). Summaries were introduced to help understand the structure present in semistructured data collections. They are now among the most studied techniques for query evaluation in XML (and other semistructured) data models and a variety of summaries have been proposed.

All of the existing proposals (as well as new ones) can be captured in DescribeX by using a novel framework that employs XPath expressions for defining the summary extents. An early version of the DescribeX framework and its application to XML Retrieval is described in [1] (this citation references an extensive list of earlier summary proposals).

The next section discusses how a developer can use DescribeX to explore multiple summaries to discover specific XPath expressions that actually occur in the collection. The last section presents features of DescribeX that allow a developer to handle large and complex summary graphs.

2. EXPLORING FEEDS

Consider a collection of sample feeds to be be aggregated. RSS feeds are composed of channels (*i.e.*, genre) and items (*i.e.*, articles in genre). The challenge is to efficiently find feed items and understand how markup has been employed to contextualize the information contained in feed items. A developer may be interested in grouping together articles that have similar meta-data structures (*e.g.*, articles that have known creation dates and authorship).

In trying to isolate the relevant elements for aggregating feed items, the developer must find which paths in a collection of RSS feeds would be best suited to finding groups of elements across different schemas. RSS feeds are comprised of multiple schemas, with multiple versions and they contain literally thousands of possible distinct root to leaf tag paths.

Our example collection consists of a sample of feeds encoded in the Atom, RSS/RDF and RSS formats. In Figure 1, we show results for *incoming* summary graphs of our collection. These graphs cluster together nodes that have the same incoming label paths. For example, in Figure 1, there is an `item` node with id `36` (center graph) that represents all `item` elements that are children of `RDF` roots (that is, an `RDF/item` incoming label path). In contrast, `item 55` (right graph) represents all `item`'s that are within `channel`'s that

are within `rss` roots (i.e, an `rss/channels/item` incoming label path).

In aggregating feeds, a developer is interested in items that have similar substructures. Even when considering a relatively simple graph, as shown in Figure 1, the developer potentially needs to write an XPath query for every possible combination in the summary graph of children or descendants of a feed item (`item` or `entry` elements). The problem with this approach is that the number of such XPath queries is usually quite large. For instance, if we consider only one level down in our Figure 1 example, it is easy to see that an Atom `entry` (left graph) may have any combination of 9 child elements (or $2^9 = 512$ possible queries). When considering descendants rather than children, those numbers are considerably larger. In addition, most of the XPath queries written this way will usually return empty answers because only a few of the combinations actually appear in the documents.

DescribeX can provide a more descriptive view of the collection to help developers find the XPath queries that capture the paths that are present in the documents. This is achieved by *refining* summary nodes. Consider now Figure 2 where `item 55` has been split into five different `item` nodes that correspond to different subtree patterns (the nodes in green are new summary nodes that resulted from the refinement). In this new summary graph, `item 55` now represents `item` elements that have `comments`, `category`, `description`, `pubDate`, `link`, and `guid` subelements whereas `item 96` represents `item` elements with a `description`, an `encoded` and a `title`. This refinement allows the developer to understand not only where `item`'s appear in the documents, but also what kind of subelements they have, thus simplifying the writing of the XPath expressions mentioned above.

For instance, in the refined trees of Figure 2 we find that, out of the 512 possible combinations of child elements for Atom `entry`'s, only 3 are actually present (indicated with nodes circled in red in the left graph). This drastically reduces the number of XPath expressions to be considered and helps to understand how markup has been employed to contextualize the information contained in feed items.

3. COVERAGE IN LARGE SUMMARIES

One of the challenges in visualizing summary graphs is how to ensure that a user is not overwhelmed by either the size or complexity of the graph. For example, in visualizing the summary of the RSS feeds, it was noted that it contains multiple schemas and possibly thousands of distinct structural paths (*e.g.*, our full RSS test corpus contains 3961 paths based on a heterogeneous collection of 58,338 files).

DescribeX supports interactively controlling how to view

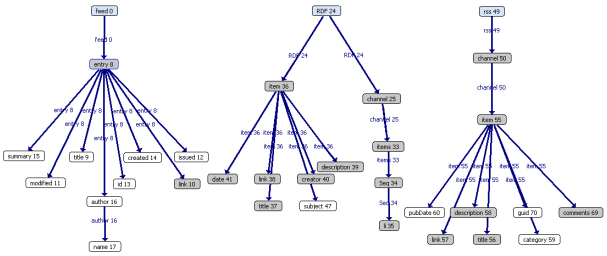

Figure 1: Summarizing Collected Feeds

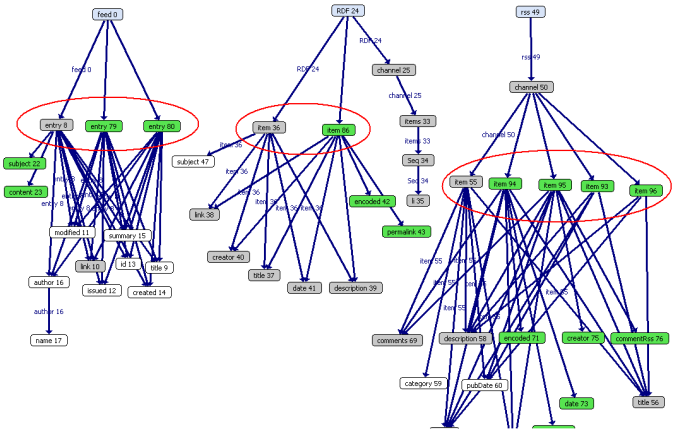

Figure 2: Summary Refinement for `item` and `entry`

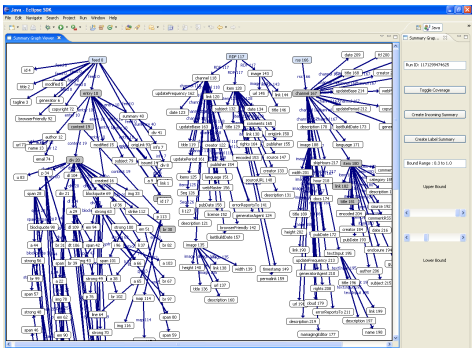

Figure 3: Collected Feeds at 100% coverage

only the most popular labels used in the collection, *i.e.* the elements with the most *coverage*. In its simplest form, the nodes with the largest extents are displayed as coverage approaches 0%, and progressively smaller extents become visible as the coverage approaches 100% (at this point, all summary nodes would be displayed). DescribeX supports both specifying an upper and a lower bound for coverage. So, in order to restrict the view to only the most common substructures, the user would select a range of low values for coverage (these are the grayed nodes in Figures 1, 2 and 3). Conversely, for a user who is interested in finding uncommon substructures, the user would select a range of high values for coverage. DescribeX also supports additional (more involved) notions of coverage.

Figure 3 shows a typical incoming path summary of RSS feeds at 100% coverage. In our previous discussions, figures 1 and 2 were based on a graph similar to figure 3 at a coverage of 85%. For most web collections, modest changes in coverage from 100% can have significant changes in the size of the displayed graph. For instance, in our full RSS test collection, a coverage of 85% (resp. 99%) of the most common extents of the incoming paths resulted in a reduction in graph size from 3961 nodes to 144 nodes (resp. 740 nodes).

4. REFERENCES

[1] M. S. Ali, M. P. Consens, X. Gu, Y. Kanza, F. Rizzolo, and R. Stasiu. Efficient, effective and flexible XML retrieval using summaries. In *Proc. of the 5th Intl Workshop of the Initiative for the Evaluation of XML Retrieval (INEX)*, 2007.