Visualizing XML trees and node sets selected by XPath expressions

Michael Piotrowski ⟨*mxp@iws.cs.uni-magdeburg.de*⟩

Otto-von-Guericke-Universität Magdeburg · FIN/IWS

2005-08-04

Abstract This report describes an approach to visualizing XML documents and node sets selected by XPath expressions. The system is based on XSLT and Lout and produces publication-quality PostScript output. We also describe educational uses of these visualizations.

1 Introduction

Since the publication of the final specification in 1999, XSLT [8] has quickly become the standard way to transform XML documents into other XML documents or into non-XML formats. For selecting elements for processing, for conditional processing and for generating text XSLT makes use of the XPath expression language [7].

XPath and XSLT model an XML document as a tree of nodes. There are different types of nodes, including element nodes, attribute nodes and text nodes. The most important kind of expression is the *location path*. A location path selects a set of nodes relative to a context node. The result of evaluating a location path expression is the *node set* containing the nodes selected by the location path. Location paths can recursively contain expressions that are used to filter sets of nodes.

The following listing—an excerpt from a larger stylesheet—is an example of the use of XPath expressions in XSLT:

```
<xsl:template match="para[preceding-sibling::*[1]/self::para
                          or preceding-sibling::*[1]/self::formalpara
                          or preceding-sibling::*[1]/self::*[@float = 1]]">
  @PP {<xsl:value-of select="ancestor-or-self::*[@lang][1]/@lang"/>} @Language
  {<xsl:apply-templates/>}
</xsl:template>
```

This template will be applicable to all nodes which match the XPath expression specified in the `match` attribute, namely `para` elements which fulfill certain conditions.

Our experience with teaching XSLT and XPath to university students has shown that students often have trouble both with finding out which nodes exactly are selected by a given XPath expression in some XML document, as well as with writing XPath expressions to select a specific set of nodes in an XML document.

But it can also be hard for more experienced XSLT programmers to

determine which parts exactly of an XML tree an XPath [7] expression is going to match, especially if the expressions are getting complex.

To be able to use XPath expressions effectively, it is necessary that students are able to form a correct mental model of the tree so that they can predict the results of the expressions.

This problem is similar to constructing hierarchical list structures in Lisp and accessing them using combinations of `car` and `cdr`. To help students understand these structures, a visualization technique called *box-and-pointer notation* (cf. Figure 1) is typically used.

Thus, for educational purposes, as well as for debugging and illustrative purposes, it would be useful to be able to

a) visualize the document tree, and
b) have the nodes selected by an XPath expression highlighted in the tree.

There are some tools available that highlight the matching elements for an XPath expression in the textual form of an XML document. There is also a program, *XPath Visualizer* [9] by Alexandre Fayolle, which uses a graphical tree representation. XPath Visualizer allows the evaluation and visualization of XPath expressions in an XML document. The XML document is displayed as a tree. In the tree, one can select the XPath context node. After entering an XPath expression, XPath Visualizer evaluates it and displays the result node set as a tree and highlights the selected nodes in the original XML document tree.

However, XPath Visualizer is written in Java, so that it runs only on the relatively small number of platforms for which a Java Virtual Machine is available; in fact the author wasn't even able to get it to run on a Solaris machine which *had* a JVM installed. The author was therefore looking for a more portable solution; lower hardware requirements and less interdependencies would also be desirable.

Also, the author was not so much interested in interactive exploration, but rather in static renderings that could be included in printed or PDF teaching materials. These renderings should have publication quality. Since XPath Visualizer was designed for interactive use, it doesn't provide this feature.

2 Approach

2.1 Technology

To achieve the goals outlined above, we have taken the following approach: An XSLT [8] stylesheet, which takes an XPath expression as a parameter, generates source code for the Lout [4][2] typesetting system.

There are a number of factors which made us choose Lout over SVG [6], which might be considered a more "natural" choice in the context of XML. For example, it may not be too difficult to describe trees in SVG (cf. [3]), but Lout offers specialized high-level constructs for trees. Furthermore, outputting SVG descriptions is not enough: Software to render the descriptions into formats usable for printing and inclusion would still be needed; in contrast to Lout, which is a proven system, SVG tools currently do not appear to have reached the same level of maturity.

XSLT processors, written in a variety of languages, are available for a large number of platforms. An XSLT 1.0 stylesheet can therefore be considered

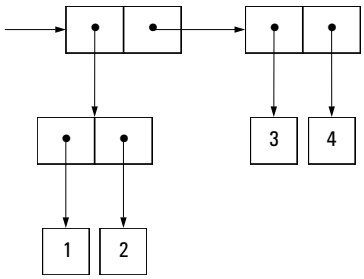

Figure 1 | Box-and-pointer representation of `(cons (cons 1 2) (cons 3 4))`.

quite portable. The author prefers the ***xsltproc*** XSLT processor from the libxslt package [5], which is written in C.

Lout is also very portable and runs on practically all UNIX and UNIX-like systems, as well as on Microsoft Windows. Lout is a very suitable target for this application since it provides high-level functions for diagrams, and it can output EPS (Encapsulated PostScript) [1] files, which can then be included directly into most document preparation systems or converted to other formats.

2.2 Usage

For example, suppose we want to generate a tree visualization of the following simple XML document:

```
<?xml version="1.0" encoding="iso-8859-1"?>

<a>
 <b>
  <c/><d/>
 </b>
 <e/>

 <f><g><h/><i/><j/></g><k><l><m/></l></k></f>
</a>
```

A typical command line on a UNIX system, using the ***xsltproc*** XSLT processor could be:

```
$ xsltproc --param xpath "/" treemark.xsl input.xml | lout -EPS - > tree.eps
```

This command line produces an EPS file; if PDF output were desired, one could directly pipe the output of Lout through ***ps2pdf*** (a part of the Ghostscript package). The generated tree is shown in Figure 2. Notice that no nodes are highlighted since we have given the XPath expression "`/`", which selects the root of the XPath tree, not the root element of the document. As our rendering shows only the elements of the documents, no visible node is selected. This type of output can be used for illustrations and for assignments where students are supposed to indicate the result nodes of XPath expressions.

If an XPath expression is given which selects nodes below the root, these nodes are highlighted. We will use a larger XML document for the next example; Figure 3 is the output for the XPath expression "`//Title | //Author[1]/Surname`", which selects the title and the last name of the first author of all articles.

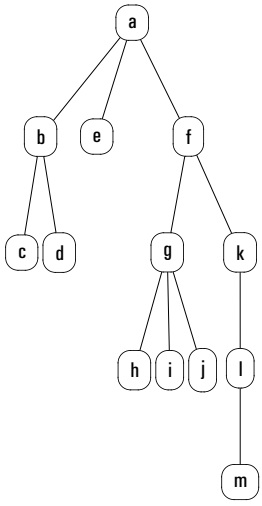

Figure 2 | Tree representation of an XML document

The command line to produce an EPS file would be:

```
$ xsltproc --param xpath "//Title | //Author[1]/Surname" treemark.xsl input.xml | \
    lout -EPS - > tree.eps
```

Figure 3 also illustrates how attributes and text nodes are rendered.

3 | Experience

The author used the stylesheet in the preparation of teaching material for a course on document processing in the winter semester 2004/2005. The tree renderings were used in two types of exercises; in both cases, the students were given graphical trees of XML documents without highlighted nodes:

1. In the first type of exercises, the students were asked to write XPath expressions to select various node sets, which were described in natural language, e.g., "the authors of the first article", "the conferences where the author with the last name of 'Suciu' has published" or "the articles referring to the article with the ID 'A2'". [1]
2. In the second type of exercises, the students were asked to indicate—in the tree—the nodes matched by certain XPath expressions, e.g., "`//Proceedings[count(Article/Author) > 1]`".

In both cases the node highlighting feature was used to prepare the solutions to the exercises.

The author's experience with the stylesheet is very good: It helped to quickly design and test exercises related to XML and XPath, and to produce esthetically pleasing diagrams for inclusion in exercise sheets and for classroom presentations. While we are quite sure that the visualization was also helpful for students, we do not have done any experiments to prove this hypothesis.

[1] These examples refer to the document shown in Figure 3

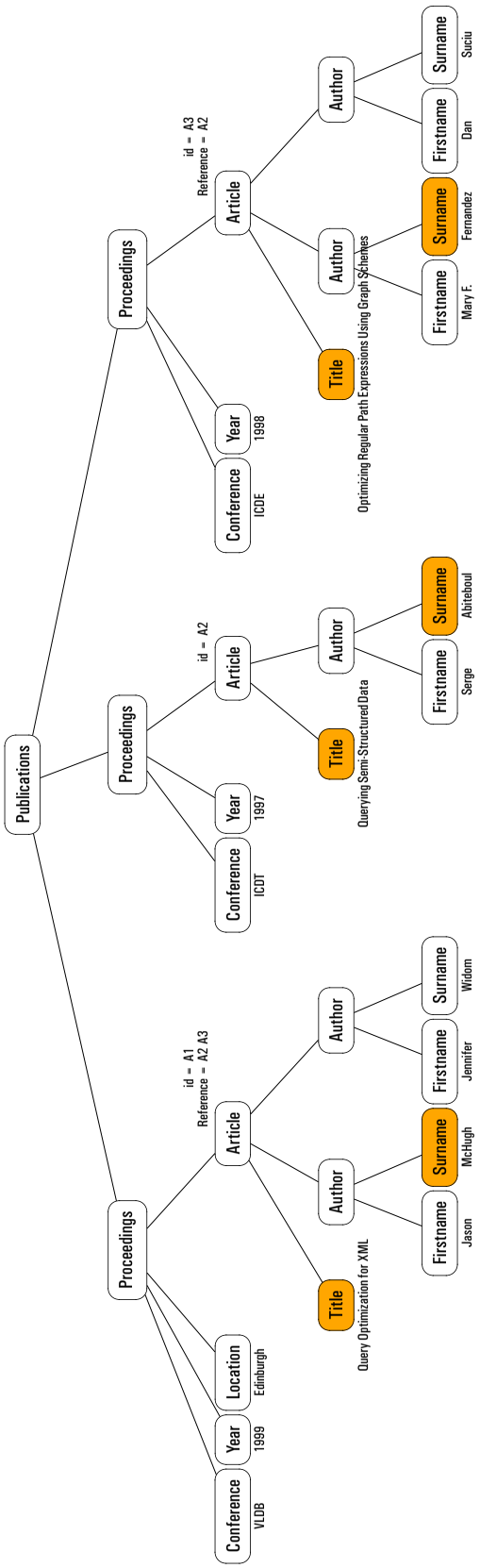

Figure 3 | Results for the XPath expression `//Title | //Author[1]/Surname`

4 Conclusion and Outlook

We have presented an approach to visualizing XML documents and node sets selected by XPath expressions. The system is based on XSLT and Lout and produces publication-quality PostScript output. We have also described the educational use of these visualizations.

The graphical trees and the node set highlighting could also be used in other types of exercises:

- The stylesheet could be used to prepare solutions for exercises where students are asked to draw a tree corresponding to an XML document.
- Students could be given a graphical tree and asked to produce the corresponding textual XML representation.
- Students could be given a graphical tree with highlighted nodes and asked to write an XPath expression which selects these nodes.

As noted above, the main purpose of the system is the production of static, publication-quality diagrams. Since the trees soon get too large to fit on a standard page, the stylesheet is mainly useful for visualizing relatively small example documents with only short text nodes. However, in educational applications this does not typically pose a problem, since the primary goal is the illustration of the relevant *concepts*, and most concepts can be shown in relatively small examples.

For some applications one might want to be able to separately specify the context node and an XPath expression to be evaluated in that context. For example, referring to Figure 2, one might want to ask:

> Element `g` is the context node. Which nodes are selected by the expression "`..`"?

Currently the context node cannot is hardcoded to "`/`". However, the same effect can be achieved by specifying an XPath expression like "`//g/..`".

The visualization system was definitely helpful for the author in teaching about XPath and we hope that it also helps the students in learning about this subject; it will again be used in the next semester.

The stylesheet described in this report is available from the author on request.

References

[1] Adobe Systems Incorporated. Encapsulated PostScript File Format Specification. Technical Note #5002 (1 May 1992). Version 3.0.

[2] Jeffrey H. Kingston. The design and implementation of the Lout document formatting language. *Software—Practice & Experience* **23(9)**, 1001–1041 (1993).

[3] Jirka Kosek. Automated Tree Drawing: XSLT and SVG, September 08, 2004. URL *http://xml.com/pub/a/2004/09/08/tree.html*.

[4] Lout Homepage. URL *http://lout.sf.net/*.

[5] Daniel Veillard. Libxslt: The XSLT C library for Gnome. URL *http://xmlsoft.org/XSLT/*.

[6] World Wide Web Consortium. Scalable Vector Graphics (SVG). W3C Recommendation (14 January 2003). Version 1.1. URL *http://www.w3.org/TR/SVG11/*.

[7] World Wide Web Consortium. XSL Transformations (XSLT). W3C Recommendation (16 November 1999). Version 1.0. URL *http://www.w3.org/TR/xpath*.

[8] World Wide Web Consortium. XSL Transformations (XSLT). W3C Recommendation (16 November 1999). Version 1.0. URL *http://www.w3.org/TR/xslt*.

[9] XPath Visualizer Homepage. URL *http://logilab.org/projects/xpathvis/*.