XPlainer: Explaining XPath within Eclipse

John W.S. Liu*
IBM Canada Inc.
Toronto Lab
jwsliu@ca.ibm.com

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

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

ABSTRACT

The popularity of XML has motivated the development of novel XML processing tools many of which embed the XPath language for XML querying, transformation, constraint specification, etc. XPath developers (as well as less technical users) have access to commercial tools to help them use the language effectively. Example tools include debuggers that return the result of XPath subexpressions visualized in the context of the input XML document.

This paper describes XPlainer-Eclipse, a novel kind of query understanding and debugging tool that provides visual explanations of *why* XPath expressions return a specific answer. XPlainer-Eclipse combines editors for visualizing both XML documents and XPath expressions as trees together with the explanation of the answers.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments—*Programmer workbench, Integrated environments*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Code inspections and walkthroughs, Debugging aids*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*User interfaces*

General Terms

Languages, Program Analysis

Keywords

XPath, Debugging, Eclipse, IDEs, XML, Data Visualization

1. INTRODUCTION

The widespread adoption of XML has motivated the development of new languages and tools geared toward XML processing. XPath [22], the most ubiquitous of XML-related languages, is used as a sub-language for tasks like XML querying, transformation, constraint specification, web service composition, etc.

*This work was completed when this author was a graduate student at Computer Science Department of the University of Toronto.

The use of XPath expressions in a large variety of computer languages (such as XSLT, XQuery, XForms, BPEL, Schema, XJ, SQL extensions, etc.) motivates the interest of a large population of developers in learning how to use the language. Providing explanations of *why* XPath expressions return specific answers is a compelling approach to facilitate understanding and debugging XPath applications.

A large number of software tools have been developed to try to help XPath users understand the evaluation of query expressions in the language. These initiatives include open source projects such as LOGILab XPath Visualizer [16], XPE XPath Explorer [6] and Visual XPath [14] and commercial products such as Altova XMLSpy XPath Analyzer [5], Top XML XPath Visualizer [4], WSAD XPath Expression [11] and Oxygen XML Editor [3].

This paper introduces the concept of explanation queries. Given an XPath expression, we define a new language, XPlainer, that relies on visual explanations to describe *why* the given XPath expression returns a sequence of selected nodes from an input XML document. The explanation provided displays all the intermediate nodes that contribute to the result of the XPath expression. While this is an intuitive notion, we show that providing explanations for arbitrarily complex expressions while supporting all the constructs in the XPath language is a non-trivial task. This difficulty justifies defining a new language (XPlainer) with the same syntax as the original target language (XPath) been explained, but whose semantics precisely state what additional information is returned to explain a query result in the target language.

General motivation for our visualization approach originates in the work of Edward Tufte [20], who states:

> *Visual explanations* is about *pictures of verbs*, the representation of mechanism and motion, of process and dynamics, of causes and effects, of explanation and narrative.

We describe a visual explanation tool based on the XPlainer language that assists XML developers to learn, understand, use, and debug XPath expressions. The explanations tell what nodes in an input XML document are *matched* by the sub-expressions, providing a representation of the basic mechanism at play during XPath processing.

The XPlainer-Eclipse tool that we have developed extends the XML and XPath development facilities available in the Eclipse environment [1] with the ability to support explanation queries. Eclipse is an open source platform built by an open community of tool providers. A large variety of both commercial and open source development tools have been integrated within this environment.

The XPlainer-Eclipse tool is capable of invoking an arbitrary XPath processor to implement the semantics of the XPlainer language. Using this approach, the tool can provide explanations that are faithful to the XPath processor invoked while **supporting all the language constructs and functions in the XPath processor**. We also address successfully the challenge of reproducing the behavior of implementation dependent features. In debugging scenarios, this ability to invoke the original XPath engine is a crucial requirement.

Programming language debuggers support stepping through the state of an execution while inspecting variable values. When dealing with XPath (or any other a declarative/functional query language), stepping through an execution is only acceptable for debugging the internal implementation of the execution engine. Any other debugging of XPath expressions should not depend on the execution path chosen by the engine's optimizer, instead it should help understanding the semantics of the expression at a logical level (and, to the extent that is possible, regardless of the engine used). Existing debugging tools achieve this by providing the result of an expression with no additional information. XPlainer goes much further by giving all the intermediate information (still at the logical level).

There is a fine line between debugging a declarative language by giving all the logical intermediate information (including implementation dependent intermediate results!), and debugging the engine of such a language by stepping through the actual execution path. XPlainer fully supports declarative debugging of an arbitrary XPath engine without crossing that line.

1.1 Related Work

There is a rich literature in graphical query languages, starting with QBE [23] in 1977. A research effort over 10 years old refers to the existence of more than fifty different visual languages for databases [21]. A more recent proposal that specifically targets visual XML queries appears in [8]. Beyond visual queries, combining data visualization with visual queries has been pursued by [15, 17].

The work we propose does not attempt to introduce a new visual query language, instead it utilizes visualizations as a mechanism to provide explanations for the semantics of an existing textual query language (XPath). The visualization of answers and intermediate results supported by XPlainer can certainly be used as a data visualization mechanism, but that is not the goal addressed in this paper.

The explanation mechanism introduced in this paper has been inspired in earlier work on graph-based data visualization [9]. The Hy+ system employed the concept of *GraphLog filter queries* that return all the intermediate tuples obtained by a Datalog logic program, and they can be (loosely) seen as the analogous of an *explanation query* for a Datalog program. The Hy+ system did not use filters to explain the answers to Datalog queries, they were used instead to create graph-based visualizations of database facts. We can apply the XPlainer language concepts described in this work to a rule based language (and not just to a functional language like XPath).

The XPath debugging tools mentioned earlier limit themselves to showing the selected nodes of an XPath expression (i.e., the result of the evaluation) either in a separate view [5, 14, 4, 16], or in the context of an existing XML editor [11, 3, 16]. All the tools currently available simply display the result of the XPath expression[1]. The state of the art tools **do not display the intermediate nodes** selected by the subexpressions that contribute to the answer. Therefore, they do not provide information about relationships among subexpressions, contexts and/or selected nodes. These are all novel capabilities provided by the XPlainer language introduced in this paper and that current tools do not posses.

Finally, we note that the application of the XPlainer language as a tool to select the subset of an XML document that contributes to an XPath answer is similar to the XSquirrel [18] *subtree queries*. These type of queries have been shown useful for defining document views, in access control applications, and in actively distributing XML documents [7]. XPlainer queries provide much more control over the nodes that are retained compared to subtree queries. XSquirrel queries retain the nodes selected by the original XPath expression together with all their ancestors and descendants, while XPlainer *explanation queries* retain all those intermediate nodes that contribute to the original XPath result (these can include nodes that are not ancestors nor descendants of the answer).

1.2 Contributions

The following are the key contributions of our work:

- We introduce the novel concept of *explanation queries* and describe XPlainer, an explanation query language for XPath.
- We provide a formal definition for the semantics of XPlainer that in turn builds upon the semantics of XPath expressions (which is a very desirable property in debugging applications)[2].
- We describe a tool that implements visual explanations based on XPlainer.

While XPlainer specifically targets XPath, the concept of explanation queries that can assist developers in learning, understanding, using, and debugging query expressions has general validity (beyond the specific case of XPath). Also, explanation queries are a convenient mechanism to extract the subset of a database that contributes to a query expression, and as such it can be used as a powerful and concise sub-document filtering mechanism.

1.3 Organization

The paper is structured as follows. In the next section we introduce the concept of explaining query expressions. We provide examples to motivate explanations and highlight the differences from simple partial evaluation of subexpressions. In Section 3 we provide an overview of a tool based on the XPlainer language (together with a description of its implementation). We conclude in Section 4.

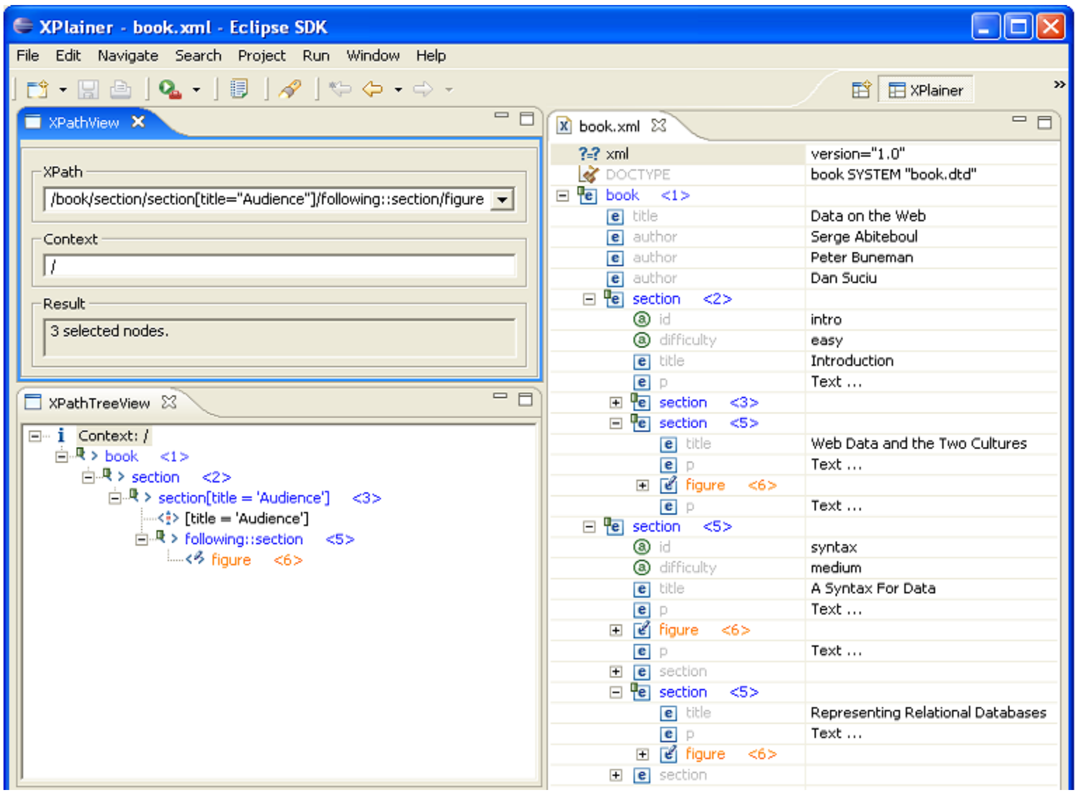
2. VISUAL EXPLANATIONS

This section provides a glimpse of the capabilities of our approach to visual explanations. We assume that the reader has a basic understanding of the XPath query language constructs.

Given an XPath query and an input XML document, an explanation of the query gives as answer all the XPath result nodes together with intermediate nodes. The intermediate nodes are those nodes resulting from the partial evaluation of the subexpressions of the original XPath query that contribute to the answer. Obtaining the explanation of a complex XPath query can be challenging, as shown in the following example.

[1]For an online sample image from XMLSpy, one of the most popular and complete tools currently available, see `http://www.altova.com/features_xpath.html`.

[2]The semantics definition is omitted in this paper due to the paper size limit, interested reader may refer to our technical report at `http://www.cs.toronto.edu/~flavio/xplainer.pdf`


XPlainer - book.xml - Eclipse SDK
File Edit Navigate Search Project Run Window Help
XPlainer
XPathView
book.xml
XPathTreeView

XPath
/book/section/section[title="Audience"]/following::section/figure

Context
/

Result
3 selected nodes.

```
Context: /
  book <1>
    section <2>
      section[title = 'Audience'] <3>
        [title = 'Audience']
        following::section <5>
          figure <6>
```

xml	version="1.0"
DOCTYPE	book SYSTEM "book.dtd"
book <1>	
title	Data on the Web
author	Serge Abiteboul
author	Peter Buneman
author	Dan Suciu
section <2>	
id	intro
difficulty	easy
title	Introduction
p	Text ...
section <3>	
section <5>	
title	Web Data and the Two Cultures
p	Text ...
figure <6>	
p	Text ...
section <5>	
id	syntax
difficulty	medium
title	A Syntax For Data
p	Text ...
figure <6>	
p	Text ...
section	
section <5>	
title	Representing Relational Databases
p	Text ...
figure <6>	
section	

Figure 1: Explanation of query q_1

EXAMPLE 2.1. Consider the query

$$q_1 = /book/section/section[title = \text{“}Audience\text{”}]$$
$$/following :: section/figure$$

which returns the figures in sections that appear after a section with $title = \text{“}Audience\text{”}$ which is itself within a section of book. An explanation of q_1 is depicted in Figure 1.

The first view in the top left corner of the figure, XPathView, is an XPath expression editor with two input fields and one message field, where the user can enter an XPath expression (query q_1 in the example).

The second input field in the XPathView indicates that the context is the root of the XML document (/, but in general this could be any other XPath expression). The message field displays the number of nodes in the result of the expression.

The second view on the left (just below XPathView) is XPathTreeView and displays a specific parse tree representation of the XPath expression that appears in XPathView. It appears as an intuitive representation of the structure of the XPath expression. In particular, the steps in the XPath expression are represented as separate nodes and they are labeled with a sequence number ($\langle 1\rangle$, $\langle 2\rangle$, $\langle 3\rangle$, $\langle 5\rangle$, and $\langle 6\rangle$ in the figure).

The XPathTreeView is paired with the XML editor view on the right side of the image in Figure 1 that displays a *book* document. Nodes in the XML editor and their corresponding leaf step in the XPathTreeView have the same step number labels and the same colours: orange for the nodes in the answer and blue for the intermediate nodes.

Since current XPath query evaluation tools do not provide explanations, the only available debugging techniques involve either partial evaluation of subexpressions or evaluating reversed axis. A partial evaluation cannot see beyond the current evaluation step, so it has no way of filtering out nodes that will have no effect in the final answer. For instance, a partial evaluation of the $/book/section/section$ subexpression would return the two top-level sections of the document, which includes one that is not an intermediate node (the second one). The same happens with a partial evaluation of $following :: section$, which also returns a superset of the intermediate nodes: all sections after section $\langle 3\rangle$, including those that do not contain figures. In contrast, an explanation of the query would include only those sections that satisfy the rest of the query, which is the one labeled by $\langle 2\rangle$ for $/book/section/section$ and those labeled by $\langle 5\rangle$ for $following :: section$. Although a partial evaluation sometimes does provide exactly the intermediate nodes (like the partial evaluation of $section[title = \text{“}Audience\text{”}]$), in general it just returns a *superset* of the intermediate nodes.

An evaluation that reverses the axis will not necessarily give us exactly the intermediate nodes either. For instance, evaluating the last reversed subexpression entails obtaining the $parent$ of all figure nodes in the answer. (Remember that $figure$ is $child ::$ $figure$ in the unabbreviated syntax, and its reverse axis is $parent$). This evaluation gives us correctly the three intermediate $section$ $\langle 5\rangle$ nodes that appear in the Figure. However, the reversed evaluation of the next subexpression, $following :: section$ will return all the $preceding$ sections, when in fact the only intermediate node at that point is section $\langle 3\rangle$.

We have shown with the previous example that we cannot rely on either partial evaluation or in evaluations that simply reverse the axes to obtain the intermediate nodes and explain *why* an XPath expression returns a specific answer.

The increasing difficulty of providing these explanations for arbitrary XPath queries motivated us to formally define the *semantics of explanations*. For instance, in the case of a more complex query like

$$q_2 = /book/((section[1]/section)[2] \mid (section[3]/section)[4]/figure))[5]/title$$

which includes parenthesis and a disjunction, obtaining the intermediate nodes without such semantics becomes extremely challenging.

XPlainer is the proposed new language for addressing the problem discussed above: an explanation of a given XPath query is computed as the result of an XPlainer query. XPlainer query answers are structured into *XPlainer trees* whose nodes correspond to subexpressions and are associated with precisely the intermediate nodes that contribute to the answer of the query being explained.

3. AN XPLAINER-BASED TOOL

The goal of an XPlainer-based tool is to provide a concrete implementation for the visual explanation approach described in the previous sections. The visual information presented by the XPlainer language describes the correspondence from the (parse) tree display of a query to the query's output and its explanation layered on a (document) tree display of the input.

This approach can be supported by another quote from Edward Tufte's work [19]:

> Amongst the most powerful devices for reducing noise and enriching the content of displays is the technique of layering and separation, visually stratifying various aspects of the data.

Our XPlainer-based tool layers on top of the input XML document annotations that explain the semantics of evaluating a given XPath query on the document. There is a conscious decision to limit the highlighting to a minimal use of color *labels* to distinguish the context, the selected nodes (the result of the query), and the intermediate nodes (the ones providing the explanation of the query result). In addition, numerical labels describe the association between XPath subexpressions and the intermediate nodes selected by them.

The XPlainer-Eclipse tool that we have developed extends the XML and XPath development facilities available in the Eclipse environment [1] with the ability to support explanation queries. Eclipse is an open source platform built by an open community of tool providers. A large variety of both commercial and open source development tools have been integrated within this environment.

While the most prominent programming language supported by Eclipse is Java (and indeed the framework itself is implemented as an extensive Java library), tools have been developed to support a variety of programming environments. Of interest to XPlainer-Eclipse are tools that support XML-related development, since most of them support XPath as an embedded language. Several of these tools have been incorporated within Eclipse, most notably around the Web Tools Platform project (WTP) [2].

In particular, there is an XML editor in Eclipse WTP that displays text and tree views of XML documents. XPlainer-Eclipse is an Eclipse plugin that uses the WTP XML editor to highlight the XPath path nodes and the XPath selected nodes directly in the XML tree.

Let us illustrate the XPlainer-Eclipse visual capabilities with an additional example. Consider the expression

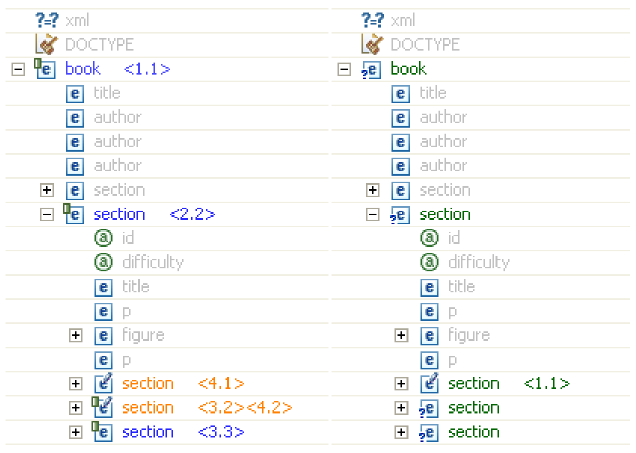
$$book/section[2]/section/preceding\text{-}sibling :: section[1]$$

This example query selects the first preceding sibling section of each child section of the second section from an XML document describing a book.

Now consider a similar expression which contains parenthesis

$$(book/section[2]/section/preceding\text{-}sibling :: section[1])$$

The explanation of the evaluation of the latter expression on the book XML document appears in Figure 2 (right-hand side) together with the explanation of the former (left-hand side). The parenthesis impact whether document order or axis order (reverse document order in this case) applies to the result of the parenthesized expression, and before the position predicate is applied.



```
xml                              xml
DOCTYPE                          DOCTYPE
book  <1.1>                      book
  title                            title
  author                           author
  author                           author
  author                           author
  section                          section
  section  <2.2>                   section
    id                               id
    difficulty                       difficulty
    title                            title
    p                                p
    figure                           figure
    p                                p
    section  <4.1>                   section  <1.1>
    section  <3.2><4.2>              section
    section  <3.3>                   section
```

Figure 2: Explaining the impact of parenthesis

XPlainer-Eclipse can explain how a predicate expression chooses a particular set of XML nodes. When the user clicks on a predicate node in the XPathTreeView, the XML nodes that make the predicate expression true will be selected and colored in green (a careful reader may also notice that the icon for predicate nodes is a small question mark). In this example we can also observe that XPlainer-Eclipse selectively expands those nodes that are highlighted in the XML editor, while leaving other nodes collapsed. This effectively filters those portions of the XML document that are irrelevant to the visual explanation. XPlainer-Eclipse has a number of additional interactive features not covered here that are part of our prototype [3] (presented also as a demo [10]). As an example, XPlainer-Eclipse users can selectively collapse (simply by clicking) portions of the XPath expression to eliminate constraints (this is useful when there are no answers to a query but the collapsed subexpression can be satisfied).

Throughout this section we have illustrated how XPlainer-Eclipse helps XPath developers understand, debug and correct the expressions that they are interested in evaluating against an example XML document. The tool achieves this goal by highlighting on the input XML document the context, the selected nodes, and the intermediate nodes that contribute to the final result (as specified in the semantics of the XPlainer language).

3.1 Implementation

The XPlainer-Eclipse tool is implemented in Java. There are two major components in the system, Core and UI, each one of them consisting of several packages with over 50 Java classes in total[13].

[3]available at http://www.cs.toronto.edu/∼consens/xplainer/

The Core component is a Java application that can be made available as a package independently of the Eclipse environment. The Core component of XPlainer-Eclipse consists of classes implementing the $\mathcal{X}_e$ tree data structure, an XPath parser, the implementation of the visual explanation function, and an XPath evaluator module. The Core component supports two key functions. First, it provides an implementation of the $\mathcal{X}_e$ tree and the $V_{T,c}$ function described in the preceding section. Second, it manages the invocation of an external XPath processor.

A very important property of XPlainer-Eclipse is that it is not tied to a particular XPath implementation. Instead an arbitrary XPath evaluator can be invoked through a standard interface and used to evaluate the $V_{T,c}$ visual explanation function. This is a critical engineering decision that allows the XPlainer-Eclipse framework to be used to provide explanations for different XPath engines. This is important because, beyond differences in the capabilities of the implementations, the XPath language itself has several areas where the semantics are implementation defined. This effectively means that only the original XPath engine can explain one of its own implementation defined features.

The main XPath engine utilized by XPlainer-Eclipse is the default XPath engine used in the Java API for XML Processing (JAXP) 1.3 [12], which is already included into J2SE 5.0 (i.e., the Java standard edition). Also, note the XPlainer-Eclipse Core components communicates with the XPath engine using the DOM as the XML data model. This module is fairly generic and can be extended to implement other internal representations of the XML data model. The UI component is developed as an Eclipse plugin.

4. CONCLUSIONS

This paper introduces XPlainer, a language defined with the novel goal of assisting users to understand and develop XPath expressions. XPlainer relies on visual explanations that describe *why* an XPath expression returns a sequence of selected nodes from an input XML document. The explanation provided displays all the intermediate nodes that contribute to the result of the XPath expression.

The paper describes an Eclipse-based tool implementing XPlainer which extends the XML and XPath development facilities available in the Eclipse environment [1]. The users of XPlainer-Eclipse can interact with tree views of XPath expressions and input XML documents. In the XML editor view, intermediate XPath expression results are selectively highlighted, connecting these nodes with the associated steps in the XPath expression.

A very important property of the XPlainer-Eclipse implementation is that it does not re-implement an XPath-like query processor. The tool relies on the semantic definition of the XPlainer language in terms of the XPath language itself to evaluate XPlainer queries by invoking an arbitrary (already existing) XPath processor. While this approach to evaluate XPlainer queries incurs obvious overhead, it has the crucial advantage of been able to *explain faithfully the evaluation of* **all** *the features of any XPath processor* invoked (even implementation-dependent features!).

Finally, we bring attention to the fact that the visualization of answers and intermediate results supported by XPlainer can be used as a powerful sub-document filtering mechanism. The semantics of the visual explanation function provides an effective and concise filtering mechanism. A large subset of the nodes in the input document can be identified by one compact XPath expression when interpreted as an XPlainer expression (i.e., one that returns a sub-document with not just the selected nodes, but all of the intermediate nodes as well). When using the language as a filter mechanism there is a clear motivation for developing XPlainer-specific optimization techniques.

5. REFERENCES

[1] Eclipse. http://www.eclipse.org/.
[2] Eclipse Web Tools Platform (WTP) Project. http://www.eclipse.org/webtools/.
[3] Oxygen XML Editor. http://www.oxygenxml.com/.
[4] TopXML. http://www.topxml.com/xpathvisualizer.
[5] XML Spy. http://www.altova.com/.
[6] XPE. http://www.purpletech.com/xpe/index.jsp.
[7] S. Abiteboul, A. Bonifati, G. Cobena, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *SIGMOD '03*, pages 527–538, 2003.
[8] D. Braga, A. Campi, and S. Ceri. XQBE (XQuery By Example): A visual interface to the standard XML query language. *ACM TODS*, 30(2):398–443, 2005.
[9] M. Consens and A. Mendelzon. Hy+: a Hygraph-based query and visualization system. In *SIGMOD '93*, pages 511–516, 1993.
[10] M. P. Consens, J. W. Liu, and B. O'Farrell. XPlainer: An XPath debugging framework (demo), 2006. http://icde06.cc.gatech.edu/prog-demo.html.
[11] IBM. WebSphere Studio Application Developer (WSAD) 5.1 XPath Expression. http://www-306.ibm.com/software/awdtools/studioappdev/.
[12] JAXP. Java API for XML Processing (JAXP) 1.3. http://java.sun.com/webservices/jaxp/index.jsp.
[13] John W.S. Liu. XPlainer: A Visual XPath Debugging Framework. Master's thesis. http://www.cs.toronto.edu/DCS/Grad/Theses/05-06MSc.html.
[14] N. Leghari. Visual XPath. http://weblogs.asp.net/nleghari/articles/27951.aspx.
[15] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. DEVise: integrated querying and visual exploration of large datasets. In *SIGMOD '97*, pages 301–312, 1997.
[16] Logilab.org. Logilab - XPath Visualizer 1.0. http://www.logilab.org/projects/xpathvis/1.0.
[17] C. Olston, A. Woodruff, A. Aiken, M. Chu, V. Ercegovac, M. Lin, M. Spalding, and M. Stonebraker. DataSplash. In *SIGMOD '98*, pages 550–552, 1998.
[18] A. Sahuguet and B. Alexe. Sub-document queries over XML with XSQuirrel. In *WWW '05*, pages 268–277, 2005.
[19] E. R. Tufte. *Envisioning information*. Graphics Press, Cheshire, CT, USA, 1990.
[20] E. R. Tufte. *Visual explanations: images and quantities, evidence and narrative*. Graphics Press, Cheshire, CT, USA, 1997.
[21] K. Vadaparty, Y. A. Aslandogan, and G. Ozsoyoglu. Towards a unified visual database access. In *SIGMOD '93*, pages 357–366, 1993.
[22] W3C. XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20, 2005.
[23] M. Zloof. Query-by-example: A data base language. *IBM Syst. J.*, 16(4):324–343, 1977.