XPlainer-Eclipse: Explaining XPath within Eclipse™ *

John W.S. Liu†
IBM Canada Ltd., Toronto
jwsliu@ca.ibm.com

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

Flavio Rizzolo
University of Toronto, DCS
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 provides a glimpse of the functionality of 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

XML is an important practical paradigm in information technology with a broad range of applications. Its many uses include being a platform-independent markup language for web development, and a common media to facilitate data exchange and integration. The widespread adoption of XML has motivated the development of new languages and tools geared toward XML processing.

Since XML is used as a data medium and carrier, accessing, retrieving and presenting XML data is crucial in most of these applications. As the most ubiquitous XML query language, XPath [19] is used as a sublanguage for tasks like XML querying and transformation, constraint specification, web service composition, etc. This widespread use of XPath expressions in a large variety of computer languages (such as XSLT, XQuery, XForms, BPEL, XML Schema, XJ, and SQL extensions) motivates an increasingly large population of developers to learn how to use the language effectively. In this context, 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 are developed to help XPath developer understand how query expressions are evaluated. Example tools include debuggers that return the result of XPath subexpressions visualized in the context of the input XML document. These initiatives include both open source projects and commercial products.

But these art tools typically simply display the result of the XPath expression; they 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.

This paper introduces the novel idea of explanation queries for XPath. We define a new language, XPlainer[**?**], that relies on visual explanations to describe *why* a given XPath expression returns a sequence of 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.

Based on this idea, we implement XPlainer-Eclipse, a novel kind of query understanding and debugging tool that provides visual explanations to assist XML developers to learn, understand, use, and debug XPath expressions. The explanations tell what nodes in an input XML document are *matched* by the subexpressions, providing a representation of the basic mechanism at play during XPath processing.

XPlainer-Eclipse is implemented on top of the Eclipse Web Tools Platform (WTP) Project [2] 1.5, and combines editors for visualizing both XML documents and XPath expressions as trees together with the explanation of the answers. The XPlainer-Eclipse tool extends the XML and XPath development facilities available

†This work was completed while the author was a graduate student at the Department of Computer Science at the University of Toronto. The opinions expressed in this article are those of the authors and do not necessarily represent the views of IBM Corporation and/or any of its affiliates.

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.

1.1 Related Work

There is a rich literature in graphical query languages, starting with QBE [20] in 1977. A 1993 research effort notes the existence of more than 50 visual languages for databases [18]. A more recent proposal that specifically targets visual XML queries appears in [8]. The idea of combining data visualization with visual queries has been pursued by [15, 17].

The explanation mechanism introduced in this paper was inspired by 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; these tuples can be seen as loosely analogous to 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. The XPlainer language concepts described in this work can also be applied to a rule-based language (not just to a functional language like XPath).

The current XPath debugging tools limit themselves to showing the result nodes of an XPath expression (i.e., the result of the evaluation) either in a separate view, or in the context of an existing XML editor. These 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.

1.2 Contributions

The key contributions of our work are as follows:

- We introduce a novel approach to *explanation queries* and describe XPlainer, an explanation query language for XPath.
- We describe a tool XPlainer-Eclipse that implements visual explanations based on Eclipse WTP 1.5.

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 subdocument 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 a 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 present conclusions 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 provides 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.

EXAMPLE 2.1. Consider the query

$$q_1 = /books/book/metadata[price > 100]$$
$$//subject[. = "Introduction"]/following :: subject$$

which returns the subjects following the "Introduction" subject in a book whose price is greater than 100. An explanation of q_1 is depicted in Figure 1.

Since current XPath query evaluation tools do not provide explanations, the only available debugging techniques involve either partial evaluation of subexpressions or evaluating reversed axes. A partial evaluation cannot see beyond the current evaluation step, so it has no way of filtering out nodes that will have no effect on the final answer. For instance, a partial evaluation of the $/books/book$ subexpression would return all books under the books element, which includes two books that are not intermediate nodes for q_1.

An evaluation that reverses the axes will not necessarily give us exactly the intermediate nodes either. For instance, the intermediate nodes of the second last step can be evaluated by a query reversing the axis $following$ to $preceding$: the XPath query is given by q_2.

$$q_2 = /books/book/metadata[price > 100]$$
$$//subject[. = "Introduction"]/following :: subject$$
$$/preceding :: subject[. = "Introduction"] \quad (1)$$

This evaluation will return all the $preceding$ subjects that are equal to "Introduction", which include a subject that is not an intermediate node for q_1 (the one under the first book node).

The previous example shows that to obtain the intermediate nodes and explain *why* an XPath expression returns a specific answer, we cannot rely on either partial evaluations or evaluations that simply reverse the axes.

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. XPLAINER-ECLIPSE TOOL

The goal of an XPlainer-Eclipse 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.

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.

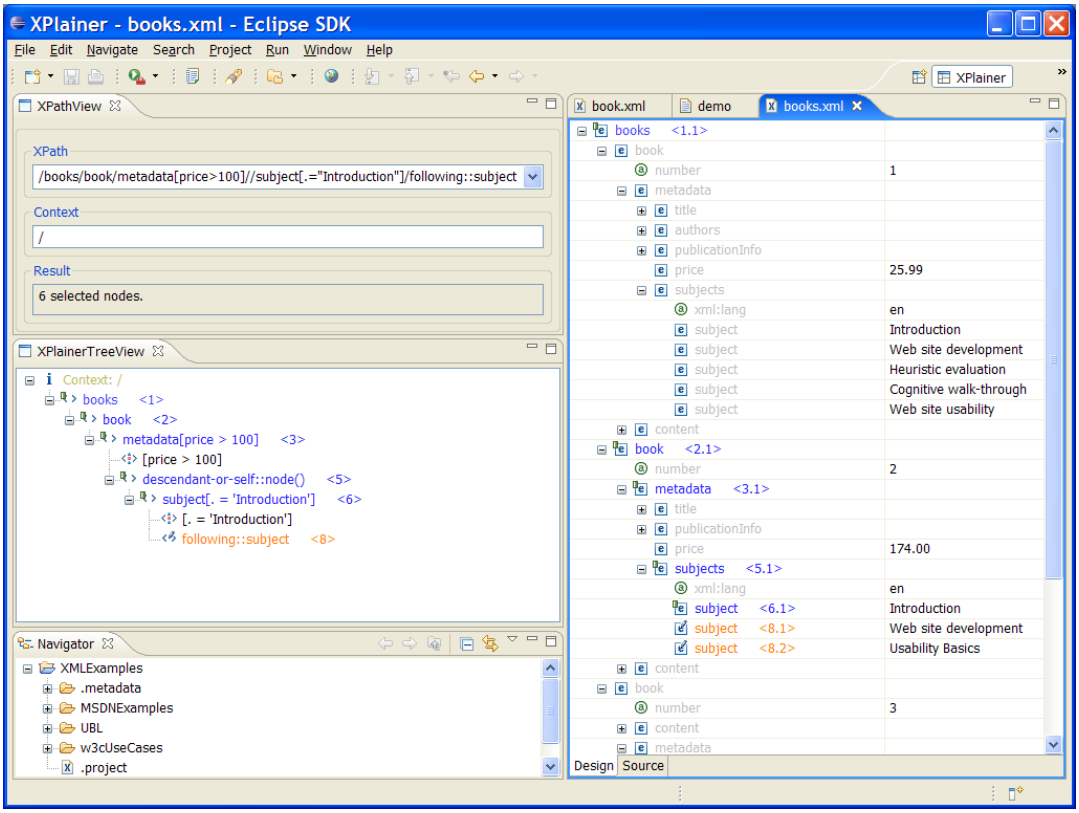

Figure 1: Explanation of query q_1

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. 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, the XPlainer-Eclipse perspective (a perspective in Eclipse is a set of views in the workbench) shown in Figure 1 consists of the four views described below.

The first view in the top left corner, XPathView, is an XPath expression editor with two input fields and one message field, where the user can enter an XPath expression. The first input field has the expression of query q_1 from Example 2.1.

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 selected nodeset (the result of the expression), or the actual result when it is not a nodeset (i.e., the result consists of string values from the XML document).

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. In particular, the steps in the XPath expression are represented as separate nodes and they are labeled with a sequence number (in the example in the figure the sequence of steps is labeled $\langle 1 \rangle$, $\langle 2 \rangle$, and $\langle 3 \rangle$, etc.).

The XPathTreeView is paired with the XML editor view on the right side of the image in Figure 1 that displays a specific instance of the *books* document. These two views are displayed shoulder-to-shoulder, so the user can better visualize the connection between the steps in the XPath expression and the highlighted nodes in the XML editor view. XPlainer-Eclipse uses the same color (orange) to label the selected nodes in the XML editor and the corresponding leaf step in the XPathTreeView (labeled $following :: subject$ in the example), in addition to having the same sequence number label $\langle 8 \rangle$. XPlainer-Eclipse also uses the same color (blue) to label the intermediate nodes that contribute to the final answer in the XML editor and the corresponding steps in the XPathTreeView, in addition to having the same sequence number labels $\langle 1 \rangle$, $\langle 2 \rangle$, and $\langle 3 \rangle$, etc.). There is another way to make the connection between the XPath steps and their associated XML nodes. We mark every XML node that is associated with an XPath step with a small icon on its image tag. We differentiate the selected nodes from the intermediate nodes with different icons.

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 that contains parentheses:

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

The explanation of the evaluation of the latter expression on the book XML document appears in Figure 2 (right side) with the ex-

planation of the former (left side). The parentheses impact whether document order or axis order (reverse document order in this case) applies to the result of the parenthesized expression before the position predicate is applied.

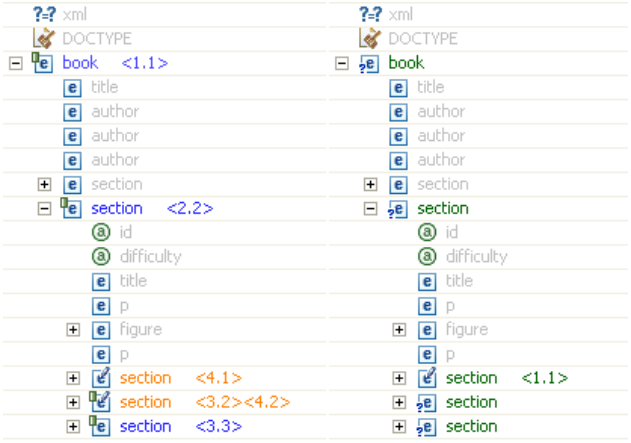

Figure 2: Explaining the impact of parentheses

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 [1] (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 supports 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].

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, a XPlainer tree and the $V_{T,c}$, a visual explanation function. Second, it manages the invocation of an external XPath processor.

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

The overall XPlainer architecture is shown in Figure 3. The XPath input view is the interface to input the XPath expression to evaluate. The input XPath is in String format and will be first parsed into a JXPath [3] parsing tree, a data structure to model XPath for evaluation. Then the XTT factory (a $\mathcal{X}_e$ tree factory) will transform the XPath in the JXPath parsing tree model into $\mathcal{X}_e$ tree model. Meanwhile the XML file is modeled as DOM and stored in the main memory by the Eclipse WTP [2]. The XML DOM is input into the XPath evaluator, then the XPath evaluator will evaluate the XPath expression in $\mathcal{X}_e$ tree to obtain the final selected nodes and all intermediate step nodes. The result will be stored in the $\mathcal{X}_e$ tree. Finally the XML DOM and $\mathcal{X}_e$ tree is displayed in the XML tree view and XPath tree view after highlighting the final selected results nodes and all intermediate step nodes.

The XPath tree view is also interactive when the user clicks on the predicate expression or the user collapses and expands the XPath tree view. The XPath tree view can also manipulate the $\mathcal{X}_e$ tree and evaluate the new XPath traversing expressions by the XPath evaluator and then send the new highlighted nodes to the XML tree view and XPath tree view.

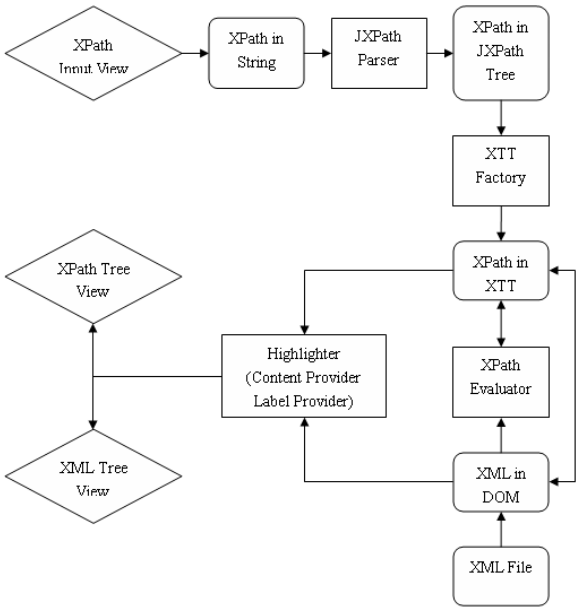

Figure 3: XPlainer-Eclipse Architecture

The XPlainer-Eclipse tool is capable of invoking an arbitrary XPath processor to implement the semantics of the XPlainer language. 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. Using this approach, the tool can provide explanations that are faithful to the invoked XPath processor while *supporting all the language constructs and functions in the XPath processor*. 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.

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.

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 in J2SE 5.0 (i.e., the Java standard edition). Note also that the XPlainer-Eclipse Core component 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 plug-in.

4. CONCLUSIONS

The paper describes an Eclipse-based tool implementing XPlainer, a language defined with the novel goal of assisting users to understand and develop XPath expressions. The XPlainer-Eclipse tool provides a visual explanation to display all the intermediate nodes that contribute to the result of the XPath expression and 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 being able to *explain faithfully the evaluation of* **all** *the features of any XPath processor* invoked.

Finally, we bring attention to the fact that the visualization of answers and intermediate results supported by XPlainer can be used as a powerful subdocument filtering mechanism. The semantics of the visual explanation function provide 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 subdocument 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.

Acknowledgments

Portions of this work were carried out in collaboration with Dr. Bill O'Farrell and Julie Waterhouse at the IBM Centre for Advanced Studies. We would also like to thank Dr. Arthur Ryman, Craig Salter and Ella Belisario from the Eclipse WTP working group for their technical support and invaluable suggestions.

Financial support was provided by an Eclipse Innovation Award, an IBM Centre for Advanced Studies (CAS) Fellowship, and the Natural Sciences and Engineering Research Council (NSERC).

5. REFERENCES

[1] Eclipse. http://www.eclipse.org/.

[2] Eclipse Web Tools Platform (WTP) Project. http://www.eclipse.org/webtools/.

[3] JXPath. http://jakarta.apache.org/commons/jxpath.

[4] 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.

[5] M. Consens and A. Mendelzon. Hy+: a Hygraph-based query and visualization system. In *SIGMOD '93*, pages 511–516, 1993.

[6] 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.

[7] M. P. Consens, J. W. Liu, and F. Rizzolo. XPlainer: Visual explanations of XPath queries. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007) Istanbul, Turkey*. IEEE Computer Society, 2007.

[8] JAXP. Java API for XML Processing (JAXP) 1.3. http://java.sun.com/webservices/jaxp/index.jsp.

[9] John W.S. Liu. XPlainer: A Visual XPath Debugging Framework. Master's thesis. http://www.cs.toronto.edu/DCS/Grad/Theses/05-06MSc.html.

[10] 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.

[11] 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.

[12] K. Vadaparty, Y. A. Aslandogan, and G. Ozsoyoglu. Towards a unified visual database access. In *SIGMOD '93*, pages 357–366, 1993.

[13] W3C. XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20, 2005.

[14] M. Zloof. Query-by-example: A data base language. *IBM Syst. J.*, 16(4):324–343, 1977.