XPlainer: Visual Explanations of XPath Queries

Mariano P. Consens John W. S. Liu Flavio Rizzolo
University of Toronto
{consens,wliu,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 introduces XPlainer, a language that provides explanations of why *XPath expressions return a specific answer. An explanation returns precisely the nodes in the input XML document that contribute to the answer. We provide a complete formalization for explanation queries based on the semantics of XPath. This enables the use of XPath engines for the evaluation of explanation queries.*

We describe a tool that uses XPlainer queries to provide visual explanations. The XPlainer-Eclipse tool is built on an extensible development environment that includes editors for visualizing both XML documents and XPath expressions as trees together with the explanation of the answers.

1 Introduction

The widespread adoption of XML has motivated the development of new languages and tools geared toward XML processing. XPath [26], 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.

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 [19], XPE XPath Explorer [6] and Visual XPath [17] and commercial products such as Altova XMLSpy XPath Analyzer [5], Top XML XPath Visualizer [4], WSAD XPath Expression [14] 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) being explained, but whose semantics 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 [24], 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 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 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 [29] in 1977. A research effort over 10 years old refers to the existence of more than fifty different visual languages for databases [25]. A more recent proposal that specifically targets visual XML queries appears in [8]. Combining data visualization with visual queries has been pursued by [18, 21]. Beyond visual queries, XVIZ [20] displays the structure of XPath expressions extracted from an XQuery workload, relating expression to each other and identifying common subexpressions .

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 [10]. 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).

Explanations are also closely related to work in *data provenance* that characterizes those tuples in a database D that "contribute" to the answer of a query over D [28, 12, 9]. Research in data provenance highlights that while the notion of finding tuples that "contribute" to an answer is intuitively appealing, actually formalizing this notion for a broad class of queries is quite challenging. The definition of explanations in this paper provides a novel answer to this challenge for the complete XPath language.

The XPath debugging tools mentioned earlier limit themselves to showing the nodes selected by a query (i.e., the result of the evaluation) either in a separate view [5, 17, 4, 19], or in the context of an existing XML editor [14, 3, 19]. They simply display the result of the query without the intermediate nodes that contribute to the result. 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 [22] *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).
- We describe a tool, XPlainer-Eclipse[1], that implements visual explanations.

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

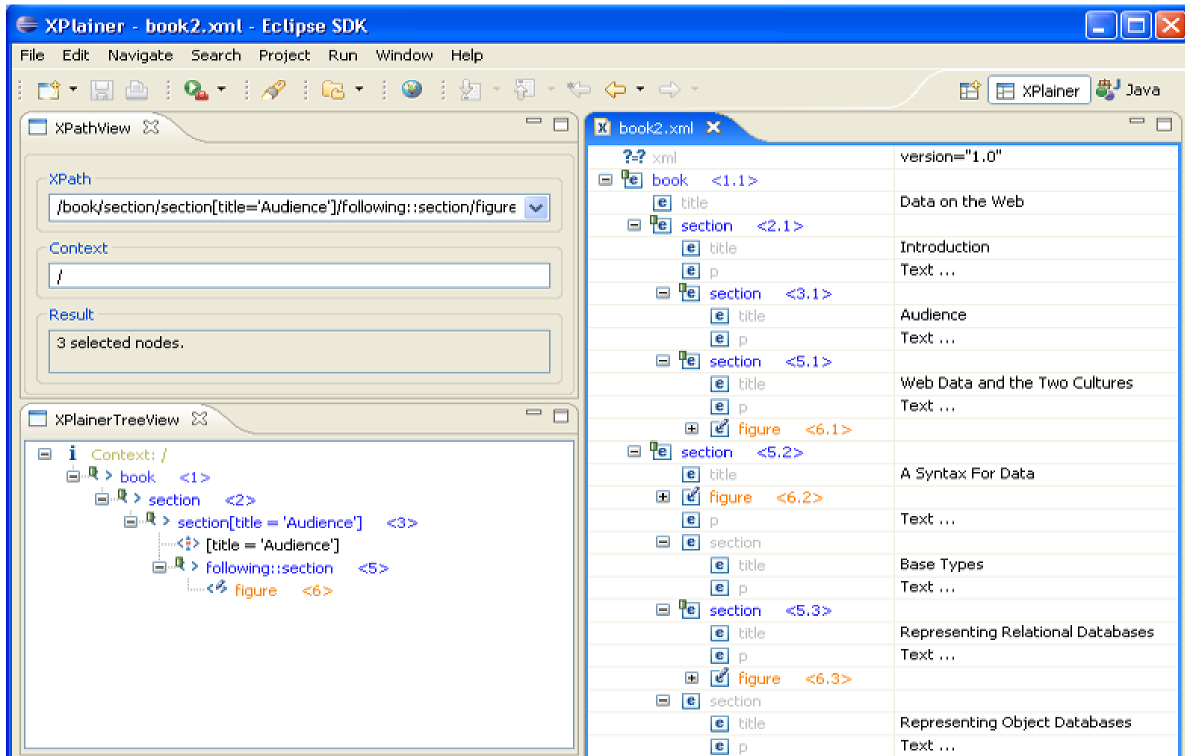

Figure 1. Explanation of query q_1

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 and provide examples to motivate explanations. Section 3 describes the formal foundations of the XPlainer language: *XPlainer trees* and the *visual explanation function*. In Section 4 we provide an overview of a tool based on the XPlainer language (together with a description of its implementation). We conclude in Section 5.

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 (a succinct formal description of the semantics of the XPath language is provided in Section 3).

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.

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 $title$ = “$Audience$” within a section of book.

An explanation of q_1 is depicted in Figure 1, a screenshot of XPlainer-Eclipse. 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 XPlainerTreeView and displays a specific parse tree representation of the XPath expression that appears in XPathView. While the formal definition of this parse tree is provided in Section 3 (see Definition 3.5), it should appear 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 XPlainerTreeView 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 XPlainerTreeView 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 $/book/section$ 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 $/book/section$ 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.

Example 2.2 Consider the expression

$$/Order/OrderLine/preceding - sibling :: *[1]$$

This example query selects the first preceding sibling of each OrderLine child of an Order (one DeliveryTerms and two OrderLine elements) in a Universal Business Language XML document describing a business transaction. The XPlainerTreeView is shown on the top left side of Figure 2 and the answer on the center.

Now consider a similar expression which contains parentheses

$$(/Order/OrderLine/preceding - sibling :: *)[1]$$

The explanation of the evaluation of the latter expression on the UBL XML document appears in the right side of Figure 2, while the XPlainerTreeView is shown on the bottom left. The parenthesis impacts 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. This modified example selects just one node, the first of the preceding siblings (the ID element).

The increasing difficulty of providing these explanations for arbitrary XPath queries motivated us to formally define the *semantics of explanations*. The following example shows a more complex expression with parenthesis, predicates and a disjunction.

Example 2.3 Consider the query

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

In queries like this, obtaining the intermediate nodes without such semantics becomes extremely challenging. We will come back to this query in Section 3 when we introduce the XPlainer language.

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 The XPlainer Language

This section introduces the XPlainer query language. We start with some preliminary definitions (XML trees, axis order and context). The second subsection introduces the XPlainer expressions and the concept of an *XPlainer tree* (a very specific parse tree for an XPath expression). The next subsection defines the semantics of explanations using a *visual explanation function* that associates intermediate nodes in an XPath expression evaluation with the corresponding nodes in the explanation tree. The fourth subsection mentions properties of the evaluation of explanations.

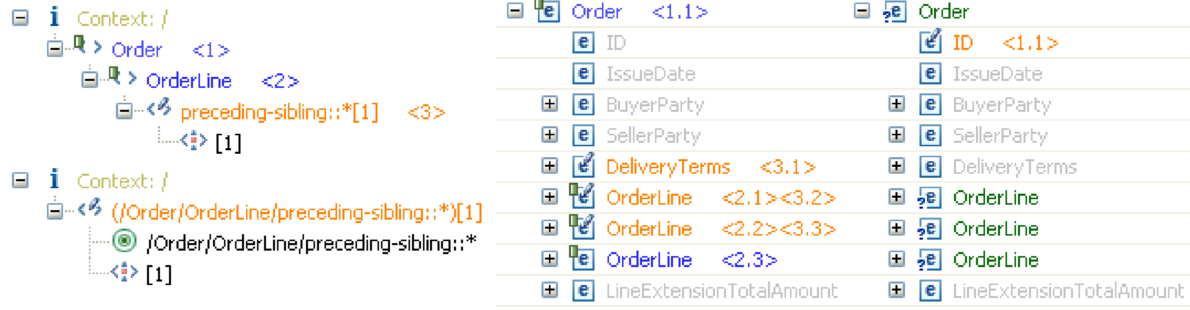

Figure 2. Explaining the impact of parenthesis

3.1 Preliminaries

An XML document is viewed as a unranked, ordered, and labeled tree in which elements are tree nodes. The structure of the tree is given by the nesting of the elements and their relative order in the document. In order to introduce XPlainer we will formalize next the notion of an *XML tree*.

Definition 3.1 (XML Tree) *Given an XML document* D*, we define the* XML tree *of* D *as an ordered tree* $T = (Inst, firstchild, nextsibling, Label, \lambda)$ *where*

- *Inst is a finite set of nodes;*
- $firstchild \in Inst \times Inst$ *represents the relationship between each non-leaf node and its first child;*
- $nextsibling \in Inst \times Inst$ *represents the relationship between two consecutive children of a non-leaf node;*
- *Label is a finite set of node names;*
- λ *is a* labelling function *that assigns labels to nodes in Inst by mapping* $Inst \longrightarrow Label$*;*

For navigating the XML tree, the mechanism used by XPlainer are the XPath *axes*, which are defined in terms of $firstchild$ and $nextsibling$, together with their inverses and their transitive closures. For instance, $child := firstchild.nextsibling^*$, and $following := ancestor\text{-}or\text{-}self.nextsibling.nextsibling^*.descendant\text{-}or\text{-}self$. All other axes are defined similarly.

Since XML documents are ordered, we need to define the *document order* $\prec_{doc}$ as a total order relation given by $descendant \bigcup following$, where $descendant$ and $following$ are the axes. Based on this order relation and its inverse ($ancestor \bigcup preceding$) we define next *axis order* and *axis position*.

Definition 3.2 *We define the binary* axis order *relation* $\prec_{axis}$ *in* $Inst \times Inst$ *as* $descendant \bigcup following$ *if* $axis \in \{self,\ child,\ descendant,\ descendant\text{-}or\text{-}self,\ following,\ following\text{-}sibling\}$ *and as* $ancestor \bigcup preceding$ *otherwise. Given a node set* $S \subseteq Inst^*$ *and* $v \in S$*, the position of* v *in* S *w.r.t.* $\prec_{axis}$ *is denoted by* $pos_{axis}(v, S)$*).*

Since XPath was designed to be embedded in other languages it has an interface that contains information about the *context* in which an expression will be evaluated. Given that XPath manipulates node sets, in addition to the node from which to start the evaluation the context has to contain the node's position relative to a node set and the node set size. This node set could be the result of the evaluation of another XPath expression or a construct of the host language.

Definition 3.3 (Context) *Let* $S \subseteq Inst^*$ *and* $v \in S$*. The* context *of* v *in* S *with respect to* $axis$ *is defined as the tuple* $c = \langle v, pos_{axis}(v, S), |S| \rangle$*. We say that* v *is the* context node*,* $pos_{axis}(v, S)$ *the* context position*, and* $|S|$ *the* context size.

3.2 XPlainer Expressions and Trees

In this subsection we introduce the syntax of the XPlainer query language. We begin by defining XPlainer expressions, which are (as expected) syntactically identical to the XPath expressions under explanation.

Definition 3.4 (XPlainer Grammar)

$$
\begin{array}{rcl}
e & := & disj \mid op(e_1, \ldots, e_m) \\
disj & := & locpath\ (\text{`}|\text{'}\ locpath)^* \\
locpath & := & par \mid step \mid abs \\
par & := & \text{`('}\ disj\ \text{`)'}\ (pred)^*\ (\text{`/'}\ locpath)? \\
step & := & axis\ \text{`::'}\ l\ (pred)^*\ (\text{`/'}\ locpath)? \\
abs & := & \text{`/'}\ locpath \\
pred & := & \text{`['}\ e\ \text{`]'}
\end{array}
$$

where $e, e_1 \ldots e_m$ *are called expressions,* $locpath$*,* $locpath_1, \ldots, locpath_m$ *are called location paths,* l *is*

a node name from the label alphabet Label (Definition 3.1), $axis$ is an XPath axis, and op is a place holder for ***any*** *XPath function and operators such as $+, -, *, div, =, \neq, \leq, <, \geq, >$, and $intersect$, as well as for context accessing functions $position()$ and $last()$.*

Note that the previous definition captures all the XPath constructs. For a detailed coverage of the XPath functions and operators the reader is referred to [27].

Next we introduce the notion of XPlainer tree, a parse tree specifically designed for the expressions in the XPlainer grammar defined above.

Definition 3.5 (XPlainer Tree) *Given an XPath expression query e, we define the* XPlainer tree *of e as an unordered tree $\mathcal{X}_e = (SubExpr, Edge, up, here, par, down)$ where*

- *$(SubExpr, Edge)$ is the parse tree for e according to the grammar in Definition 3.4;*
- *up, $here$, par and $down$ are* labeling functions *that assign XPath expressions to nodes in $SubExpr$;*
- *the* root *of $\mathcal{X}_e$ is $x_0 \in SubExpr$.*

The labeling functions $up(x)$, $here(x)$, $par(x)$, and $down(x)$ are defined for each $x \in SubExpr$ by introducing a recursive function T (Figures 3, 4, and 5) as follows.

Function $up(x_0)$ is empty and $up(x)$ is defined for all $x \neq x_0$ as follows:

$$up(x) = here(x'_1)/\ldots/here(x'_m)$$

where $x'_1, \ldots, x'_m$ are the ancestor nodes of x in $\mathcal{X}_e$.

Function $here(x)$ is defined for all $x \neq x_0$ in each of T's derivation rules (Figures 3, 4, and 5), whereas $here(x_0) = Context$.

In addition, $par(x)$ is explicitly defined for all x created by the parenthesis derivation rule of Figure 4. For all other nodes $x \neq x_0$, $par(x)$ is the same as $par(x')$, where x' is the parent of x in X_e, whereas $par(x_0)$ is empty.

Finally, $down(x_0) = e$ and $down(x)$ is defined for all $x \neq x_0$ as follows:

$$down(x) = \begin{cases} rlocpath, \text{ if } par(x') \text{ is empty} \\ intersect(rlocpath, par(x')), \text{ otherwise} \end{cases}$$

where x' is the parent of x in $\mathcal{X}_e$, and $intersect$ is an $op(e_1, e_2)$, as in rule e in Definition 3.4, that returns the intersection of the two nodesets produced by e_1 and e_2.

The $\mathcal{X}_e$ is constructed by starting from x_0 and recursively applying the derivation rules of T defined in Figures 3, 4, and 5 to $Context : e$.

Note that $rlocpath$ can be empty, in which case $T(rlocpath)$ does not get invoked hence terminating the

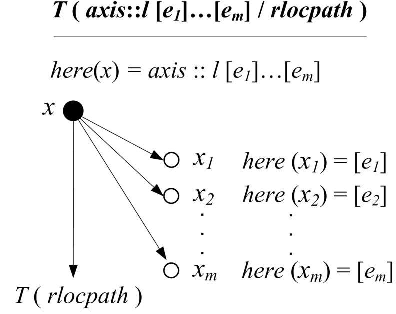

Figure 3. Location Step Derivation Rule

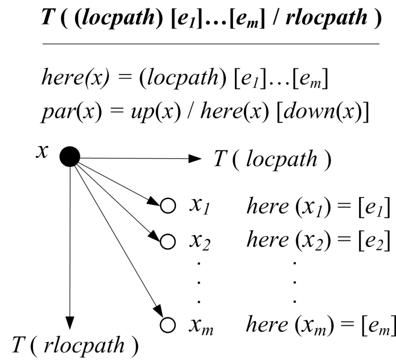

Figure 4. Parenthesis Derivation Rule

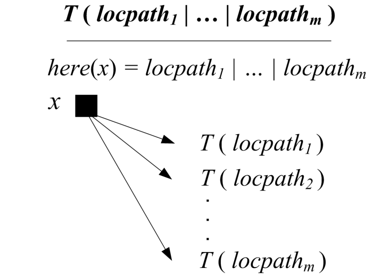

Figure 5. Disjunction Derivation Rule

construction of $\mathcal{X}_e$. Predicates $[e_i]$ can be empty as well. There is a node in $\mathcal{X}_e$ for each application of the grammar rules other than $locpath$.

Let us walk through an example to see how the XPlainer tree is constructed.

Example 3.6 Consider again query q_1 from Example 2.1 evaluated in $Context = /$ whose XPlainer tree $\mathcal{X}_{q_1}$ is shown in Figure 1. We begin the XPlainer tree construction by applying the context rule to the expression $Context : q_1$. This rule initializes $\mathcal{X}_{q_1}$ by creating the root node x_0 with $here(x_0) = Context$, $down(x_0) = q_1$ and $up(x_0) = \emptyset$. The construction continues by applying $T(q_1)$, which will invoke T derivation rules recursively until $rlocpath$

is empty in all possible branches. The first rule that applies is that of location step (Figure 3) with no predicates. The reason for that is that the entire expression matches the head of the rule $T(axis :: l[e_1] \ldots [e_m]/rlocpath)$, with $book$ matching $axis :: l$ (Remember that $book$ is $child :: book$ in the unabbreviated syntax) and $section/section[title = \text{"}Audience\text{"}]/following :: section/figure$ matching $rlocpath$. Its application creates then a new node $\langle 1 \rangle$ and sets its labeling functions as follows: $up(\langle 1 \rangle) = \emptyset$, $here(\langle 1 \rangle) = book$, and $down(\langle 1 \rangle) = section/section[title = \text{"}Audience\text{"}]/following :: section/figure$.

The construction continues by calling $T(rlocpath)$, and the next rule that applies is again location step with no predicates, this time matching $section$ against $axis :: l$ and $section[title = \text{"}Audience\text{"}]/following :: section/figure$ against $rlocpath$. Its application creates a new node $\langle 2 \rangle$ with the following labeling functions: $up(\langle 2 \rangle) = book$, $here(\langle 2 \rangle) = section$, and $down(\langle 2 \rangle) = section[title = \text{"}Audience\text{"}]/following :: section/figure$.

Next, function $T(rlocpath)$ is called again and the next rule that applies is location step with predicate $[title = \text{"}Audience\text{"}]$. At this point $rlocpath = following :: section/figure$ and a new expression node $\langle 3 \rangle$ is created together with one predicate node $\langle 4 \rangle$ (its label is not shown in Figure 1). Thus, the labeling functions for $\langle 3 \rangle$ and $\langle 4 \rangle$ are set as follows: $up(\langle 3 \rangle) = book/section$, $here(\langle 3 \rangle) = section[title = \text{"}Audience\text{"}]$, $down(\langle 3 \rangle) = following :: section/figure$, and $here(\langle 4 \rangle) = [title = \text{"}Audience\text{"}]$.

The construction continues for two more steps until $rlocpath$ is empty. At that point we get the complete XPlainer tree as shown in the XPathTreeView of Figure 1.

The following example shows the XPlainer tree for a more complex expression with nested parenthesized subexpressions.

Example 3.7 Let us consider now query q_2 from Example 2.3. For simplicity, we will replace the tag names by their initials and we obtain the XPath expression $b/((s[1]/s)[2]|(s[3]/s)[4]/f])[5]/t$. The XPlainer tree X_{q_2} of q_2 is shown in Figure 6, where the edges in the Figure are the edges in $Edges$ and $SubExpr = \{x_0, \ldots, x_{15}\}$. In addition, the labels next to each node in the Figure (in colour and in black) are the definitions of the $here()$ function for each node: for instance, $here(x_2) = ((s[1]/s)[2]|(s[3]/s)[4]/f)[5]$ and $here(x_3) = [5]$.

3.3 The Semantics of Explanations

In this subsection we present the semantics of XPlainer, which relies on the semantics of XPath. Our presentation is

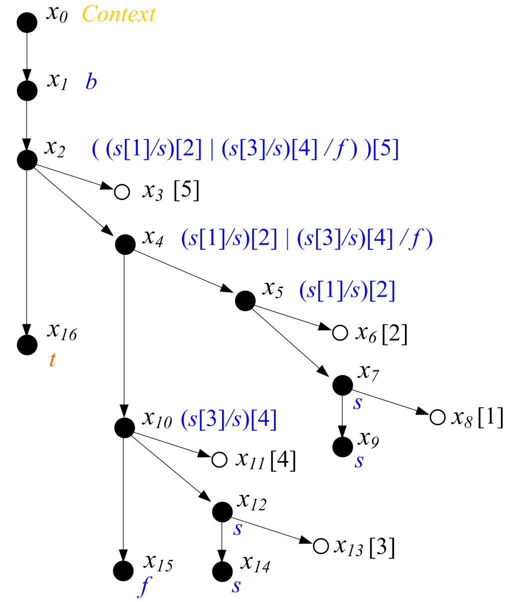

Figure 6. XPlainer tree for expression q_2

similar to the formalization given in [13]. The original formulation of the denotational semantic functions has been modified to better capture all the relevant constructs in the standard. A significant addition to the rules is the proper treatment of the interaction of parenthesis followed by predicates (a language feature that was not formalized in [13]).

The semantics of XPlainer expressions are defined by semantics functions $\mathcal{E}$ and $\mathcal{L}$ in Figure 7 and 8. Function $\mathcal{E}$ defines the semantics of XPath expressions on a context whereas function $\mathcal{L}$ defines the semantics of locations paths on a node. The distinction between context-based and node-based evaluation comes from the fact that some functions like $position()$ and $last()$ need to be evaluated on a context (they return the context position and the context size respectively). The evaluation of location paths, on the other hand, requires only the context node. The link between these two semantics functions are rules (1), (9) and (6).

With functions $\mathcal{E}$ and $\mathcal{L}$ we are able to define a new function that provides the semantics of XPlainer expressions. This *visual explanation function* returns precisely the intermediate nodes corresponding to each step in the evaluation.

Definition 3.8 (Visual Explanation Function) *Given an XML tree T that is the input to an XPath expression query e with XPlainer tree $\mathcal{X}_e$ evaluated on a context c, we define a* visual explanation function $V_{T,c} : SubExpr \to 2^{Inst}$ *as*

$$V_{T,c}(x) := \mathcal{E}[\![up(x)/here(x)[down(x)]]\!](c)$$

In other words, the function $V_{T,c}(x)$ highlights the intermediate nodes in T (or the answer nodes of e when $down(x)$ is not defined) that correspond to x in $\mathcal{X}_e$.

$$\mathcal{D}[\![locpath_1|\dots|locpath_m]\!](v) := \bigcup_{i=1}^{m} \mathcal{L}[\![locpath_i]\!](v) \quad (5)$$

$$\mathcal{L}[\![(locpath)[e_1]\dots[e_m]\,]\!](v) := \{\, w \mid w \in S \,\wedge\, S = \mathcal{D}[\![locpath]\!](v) \, \bigwedge_{i=1}^{m}(\mathcal{E}[\![e_i]\!](w, pos_{doc}(w,S), |S|) = true)\} \quad (6)$$

$$\mathcal{L}[\![locpath_1/locpath_2]\!](v) := \bigcup_{w \in \mathcal{L}[\![locpath_1]\!](v)} \mathcal{L}[\![locpath_2]\!](w) \quad (7)$$

$$\mathcal{L}[\![/locpath]\!](v) := \mathcal{L}[\![locpath]\!](v_0) \quad (8)$$

$$\mathcal{L}[\![axis :: l[e_1]\dots[e_m]\,]\!](v) := \{w|w \in S \,\wedge\, S = \{v'|\langle v, v'\rangle \in axis \,\wedge\, \lambda(v') = l\} \; \bigwedge_{i=1}^{m}(\mathcal{E}[\![e_i]\!](w, pos_{axis}(w,S), |S|) = true)\} \quad (9)$$

Figure 8. Semantics of Location Paths

$$\mathcal{E}[\![locpath]\!](\langle v,k,n\rangle) := \mathcal{D}[\![locpath]\!](v) \quad (1)$$

$$\mathcal{E}[\![position()]\!](\langle v,k,n\rangle) := k \quad (2)$$

$$\mathcal{E}[\![last()]\!](\langle v,k,n\rangle) := n \quad (3)$$

$$\mathcal{E}[\![Op(e_1,\dots,e_m)]\!](\langle v,k,n\rangle) := \mathcal{F}[\![Op]\!](\mathcal{E}[\![e_1]\!](\langle v,k,n\rangle),\dots,\mathcal{E}[\![e_m]\!](\langle v,k,n\rangle)) \quad (4)$$

Figure 7. Semantics of XPath Expressions

It is important to note that the semantics given by the visual explanation function is not the only one we can define. For instance, we may decide to return not just the intermediate nodes but also the entire subtrees rooted at them. This is the alternative semantics provided by the *visual* explanation function* defined next.

Definition 3.9 (Visual* Explanation Function) *Given an XML tree T that is the input to an XPath expression query e with XPlainer tree $\mathcal{X}_e$ evaluated on a context c, we define a* visual * explanation function $V^*_{T,c} : SubExpr \to 2^{Inst}$ *as*

$$V^*_{T,c}(x) := \bigcup_{y \in SubTree} \mathcal{E}[\![up(y)/here(y)[down(y)]]\!](c)$$

where $SubTree := \{y \mid y$ belongs to the subtree of $\mathcal{X}_e$ rooted at x $\}$

The function $V^*_{T,c}(x)$ highlights the intermediate nodes in T (or the answer nodes of e when $down(x)$ is not defined) that correspond to x and all nodes below x in $\mathcal{X}_e$. This is the semantics we use for explaining predicates (see the nodes highlighted in green in the far right side of Figure 2).

Other semantics for XPlainer expressions are also possible. For instance, we may want to define a *sub-document explanation function* which would return the sub-document that contains all those intermediate nodes that contribute to the original XPath result together with their ancestors. This semantics is similar to that of XSquirrel [22] subtree queries, with the difference that XPlainer includes nodes that are not necessarily ancestors nor descendants of the answer. For instance, consider again the explanation of query q_1 (Figure 1). The highlighted section with number $\langle 3 \rangle$ is neither ancestor or descendant of the answer and therefore would not be included in a subtree query by XSquirrel, but it will appear in an XPlainer sub-document explanation.

3.4 Properties

Since there is one XPath expression per node in any given XPlainer Tree, the number of expressions we need to evaluate is $|Q|$, where $|Q|$ is the size of the XPath query. Gottlob et al. propose a polynomial-time algorithm for XPath evaluation [13],which runs in time $O(|D|^5 \times |Q|^2)$, where $|D|$ denotes the size of the XML data. Therefore, we obtain a bound on the complexity of obtaining the visual explanations proposed above that is also polynomial ($O(|D|^5 \times |Q|^3)$ if it utilizes the polynomial-time XPath evaluation algorithm in [13].

At this point it is important to step back and observe that the notion of associating a node set with the syntactic elements of the XPath query contributes to the low complexity. This approach is not necessarily the only way to attempt to provide an explanation of the intermediate results in an XPath expression. In fact, an initial prototype of the XPlainer system simply collected the intermediate nodes in nested sequences. This initial prototype suffered from evaluation time $O(|D|^{|Q|})$, since we could potentially have $|Q|$ nested node sets to carry around. Even the simple query on the right-hand side of Figure 2 (where the OrderLine node labeled by $\langle 2.1 \rangle$ and $\langle 3.2 \rangle$ matches multiple XPath subexpressions) is an example query that will cause this run time behaviour. The semantics that we described earlier in this section avoids this complexity blowup.

4 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 [23]:

> 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 (see also [16]) 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.

XPlainer-Eclipse has a number of additional interactive features not covered through the examples in the paper (an earlier version of the tool was presented at [11]). 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, as shown in the answer displayed in the far right side of Figure 2. The same example also shows 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 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 complete query, but then after collapsing subexpressions (removing constrains) the modified query can be satisfied.

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.

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. 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 [15], 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.

5 Conclusions

This paper provides a description of 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 expression.

The main novelty and contribution of our proposal is the

formal definition of the semantics of the XPlainer language. XPlainer uses visualizations as a mechanism to provide explanations for the semantics of an existing query language. While XPlainer specifically targets XPath (a significant and increasingly popular query language), the concept of *explanation queries* that assist developers to learn, understand, use, and debug query expressions has validity beyond a specific target language.

The paper also describes an Eclipse-based tool implementing XPlainer. The users of XPlainer-Eclipse can interact with tree views of XPath expressions and input XML documents. In the XML editor view, intermediate 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 XPath 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.

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.

Acknowledgments

Portions of this work were carried in collaboration with Dr. Bill O'Farrell and Julie Waterhouse at the IBM Center 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, and the reviewers for feedback. Financial support was provided by an Eclipse Innovation Award, an IBM Center for Advanced Studies (CAS) Fellowship, and the Natural Sciences and Engineering Research Council (NSERC).

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] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT '2001*, pages 316–330, 2001.
[10] M. Consens and A. Mendelzon. Hy+: a Hygraph-based query and visualization system. In *SIGMOD '93*, pages 511–516, 1993.
[11] 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.
[12] Y. Cui and J. Widom. Practical lineage tracing in data warehouses. In *ICDE '2000*, pages 367–378, 2000.
[13] G. Gottlob, C. Koch, and R. Pichler. XPath processing in a nutshell. *SIGMOD Record*, 32(1):12–19, 2003.
[14] IBM. WebSphere Studio Application Developer (WSAD) 5.1 XPath Expression. http://www-306.ibm.com/software/awdtools/studioappdev/.
[15] JAXP. Java API for XML Processing (JAXP) 1.3. http://java.sun.com/webservices/jaxp/index.jsp.
[16] F. R. John W. S. Liu, Mariano P. Consens. XPlainer: Explaining XPath with Eclipse. In *Eclipse Technology eXchange (ETX) Workshop*, 2006.
[17] N. Leghari. Visual XPath. http://weblogs.asp.net/nleghari/articles/27951.aspx.
[18] 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.
[19] Logilab.org. Logilab - XPath Visualizer 1.0. http://www.logilab.org/projects/xpathvis/1.0.
[20] G. Miklau and D. Suciu. XViz: A tool for visualizing XPath expressions. In *XSym 2003*, pages 479–490, 2003.
[21] 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.
[22] A. Sahuguet and B. Alexe. Sub-document queries over XML with XSQuirrel. In *WWW '05*, pages 268–277, 2005.
[23] E. R. Tufte. *Envisioning information*. Graphics Press, CT, USA, 1990.
[24] E. R. Tufte. *Visual explanations: images and quantities, evidence and narrative*. Graphics Press, CT, USA, 1997.
[25] K. Vadaparty, Y. A. Aslandogan, and G. Ozsoyoglu. Towards a unified visual database access. In *SIGMOD '93*, pages 357–366, 1993.
[26] W3C. XML Path Language (XPath) 2.0. http://www.w3.org/TR/xpath20, 2005.
[27] W3C. XQuery 1.0 and XPath 2.0 Functions and Operators, 2005. http://www.w3.org/TR/xpath-functions/.
[28] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *ICDE '97*, pages 91–102, 1997.
[29] M. Zloof. Query-by-example: A data base language. *IBM Syst. J.*, 16(4):324–343, 1977.