XPlainer: Visual Explanations of XPath Queries

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

John W. Liu  
University of Toronto  
wliu@cs.toronto.edu

Flavio Rizzolo  
University of Toronto  
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. This creates the need in a large population of developers (as well as less technical users) to understand and employ the XPath language effectively.

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 of XPlainer queries based on the semantics of XPath.

We describe a tool that uses XPlainer queries to provide visual explanations. The XPlainer 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 [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.

The use of XPath expressions in a large variety of computer languages (such as XSLT, XQuery, BPEL, Schema, XJ, SQL extensions, etc.) motivates the interest of a large population of developers in learning how to use the language. To make matters more interesting, in most cases (XQuery being the obvious exception) XPath is not the “native language” of these developers. Providing explanations of why XPath expressions return specific answers is a compelling approach to facilitate understanding (and debugging) XPath applications.

This paper introduces the concept of explanation queries to address the developer needs described above. 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 how providing explanations for arbitrarily complex expressions while supporting all the constructs in the XPath language is a non trivial task.

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 also describe a visual explanation tool based on the XPlainer language that assists XML developers to learn, understand, use, and debug XPath expressions. 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. This includes supporting consistently all the language constructs and the functions that the XPath processor can access. 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.

1.1 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 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.

1.2 Related Work

There is a rich literature in graphical query languages, starting with QBE [24] 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).

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 [12] and Oxygen XML Editor [3]. These tools 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 [12, 3, 16].

The abundance of these tools provide strong evidence of the practical need for the kind of visual aid that XPlainer targets. However, none of the tools described in preceding paragraph has the ability to provide any explanatory information regarding the evaluation of an XPath expression. All the tools currently available simply display the result of the XPath expression1. 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 possess.

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).

1.3 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. Section 3 describes the formal foundations of the XPlainer language: XPlainer trees and the visual explanation function. The later function provides the semantics that assigns an XPlainer query result to an arbitrary XPath expression. In Section 4 we provide an overview of the interface and use 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 through a series of examples. 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 examples.

Example 2.1 Let us consider the query

\[ q_1 = \text{descendant} :: \text{section} / \text{descendant} :: \text{figure} \]

This simple query returns all the figures in the document that appear within sections. Query \( q_1 \) has two subexpressions: \( \text{descendant} :: \text{section} \) and \( \text{descendant} :: \text{figure} \). Each of these subexpressions are XPath location steps. The intermediate nodes of \( q_1 \) are the four blue \textit{section} elements shown in Figure 1 (one of them with a nested \textit{section}), which are all sections that contain figures (the orange nodes in the Figure). The sequence numbers next to the blue and orange nodes correspond to the numbers in the XPlainer tree of Figure 2. (The XPlainer tree is a very specific parse tree for an XPath expression.) For instance, nodes labeled by (1) in Figure 1 are the intermediate nodes of subexpression \( \text{descendant} :: \text{section} \). These intermediate nodes are part of an \textit{explanation} of \( q_1 \) on our sample XML document.

Since current XPath query evaluation tools do not provide explanations, the only available debugging techniques involve either partial evaluation of subexpression 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 \( q_1 \) from the previous example would return all sections in the document after evaluating \( \text{descendant} :: \text{section} \), even those that do not contain figures (like the second \textit{section} in the Figure 1 document for instance). In contrast, an explanation of the query would have only those sections that contain figures and therefore contribute in some way to the answer. Thus, a partial evaluation provides in general a \textit{superset} of the intermediate nodes for \( q_1 \).

Let us see if reversing the axes in each subexpression provides the intermediate nodes. If we reverse the axis in \( q_1 \)'s last step we obtain the \textit{ancestor} axis. However, evaluating the \textit{ancestor} axis on the answer of \( q_1 \) returns all ancestors of the figure rather than just the sections. Thus, the reverse axis evaluation in our example also returns a \textit{superset} of the intermediate nodes and therefore does not constitute an explanation.

The fact that both partial and reversed axis evaluations do not necessarily return precisely the intermediate nodes does not affect only queries with \textit{descendant}
the intermediate nodes. For instance, the next subexpression \( \text{section}[\text{title} = \text{"Audience"}] \), returns exactly the intermediate node, the \( \text{section} \) labeled by \( \langle 3 \rangle \). In contrast, a partial evaluation of the next subexpression, following :: \( \text{section} \), returns again a superset of the intermediate nodes: it returns all sections after section \( \langle 3 \rangle \), including those that do not contain figures.

An evaluation that reverses the axis will not give us exactly the intermediate nodes for all subexpressions in \( q_2 \) either. Evaluating the last reversed subexpression entails obtaining the parent of all figure nodes in the answer. (Remember that \( \text{figure} \) is child :: \( \text{figure} \) in the unabbreviated syntax, and its reverse axis is parent). This evaluation gives us correctly the three intermediate \( \langle 5 \rangle \) nodes that appear in the Figure. In contrast, the reversed evaluation of the next subexpression, following :: \( \text{section} \) will return all the preceding nodes, when the only intermediate node at that point is section \( \langle 3 \rangle \).

We have shown with the previous examples that we cannot rely on either partial evaluation or in evaluations that simply reverse the axis to explain why an XPath expression returns a specific answer. The increasing complexity of providing intermediate results that can help to understand the behavior of XPath queries motivated us to formally define the semantics of explanations. XPlainer is the proposed new language: 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 provides the definition the XPlainer query language. We start with some preliminary definitions (XML trees and XPath axes). 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.

3.1 Preliminaries

An XML document is viewed as an 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 \( T = (\text{Inst}, \text{firstchild}, \text{nextsibling}, \text{Label}, \lambda) \) where

- \( \text{Inst} \) is a finite set of nodes;
- \( \text{firstchild} \) is a binary relation in \( \text{Inst} \times \text{Inst} \) that represents the relationship between each non-leaf node and its first child: \( \langle v, w \rangle \in \text{firstchild} \iff w \) is the first child of \( v \);
- \( \text{nextsibling} \) is a binary relation in \( \text{Inst} \times \text{Inst} \) that represents the relationship between two consecutive children of a non-leaf node: \( \langle v, w \rangle \in \text{nextsibling} \iff v \) and \( w \) are children of the same node in \( \text{Inst} \) and \( w \) is the first neighbour of \( v \) to the right;
• The root of $T$ is $v_0 \in \text{Inst} | \neg \exists v \in \text{Inst} : (v, v_0) \in \text{firstchild} \lor (v, v_0) \in \text{nxtsibling};$

• Label is a finite set of node names called label alphabet;

• $\lambda$ is a labelling function that assigns labels to nodes in $\text{Inst}$ by mapping $\text{Inst} \rightarrow \text{Label};$

For navigating the XML tree, the mechanism used by XPlainer are the XPath axes. We give next the definition of the XPath axes in terms of $\text{firstchild}$, $\text{nxtsibling}$, their inverses and self (the reflexive binary relation over $\text{Inst}$).

**Definition 3.2 (XPath Axes)**

- $\text{self} := \{(v, v) \mid v \in \text{Inst}\}$
- $\text{child} := \text{firstchild} \cup \text{nxtsibling}^*$
- $\text{parent} := (\text{nxtsibling}^{-1}) \cdot \text{firstchild}^{-1}$
- $\text{ancestor} := (\text{firstchild}^{-1} \cup \text{nxtsibling}^{-1})^* \cdot \text{firstchild}^{-1}$
- $\text{descendant} := \text{firstchild} \cup \text{nxtsibling}^*$
- $\text{following} := \text{ancestor-or-self} \cup \text{nxtsibling}$
- $\text{preceding} := \text{ancestor-or-self} \cdot \text{nxtsibling}^{-1} \cup \text{self}$
- $\text{following-sibling} := \text{nxtsibling} \cup \text{nxtsibling}^*$
- $\text{preceding-sibling} := (\text{nxtsibling}^{-1})^* \cdot \text{nxtsibling}^{-1}$
- $\text{descendant-or-self} := \text{descendant} \cup \text{self}$
- $\text{ancestor-or-self} := \text{ancestor} \cup \text{self}$

Since XML documents are ordered, we need to define the document order $\prec_{\text{doc}}$ as a total order relation given by $\text{descendant} \cup \text{following}$, where $\text{descendant}$ and $\text{following}$ are the axes defined in Definition 3.2. Based on this order relation and its inverse $(\text{ancestor} \cup \text{preceding})$ we define next axis order and axis position.

**Definition 3.3** We define the binary axis order relation $\prec_{\text{axis}}$ in $\text{Inst} \times \text{Inst}$ as descendant $\cup$ following if $\text{axis} \in \{\text{self}, \text{child}, \text{descendant}, \text{descendant-or-self}, \text{following}, \text{following-sibling}\}$ and as ancestor $\cup$ preceding otherwise. Given a node set $S \subseteq \text{Inst}^*$ and $v \in S$, the position of $v$ in $S$ w.r.t. $\prec_{\text{axis}}$ is denoted by $\text{pos}_{\text{axis}}(v, S)$.

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 3.4 (Context)** Let $S \subseteq \text{Inst}^*$ and $v \in S$. The context of $v$ in $S$ with respect to axis is defined as the tuple $c = (v, \text{pos}_{\text{axis}}(v, S), |S|)$. We say that $v$ is the context node, $\text{pos}_{\text{axis}}(v, S)$ 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 3.5 (XPlainer Grammar)**

$$e := \text{disj} \mid \text{op}(e_1, \ldots, e_m)$$

$$\text{disj} := \text{locpath}_1 \mid \ldots \mid \text{locpath}_m$$

$$\text{locpath} := \text{par} \mid \text{step} \mid \text{abs}$$

$$\text{par} := (\text{disj}) \cdot \text{pred}_1 \ldots \cdot \text{pred}_m (\text{/locpath})?$$

$$\text{step} := \text{axis} :: l \cdot \text{pred}_1 \ldots \cdot \text{pred}_m (\text{/locpath})?$$

$$\text{abs} := \text{/locpath}$$

$$\text{pred} := [e]$$

where $e, e_1, \ldots, e_m$ are called expressions, locpath, locpath$_1$, ..., locpath$_m$ are called location paths, $l$ is a node name from the label alphabet Label (Definition 3.1), axis is an XPath axis (Definition 3.2), and op is a place holder for any of the XPath functions and operators such as $+,-,*,\text{div},=,\neq,\subset,\supset$, and intersect, as well as for context accessing functions position() and last().

For a detailed coverage of the XPath functions and operators the reader is referred to [23]. Next we introduce the notion of XPlainer tree, a particular parse tree of the expressions in the XPlainer grammar defined above.

**Definition 3.6 (XPlainer Tree)** Given an XPath expression query $e$, we define the XPlainer tree of $e$ as an unordered tree $X_e = (\text{SubExpr}, \text{Edge}, \text{up}, \text{here}, \text{par}, \text{down})$ where

- $(\text{SubExpr}, \text{Edge})$ is the parse tree for $e$ according to the grammar in Definition 3.5;

- up, here, par and down are labeling functions that assign XPath expressions to nodes in $\text{SubExpr}$;

- the root of $X_e$ is $x_0 \in \text{SubExpr}$.

Labeling functions up(), here(x), par(x) and down(x) are defined for each $x \in \text{SubExpr}$ by the recursive function $T$ (Figures 5 through 7) as follows.

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

$$\text{up}(x) = \text{here}(x_1')/\ldots/\text{here}(x_m')$$

where $x_1', \ldots, x_m'$ are the ancestor nodes of $x$ in $X_e$. 
Function $here(x)$ is defined for all $x \neq x_0$ in each of $T$'s derivation rules (Figures 5 through 7), whereas $here(x_0) = Context$.

In addition, $par(x)$ is explicitly defined for all $x$ created by the parenthesis derivation rule of Figure 6. 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 $X_e$, and $intersect$ is an $op(e_1, e_2)$, as in rule $e$ in Definition 3.5, that returns the intersection of the two nodesets produced by $e_1$ and $e_2$.

$$T(\text{axis}::l[e_1]...[e_m] / rlocpath)$$

$$here(x) = axis :: l[e_1]...[e_m]$$

\[ x \]

Figure 5: Location Step Derivation Rule

$$T(\text{locpath} | ... | \text{locpath}_n)$$

$$here(x) = \text{locpath}_1 | ... | \text{locpath}_n$$

\[ \begin{array}{c}
T(\text{locpath}_1) \\
T(\text{locpath}_2) \\
T(\text{locpath}_n)
\end{array} \]

Figure 7: Disjunction Derivation Rule

Note that $rlocpath$ can be empty, in which case $T(rlocpath)$ does not get invoked hence terminating the construction of $X_e$. Predicates $[e_i]$ can be empty as well. There is a node in $X_e$ 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 3.7** Consider again query $q_2$ from Example 2.2 evaluated in $Context = /$ whose XPlainer tree is shown in Figure 4. We begin the XPlainer tree construction by applying the context rule to the expression $Context : q_2$. This rule initializes the tree by creating the root node $x_0$ with $here(x_0) = Context$, $down(x_0) = q_2$ and $up(x_0)$ is set to empty. The construction continues by applying $T(q_2)$, which will invoke $T$ derivation rules recursively until $rlocpath$ is empty in all possible branches. The first derivation rule that applies is that of location step (Figure 5) with no predicates. The reason for that is that the entire expression matches the head of the rule $T(\text{axis} :: l[e_1]...[e_m]/rlocpath)$, with $book$ matching $axis :: l$ (Remember that $book$ is $child :: book$ in the unabbreviated syntax) and $section/section[title = “Audience”]/following :: section/figure$ matching $rlocpath$. Its application creates then a new node (1) and sets (1)'s labeling functions (except $up((1))$, which is empty) as follows

$$here((1)) = book$$

$$down((1)) = section/section[title = “Audience”]$$

$$/following :: section/figure$$

and the construction continues by calling $T(rlocpath)$. The next rule that applies is again location step with no predicates, this time matching $section$ against $axis :: l$ and $section[title = “Audience”]/following :: section/figure$ against $rlocpath$. Its application creates a new node (2) with the following labeling functions

$$up((2)) = book$$

$$here((2)) = section$$
\[ \mathcal{E}[loxpath](v, k, n) := D[loxpath](v) \]
\[ \mathcal{E}[(v, k, n)] := k \]
\[ \mathcal{E}[(v, k, n)] := n \]
\[ \mathcal{E}[Op(e_1, \ldots, e_m)]((v, k, n)) := F[Op](\mathcal{E}[e_1]|(v, k, n), \ldots, \mathcal{E}[e_m]|(v, k, n)) \]

Figure 8: Semantics of XPath Expressions

\[ D[loxpath_1 \ldots loxpath_m](v) := \bigcup_{i=1}^{m} L[loxpath_i](v) \]
\[ L[(loxpath)|e_1 \ldots e_m](v) := \{ w \mid w \in S \land S = D[loxpath](v) \bigcap_{i=1}^{m} (\mathcal{E}[e_i]|(w, pos_{doc}(w, S), |S|) = true) \}
\[ L[loxpath_1/loxpath_2](v) := \bigcup_{w \in L[loxpath_1](v)} L[loxpath_2](w) \]
\[ L[loxpath](v) := L[loxpath](v_0) \]
\[ L[axis :: l|e_1 \ldots e_m](v) := \{ w \mid w \in S \land S = \{ v' \mid (v, v') \in axis \land (v') = l \} \bigcap_{i=1}^{m} (\mathcal{E}[e_i]|(w, pos_{axis}(w, S), |S|) = true) \}

Figure 9: Semantics of Location Paths

\textit{down}(2) = \textit{section}[title = “Audience”]
\textit{/following :: section/figure}

and the construction continues by calling \( T(rlopath) \). The next rule that applies is again location step but this time with one predicate \( [\textit{title} = “Audience”] \). At this point \( rlopath = \textit{following :: section/figure} \) and a new expression node (3) is created together with one predicate node (4) (its label is not shown in Figure 4). Thus, the labeling functions for (3) and (4) are set as follows:

\( up(3) = \textit{book/section} \)
\( here(3) = \textit{section}[title = “Audience”] \)
\( down(3) = \textit{following :: section/figure} \)
\( here(4) = [\textit{title} = “Audience”] \)

The construction continues for two more steps until \( rlopath \) is empty. At that point we get the complete XPlainer tree as shown in Figure 4, with the labeling functions of the two remaining nodes defined as follows:

\( up(5) = \textit{book/section/section}[title = “Audience”] \)
\( here(5) = \textit{following :: section} \)
\( down(5) = \textit{figure} \)
\( up(6) = \textit{book/section/section}[title = “Audience”] \)
\( /\textit{following :: section} \)
\( here(6) = \textit{figure} \)

The following example shows the XPlainer tree for an expression with nested parenthesized subexpressions.

\textbf{Example 3.8} Let \( q_3 \) be the XPath expression \( a/((b[1]/c)[2]|(e/f[3])[4]/g)|[5]/h[6] \). The XPlainer tree \( X_{q_3} \) of \( q_3 \) is shown in Figure 10, where the edges in the Figure are the edges in Edges and \( SubExpr = \{x_0, \ldots, x_{16}\} \). In addition, the labels next to each node in the Figure (in colour and in black) are the definitions of the \textit{here()} function for each node: for instance, \( here(x_2) = ((b[1]/c)[2]|(e/f[3])[4]/g)[5] \) 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 similar to the formalization given in [11]. 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 [11]).

The semantics of XPlainer expressions are defined by semantics functions \( \mathcal{E} \) and \( \mathcal{L} \) in Figure 8 and 9. 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 \( \text{position()} \) and \( \text{last()} \)
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 rules are functions (1), (9) and (6).

**Definition 3.9 (Visual Explanation Function)**

Given an XML tree $T$ that is the input to an XPath expression query $e$ with XPlainer tree $X_e$ evaluated on a context $c$, we define a visual explanation function $V_{T,c} : \text{SubExpr} \rightarrow 2^{\text{Inst}}$ as

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

The function $V_{T,c}(x)$ highlights the intermediate nodes in $T$ (or the answer nodes of $e$ when $\text{down}(x)$ is not defined) that correspond to $x$ in $X_e$.

We will illustrate with the next example how the visual explanation function $V_{T,c}$ is evaluated using the semantic rules from Figures 8 and 9.

**Example 3.10** Let Context be $(v_0, 1, 1)$, let $e = \text{up}(2)/\text{here}(2)[\text{down}(2)]$ and let $V_{T,c}(2) = \mathcal{E}[e](v_0, 1, 1)$ be the visual explanation function for node (2) of query $q_2$ in Example 3.7. The evaluation of $e$ on Context $= (v_0, 1, 1)$ returns the only intermediate node from the evaluation of subexpression (2) in $q_2$, which is the first section (see node labeled by (2) in Figure 3).

By replacing $\text{up}(2)$, $\text{here}(2)$, and $\text{down}(2)$ in $e$ with their values from Example 3.7 we get

$$e = \text{book}/\text{section}[\text{section}[\text{title} = "\text{Audience}"] ]/\text{following} :: \text{section}/\text{figure}$$

and we begin the evaluation of $\mathcal{E}[e](v_0, 1, 1))$. Since $e$ can be viewed as a location path, the first rule we apply is (1)

$$\mathcal{E}[e](v_0, 1, 1) := \mathcal{L}[e](v_0)$$

Rule (1) translates the evaluation on the entire context $(v_0, 1, 1)$ to an evaluation on just the context node $v_0$. In order to simplify the exposition, let

$$\text{rlocpath} = \text{section}[\text{section}[\text{title} = "\text{Audience}"] ]/\text{following} :: \text{section}/\text{figure}$$

be the location path that comes after $\text{book}$ in $e$ such that $e = \text{book}/\text{rlocpath}$. Since $e$ is a composition of a location step ($\text{book}$) and a location path ($\text{rlocpath}$), we apply rule (7)

$$\mathcal{L}[e](v_0) := \bigcup_{w_1 \in \mathcal{L}[\text{book}](v_0)} \mathcal{L}[\text{rlocpath}](w_1)$$

which entails evaluating the location path on the union of all $w_1$’s that are returned by the evaluation of the location step. In order to obtain those $w_1$’s we apply rule (9) to the location step $\text{book}$

$$\mathcal{L}[\text{book}](v_0) := \{ w_1 \mid w_1 \in S \land \lambda(v) = \text{book} \}$$

and finish with the evaluation of the first part of the composition.

Next we evaluate $\text{rlocpath}$ by applying rule (9). For simplification, let

$$\text{pred} = \text{section}[\text{title} = "\text{Audience}"] ]/\text{following} :: \text{section}/\text{figure}$$

be the predicate in $\text{rlocpath}$ such that $\text{rlocpath} = \text{section}[\text{pred}]$. Then

$$\mathcal{L}[\text{rlocpath}](w_1) := \{ w_2 \mid w_2 \in S \land \lambda(v) = \text{section} \land \mathcal{E}[\text{pred}](\langle w_2, \text{pos}(w_2, S), |S| \rangle) = \text{true} \}$$

which returns all $w_2$’s that are section children of the $w_1$’s and satisfy the predicate $\text{pred}$. The $w_2$’s that do satisfy the predicate $\text{pred}$ are in fact the intermediate nodes from the evaluation of subexpression (2) in $q_2$ (the node labeled with (2) in Figure 3).
In order to evaluate \( \text{pred} \) we invoke rule (1) that transforms the evaluation on a context to an evaluation on a context node

\[
E[\text{pred}](w_2, \text{pos\_child}(w_2, S), |S|) := \mathcal{L}[\text{pred}](w_2)
\]

and continue with the evaluation like we did for \( rlocpath \). The \( w_2 \)'s that satisfy the predicate are those for which \( \mathcal{L}[\text{pred}](w_2) \) is not empty.

An alternative semantics to the one given by the visual explanation function of Definition 3.9 is provided by the definition below.

**Definition 3.11 (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 \( V^*_{T,c} : \text{SubExpr} \rightarrow 2^{\text{inst}} \) as

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

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

The function \( V^*_{T,c}(x) \) highlights the intermediate nodes in \( T \) (or the answer nodes of \( e \) when \( \text{down}(x) \) 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 Figure 12).

Another semantics we can define is the sub-document explanation: it returns 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 [18] 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 \( q_2 \) (Figure 3). The highlighted section with number (3) is neither ancestor nor descendant of the answer and therefore will 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 [11], 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 \( O(|D|^5 \times |Q|^2) \) if it utilizes the polynomial-time XPath evaluation algorithm in [11].

At this point 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 \( O(|D|^5 |Q|^2) \), since we could potentially have \( |Q| \) nested node sets to carry around. Even the simple query in Figure 11, where the same node in the XML document matches multiple XPath subexpressions (e.g., the first coloured section node labeled by (2) and (4) on the XML view) 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 [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.

#### 4.1 A Tour of XPlainer-Eclipse

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 (more on this on Section /refsec:impl) that uses the WTP XML editor to highlight the XPath path nodes and the XPath selected nodes directly in the XML tree.

In Eclipse, a set of related views is called a perspective. The XPlainer-Eclipse perspective shown in Figure 11 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

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

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

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 node set (the result of the expression), or the actual result when it is not a node set (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. While the formal definition of this parse tree is provided in Section 3, 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 (in the example in the figure the sequence of steps is labeled (1), (2), (3), (4)).

The XPathTreeView is paired with the XML editor view on the right side of the image in Figure 11 that displays a specific instance of the book 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 \text{preceding-sibling :: *}[1]) in the example), in addition to having the same sequence number label (4). 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 (1), (2), and (3)). 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.

XPlainer-Eclipse uses all these matching colored nodes, numeric labels, and icons to provide a visual explanation for the semantics of an XPath query. Not only selected nodes (the result of the XPath expression) are shown, but also the intermediate nodes that contribute step by step to the satisfaction of the expression by the selected nodes. The numeric labels serve an important role disambiguating multiple situations where multiple XPath steps match the same node in the XML document (e.g., the first colored \text{p} node is labeled by (3) and (4))

Our second example XPath expression is

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

which is a slightly modified expression of our previous example (changed by the addition of parenthesis). The explanation of the evaluation of this expression on the book XML document appears in Figure 12 (right-hand side) together with the explanation of the previous example (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.

Figure 12: Explaining the impact of parenthesis

The final example in our tour of XPlainer-Eclipse illustrates how the explanation of (a more complex)
disjunctive XPath expression is visualized. The XPath query

```
//cat : BuyerParty//cat : CityName = 'Elgin'
| //cat : OrderLine//cat : CityName = 'Elgin'
```

whose XPathTreeView (in Figure 13) goes together with the explanation displayed in the XML editor (in Figure 14).

The complete XPath expression that is explained appears in an XPathTreeView node just below the context node. Below this non-colored XPathTreeView node two branches describe the two subexpression that appear in the disjunction, effectively providing an independent explanation for the two disjunctive terms. (Note the use of namespaces in both the document and the expression.)

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 that are not covered in this tour [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).

### 4.2 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.

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
XT tree data structure, an XT/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 $\lambda$ 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 the XPlainer-Eclipse implementation 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 [13], 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 XPath 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 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 XPlanner language in terms of the XPath language itself to evaluate XPlanner queries by invoking an arbitrary (already existing) XPath processor. While this approach to evaluate XPlanner 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 (even implementation-dependent features!).

Finally, we bring attention to the fact that the visualization of answers and intermediate results supported by XPlanner 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 XPlanner 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 XPlanner-specific optimization techniques.

References