Inconsistent Path Detection for XML IDEs

Pierre Genevès
CNRS
pierre.geneves@inria.fr

Nabil Layaïda
INRIA
nabil.layaida@inria.fr

ABSTRACT

We present the first IDE augmented with static detection of inconsistent paths for simplifying the development and debugging of any application involving XPath expressions.

Categories and Subject Descriptors

D.2.3 [**Software Engineering**]: Coding Tools and Techniques—*Program editors*

1. INTRODUCTION

A major challenge is to develop the necessary reasoning technology to provide a safe and secure software engineering environment on which to build the next generation of web applications. To this end, we show how to equip an IDE with static detection of inconsistent paths.

We first introduce the crucial role played by XPath expressions in programs that manipulate XML data in Section 2. In Section 3, we discuss the impact of determining the inconsistency of a given XPath expression. We then present the first augmented IDE with static detection of inconsistent paths as a proof of concept in Section 4 (see the online video demo [4]) before discussing related work in Section 5 and concluding in Section 6.

2. PATHS IN XML PROGRAMMING

XPath is the W3C standard language for expressing traversal and navigation in XML data and documents seen as trees. An XPath expression is a succession of navigation steps separated by "/", where each step is made of an axis and a node test. The axis specifies in what direction to search in the tree for nodes matching the node test that can be a label. The last navigation step performs the selection of nodes that form the result of the evaluation of the entire expression. At each navigation step, nodes can be filtered through the use of qualifiers that can be themselves XPath expressions. For instance, the XPath expression:

```
/descendant::a[following-sibling::b]/child::c
```

navigates from the root of the document to all descendant nodes labeled "a", and retains only those which have at least one following sibling node labeled "b", then it finally selects and returns the set of all children nodes labeled "c" of those "a" nodes. The above expression is absolute since it begins with a leading "/" that specifies that the initial context node for the evaluation of the expression is the root node of the document tree. A relative XPath expression (without the leading "/") can be evaluated from any given context node in the tree. A more formal presentation of XPath expressions can be found in [6]. In this paper, we consider the XPath fragment whose syntax is shown in Figure 1, and that contains all XPath features for navigating forward, backward and recursively through nodes of the document. The fragment also captures the fact that nodes can be filtered using qualifiers, which are boolean expressions between brackets that can test the existence or absence of paths potentially involving attributes.

Since search, selection and extraction of information are essential for any XML processing task, XPath happens to be a core component of XML technologies. In particular, XPath plays a major role in the main XML technologies: XSLT, XML Schema, XForms, and XQuery.

XSLT is the W3C standard language for transforming XML documents into other XML documents. XSLT relies on XPath for selecting parts of the input documents to be transformed. Specifically XPath is used for two different purposes in XSLT: (1) identifying source nodes to which transformation rules apply, by the means of patterns to be matched with nodes of the input document; (2) extracting information, in particular for selecting the next input nodes to be transformed, computing boolean conditions, and producing output text in the result tree.

XML Schema is a W3C standard for the description of document types. An XML schema can be used to express a set of rules to which an XML document must conform in order to be considered *valid* according to that schema. XML Schema provides several XPath-based features for describing uniqueness constraints and corresponding references constraints.

XForms is a W3C standard for the specification of a data processing model for XML data and user interfaces for the XML data, such as web forms. XForms notably uses XPath expressions as queries for addressing and identifying fields within the form and the user-submitted data. In particular, it uses XPath expressions to bind input controls to particular parts of the form's data model.

XQuery is the W3C standard language for querying collections of XML data. XQuery is heavily based on XPath expressions to address specific parts of an XML document. It supplements this with a SQL-like "`FLWOR` expression" for performing joins. A `FLWOR` expression is constructed from the five clauses after which it is named: `FOR`, `LET`, `WHERE`, ORDER BY, RETURN.

In addition, Java and Javascript programs now make extensive use of XPath expressions through the Document Object Model (DOM).

3. PATH INCONSISTENCY

3.1 Causes and Consequences

The XPath language provides a succinct yet very expressive notation for expressing relations between nodes in trees. The possibility of expressing complex paths like first-order definable relations is counterbalanced by the fact that inconsistencies can easily be introduced. A given XPath expression e is said to be *inconsistent* if and only if, for any document (tree), the evaluation of e returns an empty set of nodes. Inconsistent paths are usually caused by at least one contradiction between two navigation steps of the path (either successive or not). Paths contained in qualifiers may also contradict paths used in selection.

Even a consistent path may systematically yield an empty result whenever the set of documents over which the path is supposed to be evaluated is constrained by a schema. A schema defines a restricted set of documents that conform to some constraints, usually expressed by the means of regular expressions. These constraints restrict the admissible elements and the way they can be composed. Widespread notations for schemas include DTD, XML Schema, and Relax NG. We say that an XPath expression e is *inconsistent in the presence of a schema* S if and only if the evaluation of e returns an empty set of nodes for any document valid with respect to S.

Detecting inconsistent paths (in the presence or absence of a schema) is crucial for any static analysis of a host language for XPath such as XSLT, XQuery, XForms or XML Schema. Since XPath plays a central role in all these languages, offering the capability of manipulating and analyzing XPath expressions in programming environments, notably IDEs, simplifies the development and verification of a large class of applications. For instance, inconsistent XSLT patterns or XQuery subpaths mean that dead code can be eliminated: if a path is statically detected as inconsistent, one can avoid evaluating it at runtime in order to save resources. In addition, all code that is supposed to treat the result of an inconsistent path can be safely eliminated. The detection of dead code in XQuery programs [3] is a straightforward application of the detection of inconsistent paths.

3.2 Efficient Logical Resolution

The problem of determining whether a path is inconsistent in the presence of a schema is undecidable in general and is known to be very hard from a computational complexity point of view for core XPath fragments. Specifically, it is an exponential-time decision problem [2, 6] for the fragment of XPath shown in Figure 1 that, in fact, supports all major features of XPath except general counting and general comparisons between data values (that cause the undecidability of XPath inconsistency [2]).

In practice, for performing this path-error analysis, we use the logical solver presented in [6]. The solver takes a given path and a schema, translates them into a logical representation and uses a decision procedure that determines the existence (or inexistence) of a tree (a document) that satisfies both the constraints expressed by the schema and the structural requirements assumed by the path. Specifically, the decision procedure is a logical satisfiability-testing algorithm that looks for a finite tree that satisfies the logical formula combining all the requirements. The decision procedure is sound and complete, which means that the search is exhaustive. One difficult aspect from an algorithmic point of view is that the search universe is very large. For typical cases involving only an XPath expression or an XPath expression and a small schema, the decision procedure performs in several milliseconds. For hard cases involving a highly complex schema like the XHTML DTD, the total number of different tree nodes to be considered is more than the square of the number of atoms in the universe. Even for these cases, it was shown in [6, 5] that the decision procedure performs in less than 3 seconds. This surprising efficiency is partly due to the use of symbolic techniques as well as other advanced implementations techniques and optimizations [6]. The advantage of such an efficiency is that it permits equipping IDEs with automated detection of inconsistent paths when the user clicks a button, as presented in the next section.

$$\begin{aligned}
\textit{path} &::= \textit{step} \mid \textit{path}/\textit{path} \mid \textit{path}[\textit{qualifier}] \\
&\mid \textit{path} \text{ union } \textit{path} \mid \textit{path} \text{ intersect } \textit{path} \\
&\mid \textit{path} \text{ except } \textit{path} \\
\textit{qualifier} &::= \textit{path} \mid \textit{path}/@\textit{test} \mid /@\textit{test} \mid \text{not } \textit{qualifier} \\
&\mid \textit{qualifier} \text{ and } \textit{qualifier} \mid \textit{qualifier} \text{ or } \textit{qualifier} \\
\textit{step} &::= \textit{axis}::\textit{test} \\
\textit{axis} &::= \text{self} \mid \text{child} \mid \text{parent} \mid \text{descendant} \mid \text{ancestor} \\
&\mid \text{following-sibling} \mid \text{preceding-sibling} \\
&\mid \text{following} \mid \text{preceding} \\
\textit{test} &::= \textit{tag} \mid *
\end{aligned}$$

Figure 1: Considered XPath Expressions.

4. AUGMENTED IDE

As a proof of concept, we have integrated this automated analysis inside IDEs. Most notably, we have equipped the XQuery Development Toolkit (XQDT) [1] with capabilities of statically detecting inconsistent paths. XQDT is a plugin for the Eclipse environment that provides support for XQuery 1.1. In particular XQDT provides code completion and code templates, as-you-type validation, and integration with existing XQuery evaluation engines.

4.1 Integration Principle

We have developed a plugin extension that takes an XPath expression e and a schema S as parameters and checks for the inconsistency of e in the presence of S. The analysis functions mark inconsistent path with syntax coloring capabilities offered by the IDE plugin. For this to be possible, the IDE plugin interacts with the plugin extension in the following manner:

1. the abstract syntax tree of the program is first analyzed in order to identify XPath expressions;

2. the evaluation context of each XPath expression is built: because some XPath host languages (like XSLT or XQuery) allow variables to be defined and then used

in paths, this step is necessary for correctly replacing variables occurring in paths by their definition;

3. when the static verification is triggered, each XPath expression and its evaluation context as well as the schema chosen by the programmer are transmitted to the plugin extension;

4. once the analysis is performed the plugin extension returns information (line number, character index) in order to mark inconsistent paths in the user interface.

4.2 Enriched Programming Experience

From the programmer's point of view, the augmented XQDT plugin can be used just as the usual XQDT plugin. The only difference happens when a given XQuery program is opened through the user interface. Two new buttons are then offered to the programmer. The first one allows him to choose a given schema (notice that this is optional: by default no schema is assumed). The second button allows the programmer to trigger the static analysis of paths which marks inconsistent XPath expressions, in the same manner as badly typed Java statements are marked in the classic Eclipse environment for editing Java programs.

The user interface of the plugin extension is shown in Figure 2. The screenshot shows an XQuery example where an XPath expression is automatically identified and marked as inconsistent (independently of any schema). In this case, the XPath expression "`$r/parent::book/author`" is trivially judged inconsistent by the analysis since if we replace "`$r`" by its definition we obtain an expression of the form:

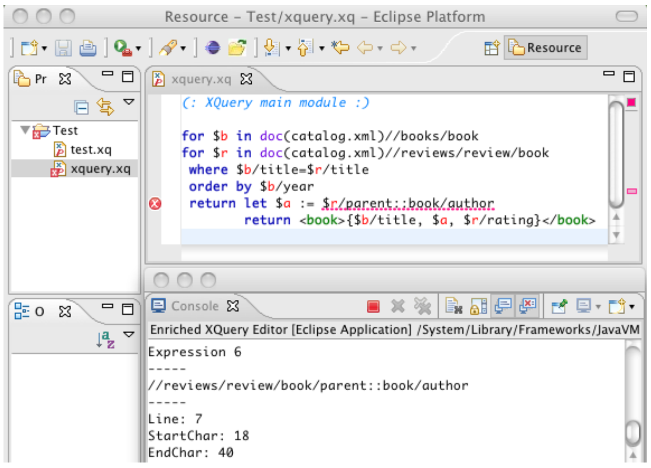
//reviews/review/book/parent::book/author

that at some point attempts to navigate from a node labeled "`review`" to children nodes labeled "**book**" and then going back to the parent node labeled "**book**", which contradicts the previous steps according to which this parent node is labeled "`review`". For this reason, evaluating this path always yields an empty set of nodes, even independently from any schema. General path inconsistencies are not so trivial to detect, especially those involving schema information. This is why inconsistent paths are clearly marked à la Eclipse in the user interface: they are underlined in red and marked with red icons both in the left gutter and next to the scroll bar on the right in order to inform the programmer.

5. RELATED WORK

To the best of our knowledge, our work is the first to provide an IDE equipped with a precise static detection of inconsistent XPath expressions. This work relies on a recent breakthrough [6] and makes use of logical techniques that were tested on real life use cases (such as XHTML and MathML types) and complex queries (involving recursive and backward navigation) [5]. As a consequence, in this context, other IDEs (even supporting syntax verification and/or runtime debugging features) do not match the static analysis precision and capabilities of the work presented here.

We presented a demo in [3] that focused on how to eliminate dead code from XQuery programs based on inconsistent path detection, whereas the present demo focuses on the more general mechanisms for integrating the basic building block for detecting inconsistent paths in any XML IDE. Even if the proof of concept is illustrated through an XQDT


Resource – Test/xquery.xq – Eclipse Platform
Resource
Pr
Test
test.xq
xquery.xq
xquery.xq
O
Console
Enriched XQuery Editor [Eclipse Application] /System/Library/Frameworks/JavaVM

```
(: XQuery main module :)

for $b in doc(catalog.xml)//books/book
for $r in doc(catalog.xml)//reviews/review/book
 where $b/title=$r/title
 order by $b/year
 return let $a := $r/parent::book/author
        return <book>{$b/title, $a, $r/rating}</book>
```

```
Expression 6
-----
//reviews/review/book/parent::book/author
-----
Line: 7
StartChar: 18
EndChar: 40
```

Figure 2: Static Analysis of Paths in Action.

extension, the general mechanism is not tied to XQuery in particular but can be applied to any XML handling program.

6. CONCLUSION

We have presented a generic and basic tool for statically detecting inconsistent XPath expressions. We illustrated how this tool can be integrated inside an IDE for simplifying the development and debugging of any application involving XPath expressions. As a proof of concept, we developed a plugin extension of XQDT that considers XPath expressions as first-class constructs and is capable of underlining inconsistent XPath expressions in the same manner as badly typed Java statements in the classic Eclipse environment for editing Java programs.

As a direction for future work, we plan to investigate further optimizations so that this kind of analyses, despite their high computational complexity, could be provided on the fly when an XPath expression is updated by the programmer.

7. ACKNOWLEDGMENTS

We are grateful to Lorena Freitas who contributed to the implementation of the plugin extension interacting with the XQDT plugin and the logical solver.

8. REFERENCES

[1] XQDT : XQuery development tools, December 2010. http://www.xqdt.org.

[2] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. In *PODS'05*.

[3] P. Genevès and N. Layaïda. Eliminating dead-code from XQuery programs. In *ICSE'10 (demo)*.

[4] P. Genevès and N. Layaïda. Video demo, February 2011. http://www.youtube.com/watch?v=uCFpxTEjj7g.

[5] P. Genevès, N. Layaïda, and V. Quint. Identifying query incompatibilities with evolving XML schemas. In *ICFP'09*.

[6] P. Genevès, N. Layaïda, and A. Schmitt. Efficient static analysis of XML paths and types. In *PLDI'07*.