SPath: A Path Language for XML Schema

Erik Wilde
UC Berkeley

Felix Michel
ETH Zürich

ABSTRACT

XML is increasingly being used as a typed data format, and therefore it becomes more important to gain access to the type system; very often this is an XML Schema. The *XML Schema Path Language (SPath)* presented in this paper provides access to XML Schema components by extending the well-known XPath language to also include the domain of XML Schemas. Using SPath, XML developers gain access to XML Schemas and thus can more easily develop software which is type- or schema-aware, and thus more robust.

Categories and Subject Descriptors: D.3.3 [**Programming Languages**]: Language Constructs and Features — Data types and structures

General Terms: Design, Languages

Keywords: XML, XML Schema, XPath, SPath

1. INTRODUCTION

The *XML Path Language (XPath)* [1] is a language for selecting parts of an XML document. XPath 2.0 extends the data model of XPath 1.0 to not only be derived from a document, but rather from a document being validated and type-annotated by an *XML Schema* [3]. Thus, XPath 2.0 becomes a *typed language* because it provides functionality for working with the *typed content* of an XML document.

Yet XPath 2.0 does not provide functionality for accessing types in the context of the schema. While the structures of an XML document are represented by a tree of interconnected nodes (which can be navigated using *location paths*), there is no such structure for types. Instead, types are identified by their *qualified names (QNames)*, and a rather small number of functions is provided which work with these type identifiers. This makes XPath 2.0 type-aware, but not schema-aware.

This paper introduces the *XML Schema Path Language (SPath)*, which builds on XPath 2.0 in several ways. It extends the data model to contain schema components as navigable structures, and it introduces new axes to navigate them, new node tests to work with them, and additional functions. The goal of SPath is to extend XPath to become a language which not only is well suited for working with XML documents, but also with XML Schemas.

2. PROBLEM

The *XML Information Set (Infoset)* [2] is the data model applications use when working with XML documents. XPath 1.0, which is based on the Infoset, is one of the most successful technologies to provide access to XML structures. It is reused in various contexts, for example *XSL Transformations (XSLT)*, XML Schema, and the *Document Object Model (DOM)*. In all these cases, documents are considered to be trees, and XPath provides access to these trees.

XML Schema turns XML documents into typed documents, with the type annotations being added by the validation process. XML Schema thus made XML more powerful and more complex. XPath 2.0 adds types to its data model, but only as an unstructured set of named items, and with little functionality to use them.

3. SPATH DESIGN

The primary goal of SPath's design is to remain as much within the limits of XPath's design principles and syntax as possible. This is easier said than done, because XPath is not a strict "design by rule" language, but has a lot of design decisions in it which instead are based on usability and utility. For example, the seemingly simple question, what an axis is, is not easy to answer. The most accurate answer probably is "anything that is likely to be used frequently as a way to explore relationships between nodes."

For SPath's design, the goal is to apply the design principles behind XPath to schemas, while still maintaining a dividing line between these two *universes*, so that they are perceived as separate, but interconnected. The principles of node kinds, navigating structures using axes, bidirectional navigation, and node tests as predicate shorthands have been adopted from XPath, but have been extended to cover XML Schema structures as well.

SPath's syntax reuses XPath's syntax wherever possible, and thus from the syntax point of view, SPath expressions have the same structure as XPath expressions (but they can contain different axes, node tests, and functions).

3.1 SPath Data Model

SPath introduces five new node kinds, `schema`, `type`, `declaration`, `occurrence`, and `constraint`. The node kinds are a different view of the structures defined by an XML Schema, instead of being a subset of XML Schema's components. In part, one goal of the data model is to unify elements and attributes. Instead of replicating XML Schema's strong separation of these two concepts, `declaration`s represent element as well as attribute declarations. The same can be said about `occurrence`s, which represent element as well as attribute usages in `type`s. If SPath users wish to make the distinction between elements and attributes, they can use node tests for doing so.

SPath has no direct representation of the model groups of XML Schema, but it exposes this information through axes which provide information about potential neighbors

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

of a node. This means that the grammar information of a schema is preserved in SPath. However, it is not available as the actual grammar, but rather in terms of what the language defined by the grammar is. Specifically, occurrences have the properties `optional` and `unbounded` and refer to a declaration, and XML Schema's content models are mapped to this alternative representation of the grammar. This maps the hierarchical structure of possibly nested model groups to an expanded sequence of occurrences.

3.2 SPath Axes

SPath introduces new axes, which can be used to navigate the schema structures represented by SPath's node kinds. Because a schema is structurally more complex than an XML document, SPath needs more axes for navigating these structures than XPath needs for navigating documents.

```
/descendant::*[42]/type::*
```

This SPath returns the type of the 42nd element in the document, in the form of a `type` node. If it is necessary to get this type's name, SPath supports XPath's `name()` function to also work on `type` nodes:

```
/descendant::*[42]/type::*/name()
```

While the example so far takes nodes from the instance universe and then selects associated nodes in the schema universe, the following example uses another axis which then selects associated node within the schema universe:

```
/descendant::*[42]/type::*/declaration::*
```

In this example, the SPath returns all declarations which are using the type selected in the previous example. These declarations can represent element or attribute declarations (the latter only if the 42nd element has a simple type).

The last example shows how SPath axes help solving problems which would be hard to tackle without this kind of language. The following SPath returns the element declarations of all elements that, according to the Schema, possibly can follow the 42nd element.

```
/descendant::*[42]/followed-by::*/declaration::*
```

It is important to mention that the `followed-by` axis works on the *effective model group*, which also takes into account particles from named groups and parent types. It thus completely covers the range of allowed elements.

Generally speaking, some axes accept node kinds from both universes (for example the `type` axis, which returns the type of an element or attribute in an instance, or of a declaration or occurrence in a schema), but all of them always return node kinds only belonging to one of the universes.

As in XPath, axes navigating through hierarchical structures have variations, providing functionality for single steps (`basetype`/`derivedtype`), recursive navigation (`supertype`/`subtype`), and recursive navigation including the context node (`supertype-or-self`/`subtype-or-self`).

3.3 SPath Node Tests

Node tests in XPath can either be *name tests* or *kind tests*. SPath follows this principle. Since the data model of XML Schema (and thus the data model of SPath) contains *unnamed* nodes, the semantics of the wildcard have been extended to select unnamed nodes as well. XPath defines *kind tests* which cover all *node kinds* encountered in XPath. Likewise, SPath provides a set of *kind tests* for each of the SPath node kinds that are described in Section 3.1.

The kind tests for nodes that can be *named* (i.e., `type()`, `declaration()`, and `constraint()`) accept a first argument that can either be a QName or a wildcard. If the wildcard is specified, *anonymous* nodes are returned as well.

3.4 SPath Predicates

SPath does not change the semantics of XPath predicates, they are evaluated in the usual way: for each item in the sequence produced by the expression preceding the predicate list, each predicate is evaluated with this item as the context, and only if all predicates evaluate to true, the item remains in the final result sequence of the step.

This means that predicates in SPath expressions can use the full set of XPath expressions as predicates. An important task of predicates, however, is the filtering of nodes based on certain criteria which in many cases are specific to the node kind. Because SPath defines a number of new node kinds, it also defines a number of functions which allow the filtering of these node kinds in predicates.

3.5 SPath Functions

A wide range of the functionality required for working with SPath is provided by the SPath axes as described in Section 3.2, and by the node tests introduced in Section 3.3. The extent to which such functionality should be covered by functions or rather by dedicated syntax constructs like axes is a question of language design.

SPath follows the principle of defining functions only for information that is either a literal property (rather than a structural) or where the function requires more or different arguments other than the context node. Since many functions like `name()` or `namespace-uri()` are semantically equivalent to the corresponding XPath functions, the respective XPath functions are extended to polymorphic functions accepting nodes from both universes.

4. CONCLUSIONS

The main contribution of SPath to the evolving landscape of XML technologies is the integration of schemas into the data model of XPath. Additionally, SPath's expressive syntax allows the easy navigation of the complex structure defined by an XML Schema. Using SPath's navigational features, applications can explore schemas at runtime and thus be programmed in a way which better supports loose coupling scenarios of XML-oriented software components.

5. REFERENCES

[1] Anders Berglund, Scott Boag, Donald D. Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. XML Path Language (XPath) 2.0. World Wide Web Consortium, Recommendation REC-xpath20-20070123, January 2007.

[2] John Cowan and Richard Tobin. XML Information Set (Second Edition). World Wide Web Consortium, Recommendation REC-xml-infoset-20040204, February 2004.

[3] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition. World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, October 2004.