XML-Based XML Schema Access

Erik Wilde
UC Berkeley

Felix Michel
ETH Zürich

ABSTRACT

XML Schema's abstract data model consists of *components*, which are the structures that eventually define a schema as a whole. XML Schema's XML syntax, on the other hand, is not a direct representation of the schema components, and it proves to be surprisingly hard to derive a schema's components from the XML syntax. The *Schema Component XML Syntax (SCX)* is a representation which attempts to map schema components as faithfully as possible to XML structures. SCX serves as the starting point for applications which need access to schema components and want to do so using standardized and widely available XML technologies.

Categories and Subject Descriptors: D.2.10 [**Software**]: Software Engineering — Design — Representation; H.3 [**Information Systems**]: Information Storage and Retrieval

General Terms: Design, Standardization, Documentation

Keywords: XML, XML Schema, SCX

1. INTRODUCTION

XML Schema [4] is today's most important XML schema language. It is more powerful than DTDs, but also is more complex and less easy to handle. For example, while DTD authors often work with DTDs in their textual form, XML Schema's syntax is harder to use and thus alternative interfaces (such as graphical editors) are more frequently used.

At first sight, XML Schema's XML syntax might seem like a syntax that is not ideal for humans, but good for machines, but on closer inspection it can be seen that the syntax is a hybrid between ease of use for humans, and ease of use for machines. Writing code which works on XML Schema's XML syntax and should in fact work on the underlying schema components is not easy. In this paper, we present the *Schema Component XML Syntax (SCX)*, which is designed to be as faithful a XML-oriented representation of XML Schema components as possible.

SCX explores the side of the syntax spectrum which should be ideal for machine interpretation. XML Schema's XML syntax is somewhere in the middle, and a non-XML DTD-like syntax [5] is on the other side of the spectrum. SCX attempts to deliver the promise of true XML Schema access using standard XML technologies, instead of having to provide specialized APIs [2] (currently, there is no standardized API for XML Schema) for working with XML Schemas.

2. USE CASES

Working with the information contained in XML Schemas can be useful in a variety of scenarios, and we briefly describe some of these uses cases here. So far we have only explored one of these areas writing code based on SCX.

User Interface Generation based on XML Schemas and on declarative UI languages such as XForms can traverse the structures of an XML Schema and generate a UI for the schema. This process would probably require additional information for improving the UI, but if such information is available in XML Schema's annotations in a machine-readable way, this information could also be accessed while processing the schema.

Versioning-Aware XML Processing adds schema access to XML processing code and thus adds the possibility to inspect the schema of an instance before processing it. Based on annotations (such as `mustUnderstand` or similar concepts), the code can then automatically cast the instance to a different version (or generate an error if this is not possible). In this scenario, schema access is only required in the adaptation layer between instance parsing and application processing.

Composite Schemas is the idea of embedding additional information in schema annotations which cannot be formally expressed in XML Schema. While for example the combination of the grammar features of XML Schema and the rule-oriented approach of Schematron [1] are a useful combination, the question remains how to author and maintain such a "composite schema" and how to use it for validation. SCX can be used to automatically extract a Schematron schema from an XML Schema, which means that the "two schemas" are always aligned.

Best Practices are often defined in XML Schema user groups to limit the variability allowed by XML Schema. SCX can be used to define these best practices in a more formal way, in the simplest case by defining a Schematron schema for SCX documents.

Canonicalization and **Normalization** are areas which are comparable to the best practices checking described above, but instead of just checking a schema for conformance with a set of rules, they transform the input to adhere to a set of rules. Canonicalization often is used to term some set of rules, whereas normalization usually refers to a specific set of rules which, in terms of the affected data model, leads to a model which is in its most simplified form.

Schema Documentation is the process of producing human-readable documentation for an XML Schema. There are several tools on the market which generate schema documentation. Using SCX and XSLT, we are currently working on *X2Doc* [3], a flexible and extensible documentation tool. The advantage of X2Doc is that it uses standard XSLT templates, which means that it is open and easily extensible.

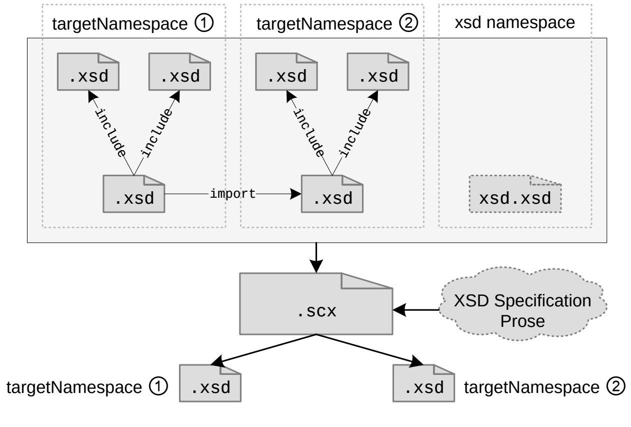
WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

```
<scx:type-definition uid="d11e34" document="ipo.xsd" path="schema[1]/complexType[2]" position="13">
 <scx:name>RestrictedPurchaseOrderType</scx:name>
 <scx:target-namespace>http://www.example.com/IPO</scx:target-namespace>
 <scx:base-type-definition>d11e18</scx:base-type-definition>
 <scx:category>complex</scx:category>
 <scx:derivation-method>restriction</scx:derivation-method>
 <scx:abstract>false</scx:abstract>
 <scx:attribute-uses>
  <scx:attribute-use>
   <scx:required>false</scx:required>
   <scx:attribute-declaration>d11e31</scx:attribute-declaration>
  </scx:attribute-use>
 </scx:attribute-uses>
 <scx:content-type>
  <scx:variety>element-only</scx:variety>
  <scx:particle path="schema[1]/complexType[2]/complexContent[1]/restriction[1]/sequence[1]">
   ...
  </scx:particle>
 </scx:content-type>
</scx:type-definition>
```

3. SCX DESIGN

SCX is as closely aligned with XML Schema's abstract data model as possible. This makes it much easier to access information on the component level than in the XML syntax. For example, it is not trivial to determine what the effective type of a derived complex type is. The reason for this is that both restriction and extension work in a way which sometimes require to collect information along the complete derivation chain. In addition, the rules how the effective type is determined are quite complex, for type restriction for example attributes which are not repeated are part of the restricted types, but attribute wildcards are not.

The above SCX example shows a small part from the example schema of the XML Schema primer. The attribute use is part of the `RestrictedPurchaseOrderType`, even though the type definition does not explicitly contain this attribute, it is inherited from the supertype. Generally, SCX makes the structure of an XML Schema more accessible by merging several sources of information, as shown in the following figure, and representing them in an easily accessible way.



SCX takes various input sources; the schema documents for a schema, any imported schemas, the schema for schemas (which contains the type definitions for the built-in types), and information from the specification which is not formally defined (for example the fact that list types always have a fixed `whitespace="collapse"` facet).

If applications use SCX for schema manipulations, the modified SCX can be transformed to the XML syntax of XML Schema in a final step, so that the schema can be used as input for schema-aware applications.

In some cases, components do not capture all information in an XSD document. For example, in XSD markup, type definitions can contain annotations in various locations. Schema components do contain all those annotations as a set, but they do not preserve the original location of each annotation. This might lead to problems, because location information for annotations often is semantically relevant. SCX therefore preserves this information in attributes.

4. CONCLUSIONS

Even though XML Schema is a core component of the XML technology landscape today, its essence so far is surprisingly inaccessible using standard XML technologies. SCX enables XML-based access to XML Schemas, and thus opens its data model to XML technologies, most notably XPath and thus XSLT and XQuery. XML-centric applications are often built with the assumption that the underlying schema will not change, but when moving towards a more flexible and robust approach where schemas can change, technologies for providing access to schemas are critically needed.

5. REFERENCES

[1] International Organization for Standardization. Information Technology — Document Schema Definition Languages (DSDL) — Part 3: Rule-based Validation — Schematron. ISO/IEC 19757-3, April 2006.

[2] Elena Litani. XML Schema API. World Wide Web Consortium, Member Submission SUBM-xmlschema-api-20040309, March 2004.

[3] Felix Michel and Erik Wilde. Extensible Schema Documentation with XSLT 2.0. In *Poster Proceedings of the 16th International World Wide Web Conference*, Banff, Alberta, May 2007. ACM Press.

[4] 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.

[5] Erik Wilde and Kilian Stillhard. Making XML Schema Easier to Read and Write. In *Poster Proceedings of the Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003.