# XML Validation: Looking Backward

## Strongly Typed and Flexible XML Processing are not Incompatible

Pierre Genevès
CNRS
pierre.geneves@inria.fr

Nabil Layaïda
Inria
nabil.layaida@inria.fr

## ABSTRACT

One major concept in web development using XML is validation: checking whether some document instance fulfills structural constraints described by some schema. Over the last few years, there has been a growing debate about XML validation, and two main schools of thought emerged about the way it should be done. On the one hand, some advocate the use of validation with respect to complete grammar-based descriptions such as DTDs and XML Schemas. On the other hand, motivated by a need for greater flexibility, others argue for no validation at all, or prefer the use of lightweight constraint languages such as Schematron with the aim of validating only required constraints, while making schema descriptions more compositional and more reusable.

Owing to a logical compilation, we show that validators used in each of these approaches share the same theoretical foundations, meaning that the two approaches are far from being incompatible. Our findings include that the logic in [2] can be seen as a unifying formal ground for the construction of robust and efficient validators and static analyzers using any of these schema description techniques. This reconciles the two approaches from both a theoretical and a practical perspective, therefore facilitating any combination of them.

## Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation—*markup languages*; D.3.2 [**Programming Languages**]: Language Classifications—*extensible languages*

## Keywords

XML, validation, schemata, foundations.

## 1. INTRODUCTION

Several programming languages have been proposed to offer native language support for the very popular XML format for the processing and exchange of tree-structured data. In particular, languages with built-in XML type systems have been proposed (XDuce, CDuce, C#...) with the goal of offering static analysis capabilities in compilers. Such capabilities allow guaranteeing, at compile time, that generated trees conform to a given type.

In the recent years, XML use in applications and systems increased significantly and spread rapidly to different areas with various expectations (Databases, Programming Languages, Web Application Development...). Driven by these different needs, XML type systems also evolved in standard bodies (W3C, Oasis) and various communities. In particular, Document Type Definition (DTD) have been superseded by richer type systems such as XML-Schema, DSD, and Relax NG. Beside the introduction of more elaborate type constructions, these type systems share common principles in that types are described mainly thought structural constraints: constraints on parent and sibling tag names described by regular expressions.

Some XML practitioners argue that such type descriptions and the subsequent validation process are not suitable for a wide range of applications where greater flexibility is needed. For example, applications with rapidly changing types or even loosely known types are needed in order to accommodate more data sharing between applications or application versions. One of the main issues raised is related to XML validation which prevents, in the case of failure, any additional processing of documents. In particular, applications interested in some portions only of the type or with a slightly modified version of the initial type. As a consequence, types expressed by means of sets of constraints between possibly distant elements in the tree have been advocated as a good alternative to schema validation. Some others find in constraint oriented validation a complement to plain XML validation. It is the case of DSDL (Document Schema Description Languages), an ISO standard, which has been designed to allow both kinds of validation, through Schematron [3].

## 2. LIGHTWEIGHT VALIDATION

Schematron [3] focuses on validating documents using tree patterns or paths instead of regular expressions. The language definition is simple enough to yield very compact descriptions, yet provides very powerful constraint specification via XPath [1]. The Schematron language differs from most other XML schema languages in that it is a rule-based language that uses regular path-expressions instead of regular expressions. This means that instead of relating sibling nodes horizontally it makes assertions applied to a specific context within the document to elements deeper in the tree or vertically. This approach allows many kinds of structures to be represented which are inconvenient and difficult in grammar-based schema languages. The general syntax and semantics of Schematron language constructs is defined in the standard specification [3]. Basically, Schematron allows to describe and mix two main kinds of constructs: (i) `report` elements allow to diagnose which variant of a language

*WWW 2013 Companion,* May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2038-2/13/05.

you are dealing with; and (ii) `assert` elements allow to confirm that the document conforms to a particular constraint. The Schematron defines a validation process consisting of two stages: first, find context nodes in the document (typically an element) based on XPath path criteria; and then, check if some other XPath expressions are true, for each of those nodes. For example, the following Schematron schema is composed of two rules:

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
 <sch:pattern  id="p1">
   <sch:rule context="self::a">
     <sch:report test="descendant::a">Error</sch:report>
   </sch:rule>
   <sch:rule context="table">
     <sch:assert test="count(descendant::td)>1">OK</sch:assert>
   </sch:rule>
 </sch:pattern>
</sch:schema>
```

The first rule ensures that anchors elements `a` are not nested, and the second rule guarantees that `table` elements contain at least two cells (`td` elements).

The advantage of such a description is that it is much more succinct than a grammar-based specification of the same constraints[1]. Typically, this Schematron can be used alone if an application only requires these two constraints to be satisfied. It can also be used to further restrict documents after a first validation step has been done using a DTD or XML Schema for XHTML for instance. Several Schematron implementations (validators) exist for this purpose [3].

## 3. STATIC ANALYSIS

Succinct descriptions of constraints and flexible validation constitute undeniable strengths of Schematron, as briefly illustrated by the above example. However, hitherto, no static analysis technique exist for Schematron: suprisingly, the problem of comparing Schematron descriptions (determining containment and equivalence for Schematron) has not been addressed in the litterature. This is even more surprising when compared to the heavily researched topic of determining containment and equivalence for grammar and automata-based specifications (see XDuce and the follow-up works for instance). We fill this gap: we propose the first method for static analysis of Schematron descriptions. It consists in three steps:

1. First, a given Schematron description is compiled into the logic of [2]. This logic is already known to capture regular tree grammars and XPath queries [2]. We implemented a compiler for Schematron by building on top of the XPath compiler introduced in [2]. This compiler takes any Schematron construct and translates it in terms of a corresponding logical formula, respecting the semantics of Schematron described in [3].
2. Second, we formulate the problem we are interested in solving (involving the compilation of one or possibly several schematron and/or DTDs, XML schemas) into a logical formula;
3. Finally, we use the satisfiability solver of [2] to check the logical formula for satisfiability.

Owing to this approach, we can do all sorts of analyses that were not possible before. Such analyses can roughly be divided in two categories:

1. **Intra-Schematron static analyses**: these analyses focus on a single Schematron description with the aim to optimize it. For instance we can automatically detect and remove unreachable rules by comparing patterns and rule positions. We can also automatically reorder rules to avoid useless computations. Finally, we can check for potential errors in the description, by testing for coverage of rules and detecting potential missing cases.
2. **Inter-Schematron static analyses**: these analyses consider several Schematron descriptions and even possibly other regular grammar-based descriptions (such as DTDs or XML Schemas or Relax NG schemas). The goal here is to compare descriptions to check whether constraints expressed by one schema are implied by the constraints described in another one. This is essential for checking containment, equivalence, forward and backward compatibility as well as performing redundancy tests. Another application is to detect potential contradictions stemming from separate schemas. For instance, by checking for satisfiability of the first Schematron rule of the above example in the presence of the XHTML DTD, we observed that this DTD does not prevent the nesting of HTML anchors (although this is semantically forbidden by the recommendation).

## 4. CONCLUSION

We have built a compiler that translates Schematron rule-based constraints into a unifying logic, known to capture regular tree grammars (à la DTDs, XML Schemas and Relax NG). This makes it possible to use existing satisfiability solvers such as the one found in [2] to build static analyzers processing Schematron descriptions.

Our contributions are thus twofold: (1) We provide the first static analysis technique for Schematron; and (2) amid a growing debate that apparently opposes two approaches (strong grammar-based typing vs. lightweight and flexible rule-based typing), we exhibit a common formal ground, based on logic, that is capable of representing both kinds of constraints. Satisfiability solvers for this logical representation can be used to implement static analyzers capable of supporting both rule-based descriptions (à la Schematron) and also grammar-based specifications (à la DTD).

We believe this constitutes an important step in the finding of unifying formal models for representing the various kinds of structured web data constraints, and benefit from the advantages of distinct meaningful approaches. As a future work, it would be interesting to implement common validators by investigating the use of model-checkers for the underlying unifying logic.

## 5. REFERENCES

[1] J. Clark and S. DeRose. XML path language (XPath) version 1.0, W3C recommendation, November 1999.

[2] P. Genevès, N. Layaïda, and A. Schmitt. Efficient static analysis of XML paths and types. In *PLDI '07*, pages 342–351, 2007.

[3] ISO/IEC. Document schema definition language – schematron. http://www.schematron.com, 2012.

[1] Noticeably, the XHTML working group did not syntactically encode the constraint prohibiting the nesting of anchors in the XHTML DTD because this constraint would have been too cumbersome to describe using the DTD grammar-based formalism.