SchemaScope: a System for Inferring and Cleaning XML Schemas

SchemaScope: a System for Inferring and Cleaning XML Schemas

Geert Jan Bex Frank Neven Stijn Vansummeren*

Hasselt University and Transnational University of Limburg, Belgium
firstname.lastname@uhasselt.be

ABSTRACT

We present SchemaScope, a system to derive Document Type Definitions and XML Schemas from corpora of sample XML documents. Tools are provided to visualize, clean, and refine existing or inferred schemas. A number of use cases illustrates the versatility of the system, as well as various types of applications.

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Languages, Theory

Keywords

XML, regular expressions, schema inference

1. INTRODUCTION

Although the presence of a schema enables many optimizations for operations on XML documents, a recent study of Barbaosa et al. [4] has shown that approximately only half of the XML documents available on the World Wide Web refer to a schema. Even worse, about two-third of those schemas are syntactically or semantically incorrect, as shown by Sahuguet [14] and Bex et al. [6, 12].

In this demo we present the tool SchemaScope to address these issues. SchemaScope supports (1) the automatic inference of Document Type Definitions (DTDs) and XML Schema Definitions (XSDs) from corpora of sample XML documents and (2) tools to visualize, clean, and refine existing or inferred schemas.

Schema inference. The automatic inference of schemas from a corpus of XML documents is particularly useful in those situations when no existing schema is available. Although several command-line tools are available for this task [5, 7, 9, 10] the quality of the inferred schema is always heavily dependent on the quality of the XML corpus: When the corpus does not completely cover the intended schema the inferred schema may be too specific; when the corpus contains errors the inferred schema may be too general. For this reason, SchemaScope not only allows automatic inference of DTDs and XSDs (using the algorithms developed in our earlier work [5, 7]), but also provides appropriate visualization tools to allow a human expert to fine-tune the inferred schema based on the actual corpus. Furthermore, as more sample documents become available, the schema can be evolved to capture the corpus more precisely.

Schema cleaning. A related, but distinct setting is the one where one has a corpus of XML documents, as well as an existing schema that is supposed to describe it, but for which some documents fail to validate. In order to use the schema to aid in further efficient processing or querying of the corpus, it is then desirable to clean the schema based on the corpus. SchemaScope supports such cleaning by allowing users to interactively relax the content models of the original schema, at each step showing the parts of the corpus that become valid during this relaxation, and measuring how many fragments in the corpus actually necessitate the relaxation. The latter helps in rejecting those schema changes which are due to errors in the corpus.

Schema refinement. A final setting is where one has a corpus of XML documents, as well as an existing schema that describes the corpus, but where the schema is too general in the sense that some parts of its content models are never realized in a document. Since schemas that better describe the true structure of the data provide more information to be exploited with regard to storage and query optimization, it is in this case preferable to refine the existing schema. SchemaScope supports such refinement by visualizing the support each content model part has in the corpus, and by indicating those XML document fragments that become invalid when the user modifies the schema.

In short, SchemaScope is a tool intended to complement existing high-quality schema editors like Stylus [1] and oXygen [2]. In the following section we describe the working of the system in more detail. We present an overview of the demonstration in Section 3.

2. SYSTEM COMPONENTS

SchemaScope consists of a number of interacting modules that provide the required functionality. A schematic

*Postdoctoral Fellow of the Research Foundation - Flanders (FWO).

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.
ACM 978-1-60558-102-6/08/06.

overview of the application's modules and their interactions is shown in Figure 1.

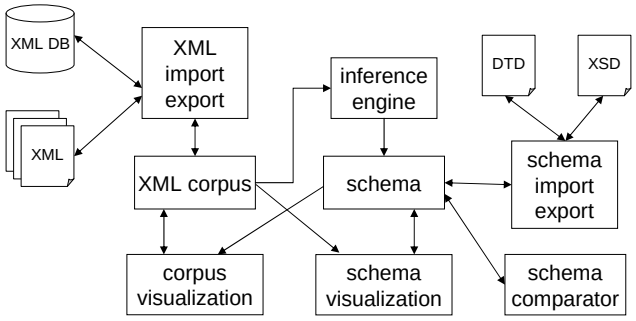

Figure 1: Schematic overview of the SchemaScope system's components. Figures 2 and 3 show details of the schema visualization component.

Import and export. XML documents or fragments of such documents can be imported (depicted by "XML import" in Figure 1) from a range of data sources including files, URLs, and XML database query results. XML schemas in DTD or XSD syntax can be imported ("schema import" in Figure 1) and are converted to an internal representation, simply denoted by the term "schema". A derived or modified schema can be saved as a DTD or XSD (in the former case with potential loss of precision).

Schema inference engine. The inference module generates a DTD or XSD from the imported XML corpus using the algorithms presented in previous work [5, 7]. Although these algorithms cannot infer every possible target DTD or XSD (a classical result of Gold [11] states that no such algorithm exists), they can infer those subclasses of DTDs and XSDs that are used in practice [5, 7]. Furthermore, the inferred content models are always concise and human-readable.

The user actually has a choice between several algorithms for the inference of complex type content models. Some algorithms work well when only a small number of sample documents are available, while others yield better content models, but require more data. Simple type content models (like int, base64, string, ...) and types of attributes are inferred based on a number of heuristics.

Schema visualization. Imported or inferred schemas can be visualized by the schema representation module. A textual, outline, or graph view (cf. Figure 2) is provided to inspect the ancestor relationship between the elements in the schema, while content models can be viewed as text, e.g.,

```
annotation?  (attribute | attributeGroup)*
   anyAttribute?
```

or as hierarchical graphs (cf. Figure 3). All views are annotated and color coded with frequency information calculated from the imported XML corpus. Parts of content models that are realized by only few members of the corpus are brightly rendered, so as to call attention to opportunities for schema cleaning or refinement. Using these views the

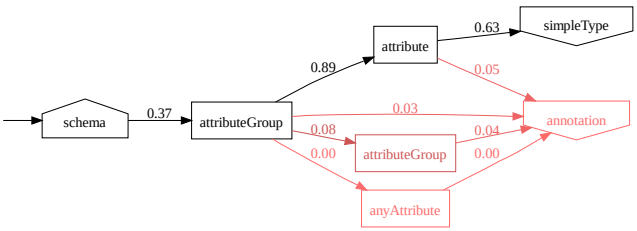

Figure 2: Schema view centered on the XML Schema for XSD's `attributeGroup` element, showing its parent (`schema`) and children (`annotation`, `attribute`, `attributeGroup`, and `anyAttribute`). Rectangular elements are fully expanded, upward pointing elements are partially expanded, and downward pointing elements are not expanded. Numbers and colors indicate the elements' support in the corpus.

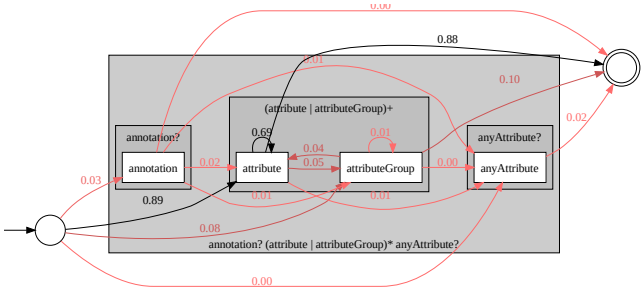

Figure 3: Content model view of the XML Schema for XSD's `attributeGroup` element, shown here as a fully expanded deterministic finite automaton. As in Figure 2, numbers and colors show the support for the syntactic structure of the model.

schema can be edited, while the impact of the changes on the frequency information can be used as feedback and guidance. An additional view provides a number of metrics of the schema [8, 6, 13].

XML data visualization. Individual imported documents or fragments can be viewed using the corpus visualization module. More importantly, the corpus as a whole can be visualized in a list layout based on its properties with respect to the current schema. One can view the data that realizes a particular content model part in the schema or one can view and inspect those XML documents or fragments that are no longer valid after the user has modified the schema. The XML documents can also be ranked in the list according to one of several measures that quantify the degree of conformance to the schema. In this view a cut-off can be set so that one obtains subsets of the corpus. Those can subsequently be used to attempt to infer more appropriate schema for the individual subsets. An additional view provides metrics and statistics of the corpus.

Schema comparator. During schema refinement or correction, it is important to be able to compare the obtained schema with its previous versions. This module allows semantic containment tests of individual content models and even whole schemas. An example is given in Figure 4 where two content models are compared; differences are shown us-

ing colors.

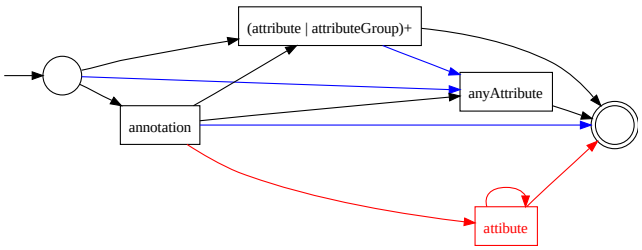

Figure 4: Schema comparator focussed on two content models for `attributeGroup`, syntactic structures present in both models are shown in black, those only present in either one in red or blue.

XML generation. Finally, we come full circle with the XML generator module that can synthesize a collection of XML documents valid to the specified schema. Such a corpus can be used, e.g., to test external applications that use the schema. The generation process is parametrized in order to allow fine-tuning to many requirements. It should be noted that the generator's expressive power complements that of ToxGene, the current state of the art [3]. Whereas ToxGene strictly respects the metrics specified by the user, thus potentially generating invalid documents, our implementation yields documents that are guaranteed to be valid.

3. DEMO OVERVIEW

In the demonstration, we focus on a number of real-world use cases that serve as motivating examples for the features implemented in SchemaScope and illustrate its versatility.

Schema inference. For the purpose of demonstrating the automatic inference of schemas, we use a corpus of XSD documents; infer a schema from it; and compare this inferred schema with the actual XSD for XML Schema Definitions [15]. Aided by the various schema visualization views, we refine the inferred schema based on the semantics implied by the corpus' ontology and frequency information. Figure 5, for instance, shows the content model for the `attributeGroup` element inferred from incomplete and noisy data. Using this view of the content model, it is immediately clear to a human expert that the element `attibute` (versus `attribute`) is due to noise. Finally, we save the resulting schema as an XSD using the schema export component to illustrate that the result is concise and human readable.

Schema cleaning. To demonstrate schema cleaning, we have harvested a corpus of real-world XHTML documents from the World Wide Web (all on a specific topic) that—although well-formed—are not all valid according to the specified XHTML DTD. Starting from this real-world corpus and the W3C XHTML DTD specification, we derive a more relaxed DTD that validates a larger fraction of the corpus, while still rejecting those documents that deviate too much from the specification. Crucial in this relaxation is the XML data visualization component that allows us to determine the balance between the number of valid documents and the precision of the schema. Furthermore, the XML schema visualization component (cf. Figure 5) helps in identifying noisy data.

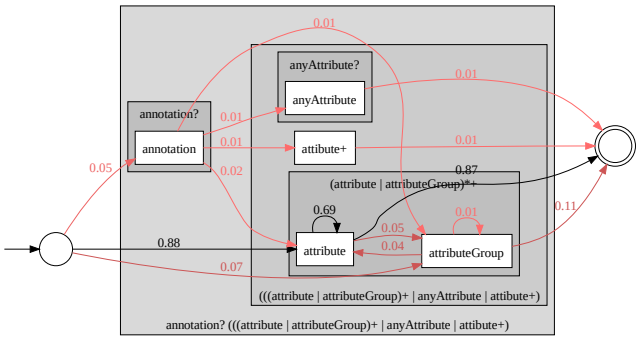

Figure 5: Content model for `attributeGroup` derived from incomplete and noisy data.

Schema refinement. For the purpose of demonstrating schema refinement, we consider Microsoft's WordML document format. WordML specifies the syntax of Microsoft Word documents in XML form. While developing a converter from some arcane and obsolete document format to WordML, we noted that some documents that were valid according to WordML's XSD were nevertheless incorrectly processed by Microsoft Word. The WordML XSD is in fact too general since a number of syntactic constraints are coded in the application's logic, rather than in the documents' specification. In this part of the demonstration, we therefore import a corpus of WordML documents and the XSD provided by Microsoft and gradually refine the schema to capture some of the logic that is imposed by the application, but that is not integrated into the original schema. The schema comparator is used to visualize the differences between the refined and the original content models.

4. REFERENCES

[1] Stylus Studio. `http://www.stylusstudio.com/`.

[2] `<oXygen/>` XML editor and XSLT debugger. `http://www.oxygenxml.com/`.

[3] D. Barbosa, A. O. Mendelzon, J. Keenleyside, and K. A. Lyons. ToXgene: an extensible template-based data generator for XML. In *WebDB 2002*, pages 49–54, 2002.

[4] D. Barbosa, L. Mignet, and P. Veltri. Studying the XML Web: gathering statistics from an XML sample. *World Wide Web*, 8(4):413–438, 2005.

[5] G. J. Bex, F. Neven, T. Schwentick, and K. Tuyls. Inference of concise DTDs from XML data. In *VLDB 2006*, pages 115–126, 2006.

[6] G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: a practical study. In *WebDB 2004*, pages 79–84, 2004.

[7] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML Schema Definitions from XML data. In *VLDB 2007*, pages 998–1009, 2007.

[8] B. Choi. What are *real* DTDs like? In *WebDB 2002*, pages 43–48, 2002.

[9] J. Clark. Trang: Multi-format schema converter based on RELAX NG. `http://www.thaiopensource.com/relaxng/trang.html`, June 2003.

[10] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: learning document type descriptors from XML document collections. *Data mining and knowledge discovery*, 7:23–56, 2003.

[11] E. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, May 1967.

[12] W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and Complexity of XML Schema. *ACM TODS*, 31(3):770–813, 2006.

[13] A. McDowell, C. Schmidt, and K. bun Yue. Analysis and metrics of XML Schema. In *Software Engineering Research and Practice*, pages 538–544, 2004.

[14] A. Sahuguet. Everything you ever wanted to know about DTDs, but were afraid to ask. In *WebDB 2000*, pages 69–74, 2000.

[15] H. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema part 1: structures*. W3C, May 2001.