jInfer XML Schema Inference Framework

Michal Klempa, Mário Mikula, Robert Smetana, Michal Švirec, Matej Vitásek
Advisors: RNDr. Irena Mlýnková, Ph.D., Martin Nečaský, Ph.D.

Praha, 2011

Abstract

At present day, there are many algorithms for solving the problem of XML schema inference. However, virtually none of them is widely used by the public to solve this task in practice. This is partly because of the lack of a simple user interface to these algorithms. We present a NetBeans ([net]) based pluggable framework to fill this gap. This way we enable people to deal with XML in practice, to obtain schemas for their documents by following a few simple steps, without the need to read and understand theoretical aspects. We hope our solution will encourage the usage of XML schemas in practice, since the creation of a schema for existing sets of documents will be more affordable (one doesn't have to write it by hand). We suggest that our framework can also be used as an experimental sandbox for scientific developers, willing to test their new algorithms without the need to develop a complicated GUI.

Introduction

Although many algorithms (for example those proposed by [Aho96, BNST06, BNV07, VMP08, Vyh]) try to solve the problem of schema inference for an existing set of XML documents, have you ever tried to solve the problem in practice?

When the problem arises, one wants to find a solution as quickly as possible, without the need to investigate many scientific papers. We focus on this group of potential users - programmers, system administrators, XML coders, XML maintaners in private and corporate R&D.

Our framework is designed to be extensible and is implemented as a set of modules for NetBeans platform. This means that the second group of users of our framework - the researchers willing to experiment with their new algorithms - may easily gain benefits from the user interface we provide. Thus, speeding up their development, making easier the comparison with other algorithms already developed for the framework.

Nevertheless, it is not all about the user interface. We provide XML/XSD/DTD import and export modules that are ready to use. We implement a sample inference algorithm (see [Aho96]). We provide visualization tools to display input XML documents as a set of grammar rules, and tools to visualize non-deterministic finite automatons used in many of the inference algorithms. This way, extending existing NFA base algorithms with user interaction is easier for researchers.

Related work

Most of the algorithms mentioned don't have an implementation publicly available, nor are they ready to use. Probably the best solution nowadays is [BNV08], dealing with schema inference and evolution. It employs user interaction, schema visualization and user-aided schema refinement to obtain better results. Authors use their own algorithms for automatic schema inference ([BNST06, BNV07]) and then present a user generated schema with optional GUI to refine it. Schema refinement and visualization are strong points of their solution. Also, the possibility to do schema evolution (given old schema and not-all-valid XML documents, to update the schema) is unique in the field. Since we are both trying to solve (almost) the same problem, we predict that we will approach some good ideas from other known solutions in our further work. There is no mention of extensibility or pluggable design, which we, on the other hand, consider our top priority.

Short architecture overview

We divide the inference process into three steps (as illustrated in figure 1):

1. Import of various formats (XML, XSD, DTD, XPath, etc.) into an internal representation - *Initial Grammar*.
2. Simplification (generalization) of *Initial Grammar* with optional help of user interaction.
3. Export into various schema formats (XSD/DTD/...).

Our internal representation is a regular grammar, i.e. a list of rules. We represent the right side of grammar rule as a regular expression with our own set of classes to represent regexps.

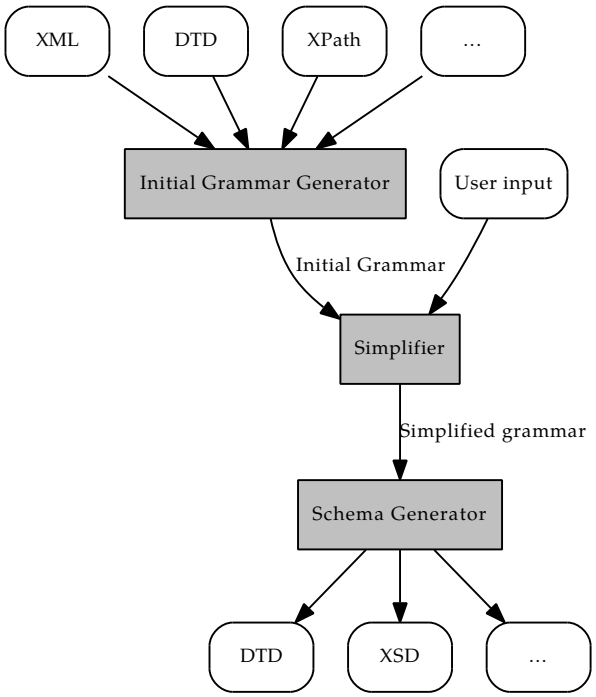

Figure 1: High-level view of the inference process

On input, all various formats are at first converted into internal representation - from XML, rules are extracted as positive examples, concatenations; from XSD/DTD, rules are extracted as they are written in the schema. From XPath we extract only explicit concatenations for now.

Rules are sent to `Simplifier` module, which has to do the simplification algorithm and return a *Simplified Grammar*, which is then easily exported by `Schema Generator` modules.

Simplification can be either automatic, or user-aided. We provide visualization of non-deterministic finite automata. This helps algorithms that use merging state algorithms with user interaction. Using simple API, algorithm developer can call our library to present the automaton to the user to make human decisions and continue in simplification process. See figure 2 for an automaton visualization tool.

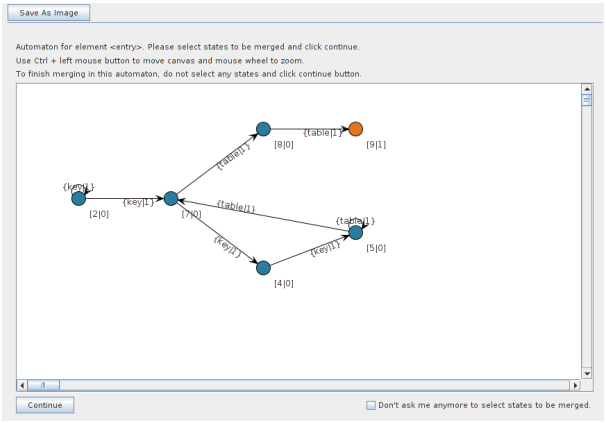

Figure 2: Visualization of automaton in inference process

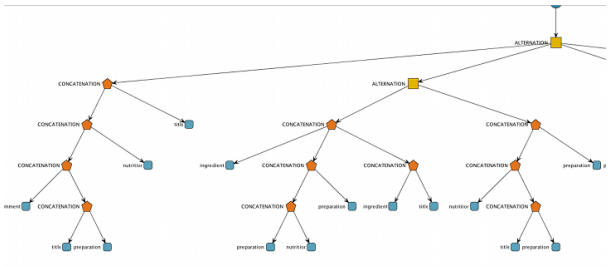

Figure 3: Visualization of a grammar rule

We provide visualization of rules themselves, which is useful for methods not using automata at all. Algorithm can use API to display rules to user in progress (see fig. 3).

Extensibility

Developers willing to experiment with their own algorithms need to consider the `Simplifier` module only. We provide a sample one, which is recursively divided into submodules. One can find the exact place where to implement the functionality he wants, and replace only that part of the simplification chain. It is still optional, whether to use this new experimental module or the old, bundled one. A developer doesn't need to submit modified sources to our repository, just keeping his software (module) installed in the same NetBeans instance as jInfer will suffice.

Due to the fact that we implement [Aho96], a good support for merging state algorithms is already provided (for example automaton visualization). We divide automaton state merging in two tasks: merging strategy and merging criterion. The former one is responsible for deciding whether to merge two states or not. It has to decide where to stop merging and work is done, it uses merging criterion to test two states equivalency and can measure automaton as it wants to decide whether to continue or stop process. Merging criterion (we implement k, h-context) can be replaced and plugged into the merging strategy. One can implement for example s, k-strings ([GGR+00]).

We provide replacable parts in automaton-to-regexp conversion problem, so one can replace our state removal method with, for exmple, a simple weighted heuristic.

Possibilities are open.

Open issues

There are, however, still many issues to work on. First one is the set of supported input/output formats. On input, one can imagine support for XML Queries, on output export to languages like Relax NG or Schematron. There is

room for implementing schema visualization tool into jInfer, to enable user-aided schema refinement (to approach [BNV08]).

For now we ignore content models of attribute and text node values, but there is an interface waiting to be implemented. Thus, simple data type detection is waiting for its programmer.

Many algorithms published, wait to be implemented as jInfer modules, perhaps some of them will be implemented in the course of our master theses.

Conclusion

The framework presented in this paper aims to be the tool used to obtain schemas from XML/XSD/DTD files (and more after extension). While maintaining a distributable bundle for users, with the best modules and algorithms implemented so far, we predict it will be used in future studies of new algorithms as a test environment. The best of them will probably be merged into jInfer source tree to aid common users to solve their schema inference problems quickly and easily.

Weak side of our solution is the lack of good-quality automatic or semi-automatic algorithms to be used by common user, as well as the lack of content model inference. Our goal was to produce a stable framework for future development and we hope to negate these issues by implementing our master theses in this environment, providing it with a bundle of different algorithms, solving different input and output formats and infering more concise schemas.

References

[Aho96] H. Ahonen. *Generating grammars for structured documents using grammatical inference methods*. PhD thesis, Department of Computer Science, University of Helsinki, Series of Publications A, Report A-1996-4, 1996.

[BNST06] Geert Jan Bex, Frank Neven, Thomas Schwentick, and Karl Tuyls. Inference of concise dtds from xml data. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 115–126. VLDB Endowment, 2006.

[BNV07] Geert Jan Bex, Frank Neven, and Stijn Vansummeren. Inferring xml schema definitions from xml data. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 998–1009. VLDB Endowment, 2007.

[BNV08] Geert Jan Bex, Frank Neven, and Stijn Vansummeren. Schemascope: a system for inferring and cleaning xml schemas. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1259–1262, New York, NY, USA, 2008. ACM.

[GGR+00] Minos Garofalakis, Aristides Gionis, Rajeev Rastogi, S. Seshadri, and Kyuseok Shim. Xtract: a system for extracting document type descriptors from xml documents. *SIGMOD Rec.*, 29:165–176, May 2000.

[net] Netbeans platform. `http://netbeans.org/features/platform/`.

[VMP08] Ondřej Vošta, Irena Mlýnková, and Jaroslav Pokorný. Even an ant can create an xsd. In *DASFAA'08: Proceedings of the 13th international conference on Database systems for advanced applications*, pages 35–50, Berlin, Heidelberg, 2008. Springer-Verlag.

[Vyh] Julie Vyhnanovská. Automatic construction of an xml schema for a given set of xml documents.