Testing of Semantic Properties in XML Documents

Dominik Jungo, David Buchmann and Ulrich Ultes-Nitsche

telecommunications, *networks* & *security* Research Group
Department of Computer Science, University of Fribourg
Boulevard de Pérolles 90, 1700 Fribourg, Switzerland

Abstract. XML is a markup language with a clear hierarchical structure. Validating an XML document against a schema document is an important part in the work flow incorporating XML documents. Most approaches use grammar based schema languages. Grammar based schemas are well suited for the syntax definition of an XML document, but come to their limits when semantic properties are to be defined. This paper presents a rule based, first order schema language, complementary to grammar based schema languages, demonstrating its strength in defining semantic properties for an XML document.

1 Introduction

XML [9] [4] is an easily extensible markup language with a hierarchical structure, readable by computers as well as by humans. Own sub languages, can be defined with a so called schema. A schema defines the languages syntax and semantics. Syntax means the hierarchy of the XML elements. The term semantics depends on the expressiveness of the chosen schema language.

Own sub languages with syntax and semantics defined in a schema can be used as an exchange or serialisation format for data used in an application. In many cases, XML data gets from a producer to a consumer, which treats the received XML documents. It is important to define clearly the languages syntax and semantics, similar to the definition of public methods or functions in libraries, so that the producer and consumer of XML documents have the same understanding of the document. Different grammar based languages exist to create these definitions. DTD (Document Type defintion) [2] was a first attempt for a grammar based schema language. DTD is itself not written in XML, name spaces are not supported and certain syntactical aspects cannot be expressed. These points turned out to be a too big disadvantage. W3C designed XML schema [12] [14] as alternative to DTD and is now widely used to define the syntax of XML sub languages. W3C XML schema as well as Relax NG [15] are grammar based schema languages written in XML itself. The purpose of grammar based schema languages is mainly to define a document's syntax.

The paper is organised as follows. Section 2 shows the limits of grammar based schema languages justifying why it is necessary to introduce a new kind of schema languages. One of these new kind and more powerful languages, CLiXML, is introduced and explained more detailed in Section 3 using an example. Section 4 shows some alternatives to CLiXML and finally Section 5 concludes this paper.

Jungo D., Buchmann D. and Ultes-Nitsche U. (2006).
Testing of Semantic Properties in XML Documents.
In *Proceedings of the 4th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 38-46
DOI: 10.5220/0002484500380046

2 Limits of Grammar Based Schema Languages

Grammar based languages are well suited for a languages syntax definition, but reach quickly their limits when they are used to define semantic properties of an XML document. In the case of grammar based schema languages, semantics means specifying data types of attribute values. W3C XML schema knows simple and complex data types. Elements, Comments and Text belong to complex data type. Attribute values, comparable to simple data types, such as integers or strings, in programming languages, are simple data types. Complex data types can be extended using choices and sequences. Simple types can be adapted using restrictions, enumerations and regular expressions. This approach lets one define only a very limited set of semantics properties.

Let us illustrate the restricted schema capabilities with an example. Listing 1 shows an XML representation of an oriented labeled loop free graph respecting referential integrity. Referential integrity means that each edge is connected to an existing vertex. The graph consists of a list of vertices. Each vertex has a label (identifier, attribute *id*) and a list of edges from the vertex to other vertices, defined by the *to* attribute in the edge element. These syntactical properties can be easily defined by a grammar based schema.

```
<graph>
  <vertex id="0">
    <edge to="5"/>
    <edge to="6"/>
  </vertex>
  <vertex id="5"/>
  <vertex id="6"/>
  ...
</graph>
```

Listing 1. An XML representation of an oriented labeled loop free graph.

Consider the case of writing an application which expects as input an oriented loop free connex graph respecting referential integrity, in XML format, from an external source. We cannot trust this external source to deliver graphs as expected. Our application that treats these documents only works fine if the input document meets the given requirements. Hence we have to test the XML documents if they represent a graph that is oriented, loop free, connex and respecting referential integrity prior to the treatment within the application. This can happen using either the host language, the language in which the XML treating application is written, or using a schema document and a schema validator which checks whether the document meets all requisites specified in the schema document. Using a schema has the advantage that it can be negotiated between the document provider and the user before writing any applications. The schema document acts as an interface descriptor or contract between provider and consumer. Both parties, the provider and the consumer, can check the XML documents against the same schema.

Properties being well formed and having a valid semantics can be tested using grammar based schema. The documents correct structure, i.e. whether the labels are integer

and unique values, whether the graph element contains only vertex elements, the vertex elements contain only edges an so on, can be verified with an XML schema. Semantic properties, like testing whether the graph is connex, loop free and is respecting referential integrity cannot be tested using W3C XML schemas. W3C XML schemas are missing powerful constructs letting define required relationships between any components within the XML document. Roberts [10] describes it as missing *cross-field checks such as "if element x contains value y then element z should be mandatory"*. W3C schema grammars allows only to define which element x can appear as child of element z, but any other conditions than parent-child relationship cannot be expressed. Hence a new kind of not grammar based schema has to be introduced. The solution therefore are rule based schemas. Section 3 gives a short introduction to the most outstanding rule based schema language: Constraint Language in XML (CLiXML). Unlike grammar based schema languages, rule based schema languages have a much simpler structure. A rule based schema consists of a list of rules. Each rule has an output part which will be outputted by the schema validator when the rule succeeds, respectively fails and the rule itself, which contains constraints and restrictions of the XML document, and its content, i.e. required relationships between its content.

3 Testing of Semantical Properties in XML Documents

W3C XML schema cover basically syntactical properties. So only semantical properties have to be tested by the newly introduced language, since it does not assert it's claim to be an all-in-one language suitable for every purpose, doing work already done by W3C XML schema. All grammar based rules show that their template-like structure is cumbersome and hindering to describe semantical properties. They let only define a parent child relationship. In the language described here, CLiXML, another approach is used. Nodes (elements, attributes, text, comments) are selected by an XPath [3] expression and constraints are set up on, respectively between the selected nodes. XPath is the language defined by W3C to select nodes of an XML document. Selections are made using a regular expression inspired language, adopted for XML's tree structure. XPath requests result in lists. Hence using first order logic expressions seems to be quite a natural approach.

Quantifiers *forall* and *exists* contain a predicate variable (variable name and an XPath expression to select a list of nodes). The quantifier contains a further quantifier, a logical predicator (equal, less, bigger...) or a logical operator (and, or, not, ...). Predicators are binary. This means that predicator elements contain two attributes *op1* and *op2*, which are both related together by the predicator. Logical operators are elements that group operators, predicators or quantifiers. Nested quantifiers and predicators can use the preliminarily defined variable to specify the needed relationships between the selected nodes; i.e. inner quantifiers can define XPath expressions relative to the selected node or comparators can use the selected node and compare it with another one. The quantifiers act like iterators when being evaluated. In each iteration step, the predicate variable takes one value from the list. A forall test succeeds only if the nested test succeeds for all iteration steps. Exists succeeds if the nested test succeeds for at least one iteration step.

CLiXML was specified by Systemwire [1]. Systemwire integrated their implementation of CLiXML into a commercial product but published the languages XML schema on their website. An open source version [7] of CLiXML, corresponding to the XML schema, is being implemented by one of the authors of this paper. The open source implementation is also used in the VeriNeC project [13] [8] [6].

Let us illustrate the language's power using our example from Listing 1. Among others, we required that the graph fulfills referential integrity. I.e. each edge leads to an existing vertex. In other words, for all edge destination id's there must exist a vertex such that the edge destination id is the same as the vertex id. Or mathematically:
$\forall e \in \prime/graph/vertex/edge\prime, \exists v \in \prime/graph/vertex\prime : \$e/@to = \$v/@id$.
This can be translated into the CLiXML rule illustrated in Listing 2.

```
<rule id="valid_edges">
  <report>edge leads to non existing vertex</report>
  <forall var="edge" in="/graph/vertex/edge">
    <exists var="vertex" in="/graph/vertex">
      <equal op1="$vertex/@id" op2="$edge/@to" />
    </exists>
  </forall>
</rule>
```

Listing 2. An referential integrity CLiXML rule.

One could argue that the *id* and *idref* types from XML schema are able to describe exactly this scenario. But when using different elements with *id* attributes, XML schema can only test if the referenced ids exist, but not if they exist where they are expected. Listing 3 shows a graph with labeled edges, where the edge labels are also id attributes. In this case schema is not able to make a difference between ids of a vertex and ids of an edge. So it would be possible to define an edge connected to another edge if we would just rely on W3C schema.

```
<graph>
  <vertex id="0">
    <edge id="e1" to="5"/>
    <edge id="e2" to="6"/>
  </vertex>
  <vertex id="5"/>
  <vertex id="6"/>
  ...
</graph>
```

Listing 3. An XML representation of an oriented labeled loop free graph with labeled edges.

Testing whether the graph is loop free can be done using the label property in an oriented graph. For all vertices *srcV* and for all edges within the vertex *srcV*, named *dst*, it must hold that the id of the source Vertex *srcV* is strictly smaller than the "to" attribute of the destination vertex *dst*.

Or mathematically:
$\forall srcV \in \prime/graph/vertex\prime, \forall dst \in \prime\$srcV/edge\prime : \$srcV/@id < dst/@to$
Listing 4 shows the CliXML rule which tests whether the graph is loop free or not.

```
<rule id="loop_free">
  <report>graph is cyclic</report>
  <forall var="srcV" in="/graph/vertex">
    <forall var="dst" in="$srcV/edge">
      <less op1="$srcV/@id" op2="$dst/@to" />
    </forall>
  </forall>
</rule>
```

Listing 4. An CLiXML rule that checks for loop freeness.

Finally, checking whether the graph is connex can be done checking whether all vertices are reachable, except the vertex with the smallest id. In words, for all vertices, there exist an edge leading to it, or the vertice is the first.
$\forall v \in \prime/graph/vertex\prime, \exists e \in \prime/vertex/edge\prime : \$v/@id = min(/vertex/@id) \vee \$v/@id = \$e/@to$

```
<rule id="connex">
  <report>graph is not connex</report>
  <forall var="v" in="/graph/vertex">
    <exists var="e" in="/graph/vertex/edge">
      <or>
        <equal op1="$v/@id" op2="min(/graph/vertex/@id)"/>
        <equal op1="$v/@id" op2="$e/@to"/>
      </or>
    </exists>
  </forall>
</rule>
```

Listing 5. Listing: An CLiXML rule that checks whether the graph is connex.

Listing 5 shows the rule that checks whether the graph is connex.

3.1 Macros

CLiXML offers the possibility to write own macros. A macro is a template like structure. A macro is defined by a name, contains a list of parameters and a macro body. A macro can be called with a macro call element, containing a list of parameters. When a macro is called, the caller element is replaced by the definition body, with the variables replaced by it's parameters within the macro body. Macros can contain calls to other macros or to themselves. On a macro call, macro's are always evaluated by one level. Only when coming to the evaluation of a macro call in the body of the previously replaced macro, the macro is replaced by the next level. This level by level evaluation allows to write recursive macros.

3.2 User Defined Functions

Where CLiXML predicates, quantifiers and macros are not enough, one can introduce own validator functions. Therefore the validator has to be extended and these functions are only usable in the validator they were written for. Open CLiXML offers an interface which functions must implement. But these functions cannot be used on other implementations using another mechanism for user defined functions.

A function contains a name which is passed by an attribute and function parameters by a list of elements. Implementing own functions should be avoided, wherever possible, since own function implementations depend on validators implementation and cannot be passed as easily as the schema document. Using the example graphs, lets choose colored graphs and decide whether the number of used colors equals the graphs chromatic number. This would be a candidate to be expressed in a user defined function since macros are not adapted to do things that are that complex as determining the chromatic number.

4 Alternatives to CliXML

This section gives a short overview of alternatives for CLiXML allowing to define semantic properties.

4.1 Using a Host Language

One alternative to check the XML Document's semantical validity is surely doing this by own written functions in the host language of the application, using or building up the XML document. By doing so, one loses the independence of host systems. XML's strength is its independence from programming languages and systems, and it is not in XML's sense to hard code a semantics validator. Creator and consumer have to agree on the same validating function. When one side uses another host language, they have to write their own validator and it is not guaranteed that it behaves as the other side's validator. Another consequence using the host language instead of a schema document is, that one is losing the schema as contract document. One should be discouraged from hard coding the document's semantics in a host language language.

4.2 Schematron

As alternative to CLiXML, there exists Schematron [5]. Schematron is a rule based schema language also using XPath expressions, very similar to CLiXML. Reference implementation is written in XSLT. This is a very interesting concept, since it can be employed wherever an XSLT processor exists. Schematron's drawback is that it is not very strongly structured. Instead of splitting up rules into hierarchically nested elements, it uses a flat structure. The whole complexity of a rule lies within a single attribute containing an own language paired with XPath. Hence recursive definitions are not possible. It is proposed to use javascript or any other wrapper language [1] to eliminate this drawback. Schematron's drawbacks rise from a too simple and flat language structure, but allow an XSLT implementation.

[1] language which calls the validator

4.3 OASIS Content Assembly Mechanism

Content Assembly Mechanism (CAM) [11] was specified by the OASIS group. It aims to be a system to define, validate and compose XML documents. Unlike CLiXML, it is quite complex since it was designed for more tasks than CLiXML. CAM contains a part with rule based schemas. In CAM they are called business rules. The business rules are similar flat structured as rules in schematron with the same drawbacks. Recursive structures are missing. This makes CAM less powerful in expressing semantic rules than CLiXML. Constraint actions (see Listing 6) catches the readers eyes, when browsing the CAM tutorials.

```
<constraint action="functionName(xpath)"/>
```

Listing 6. A CAM constraint action.

The action attribute contains a java or c like function call with an XPath parameter. Such structured, nested content within one attribute is an indicator for improper language design. Having a closer look at the schema definition of CAM makes our skepticism stronger. The action attribute is declared as an unstructured string, but in the tutorials described as a c-like function call. Listing 7 shows a proper suggestion splitting up function name and XPath parameter into own attributes for improvement. The function like structure from Listing 6 has been apparently chosen, to keep the definition part simple and the language easily readable by humans. But this introduction of unstructured data makes it more difficult to be treated by an application. CAM offers a much longer list of useful predicates than CLiXML or Schematron do. CAM is used by a community, including people from industry.

```
<constraint action="functionName" parameter="xpath"/>
```

Listing 7. A proposal for a proper CAM constraint action.

5 Conclusion

CLiXML has been used by the autor of this paper to define acceptable event series in simulation log files [8]. By using CLiXML we experienced that CLiXML is working fine for strongly nested documents, but is rather clumsy for flat structure XML files.

Coming back to our graph example, using a structure as in Listing 8 complicates things a lot. We have to use recursive macros and things gets unnecessarily complicated, when a flat structure is chosen. Taking a well adapted XML structure reduces complexity for the schema as well as for the application treating the data.

```
<graph>
  <vertex id="0"/>
  <vertex id="1"/>
  <vertex id="2"/>
  ...
  <edge from="0" to="1"/>
  <edge from="0" to="2"/>
  <edge from="1" to="2"/>
  ...
</graph>
```

Listing 8. An alternative XML representation of a graph.

CLiXML specifications are not everywhere as precise as they should. This imprecise specification let the risk of not being compatible with the original CLiXML implementation. Details about how a macro is called are missing and the report output is completely open too. The macro calls are made from the Macro namespace. But a schema belonging to this namespace is missing. An own schema has been introduced there. The CLiXML schema itself was in some minor points not W3C schema compliant and needed adjustment. As a help to understand why a test failed, the open source implementation of CLiXML offers the possibility to add a stack trace to the report, showing which part of the rules failed and which succeeded. Quantifiers show in the stack trace which values a variable has taken during testing a document. Stack traces including variables has turned out to be useful.

This paper presented the relevance of schema documents in the work flow process where XML data is used. The limited capabilities for describing semantics using a schema document were explained using the graph example. Finally the rule based schema language, CLiXML was explained and compared to other rule based languages.

References

1. Clixml specification. Technical report, Systemwire.com, http://www.clixml.org/spec.html, 2003.
2. Jon Bosak, Tim Bray, Dan Connolly, Eve Maler, Gavin Nicol, C. Michael Sperberg-McQueen, Lauren Wood, and James Clark. Guide to the w3c xml specification (xmlspec) dtd, version 2.1. Technical report, W3C, http://www.w3.org/XML/1998/06/xmlspec-report.
3. James Clark and Steve DeRose. Xml path language (xpath). Technical report, W3C, http://www.w3.org/TR/xpath, 1999.
4. Elliotte Rusty Harold and W. Scott Means. *XML in a Nutshell*. O'Reilly, Sebastopol, 3rd edition, 2004.
5. Rick Jelliffe. An xml structure validation language using patterns in trees. Technical report, Academia Sinica Computing Centre, http://xml.ascc.net/resource/schematron, 2005.
6. Dominik Jungo. The role of simulation in a network configuration engineering approach. In *ICICT 2004, Multimedia Services and Underlying Network Infrastructure*, Cairo, Egypt, 2004. Information Technology Institute.
7. Dominik Jungo. Open clixml. Technical report, University of Fribourg, http://clixml.sourceforge.net/, 2005.
8. Dominik Jungo, David Buchmann, and Ulrich Ultes-Nitsche. A unit testing framework for network configurations. In *Proceedings of the 3rd International Workshop on Modelling,*

Simulation, Verification, and Validation of Enterprise Information Systems (MSVVEIS 2005), Miami, Florida, USA, 2005. INSTICC Press.
9. Liam Quin et al. Extensible markup language (xml). Technical report, W3C, http://www.w3.org/XML/, 1996-2003.
10. Martin Roberts. jcam. Technical report, CAM, http://jcam.org.uk/, 2006.
11. OASIS CAM TC. Oasis content assembly mechanism (cam) tc. Technical report, OASIS, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=cam, 2004.
12. Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. Xml schema. Technical report, W3C, http://www.w3.org/TR/xmlschema-1/, 2004.
13. Ulrich Ultes-Nitsche, Dominik Jungo, and David Buchmann. Verified network configuration. Technical report, University of Fribourg, http://diuf.unifr.ch/tns/projects/verinec/, 2004–2005.
14. Eric van der Vlist. *XML Schema*. O'Reilly Media, Inc., 2002.
15. Eric van der Vlist. *RELAX NG*. O'Reilly Media, Inc., 2003.