UDC 004.062

O.I. Marchenko, D.V. Pazii

TECHNIQUE FOR ACCELERATION OF XML DOCUMENTS VALIDATING AGAINST XSD SCHEMA

Abstract. This work deals with researching of techniques of validation XML-files against XSD-schemes and proposes a new technique based on performing ordinary and partial validations. Suggested technique of XML validation lies in the fact that file is valid. The next step is to perform partial validation on every change of XML-file instead of full.

Keywords: XML, XSD, validation

Introductory

As for the last decade, the complexity of software has grown in many times in comparison with the beginning of 2000. If twenty years ago web sites were kind of static pages with the text decorated with bunch of tags, nowadays every web site is a web application with complicated structure, lots of multimedia content and non-trivial design. Most of businesses solutions are based on using of modern web technologies, as web solutions are platform independent. This means much less money and time consuming process of development for wide variety of target platforms. After appearance of smartphones and tablets amount of target platforms and even screen resolutions became even greater and thus making many pros of web applications.

As known, XML is an extensible markup language, which is basically a plain text document which syntax has to correspond some rules [1]. Since the invention of XML, developers found a huge amount of different use cases, like:

- data storage;
- usage within web translations and services;
- platform independent application settings;
- large amount of XML extensions such as electronic books (fb2, epub, XHTML), RSS news, etc;
- UI declarations.

Web services are almost mandatory part of every web application for server-client solutions. Functioning of web service is based on exchanging of XML messages, mostly by SOAP protocol. Confidence that received XML message conforms some specific rules makes developer able to perform some processing and transformations with assurance as shown on fig. 1. Thus, one of the main tasks appears.

The point is finding out whether XML document conforms specified rules or not, and if it is not then why.

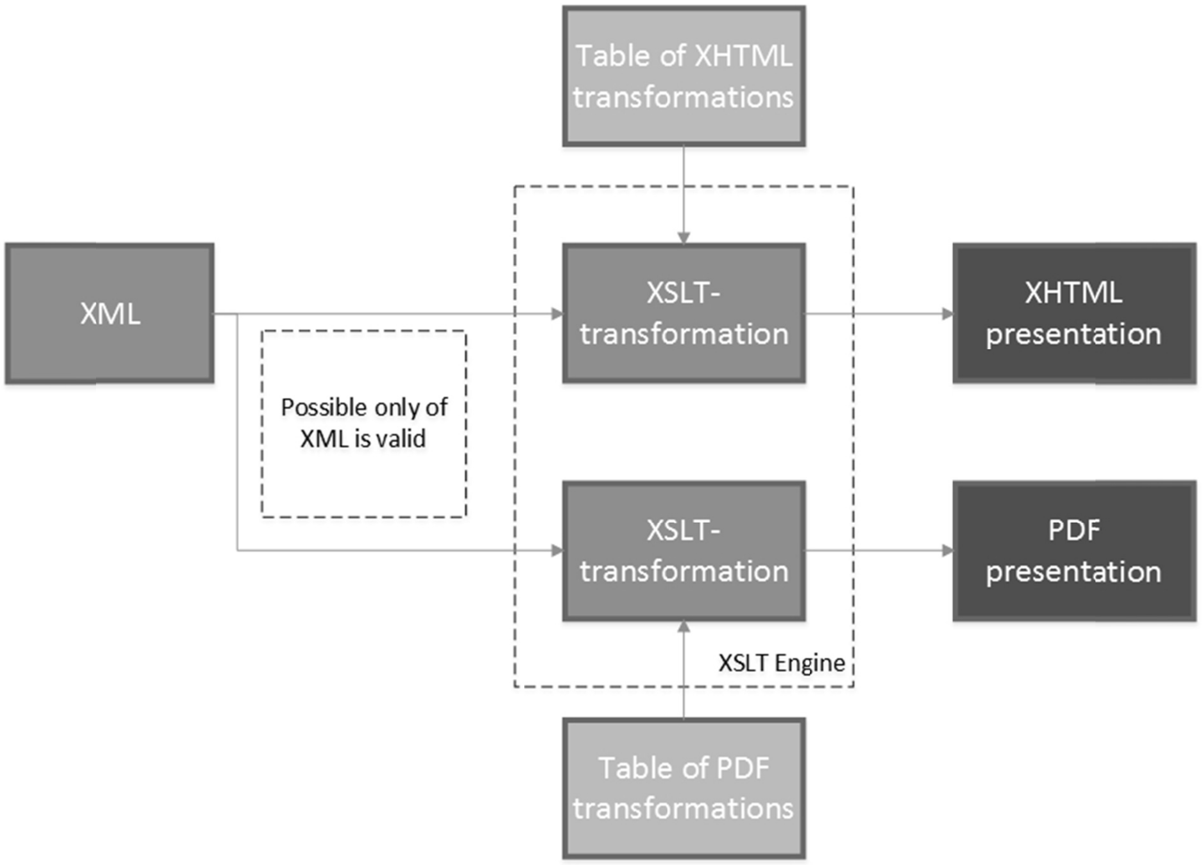


Table of XHTML transformations
XML
XSLT-transformation
XHTML presentation
Possible only of XML is valid
XSLT-transformation
PDF presentation
XSLT Engine
Table of PDF transformations

Fig 1 – XML transformations

Aims

Most of documents is not just plain text, but has some semantics. XML solves syntax presentation problem, whereas schema partially solves the problem of semantic meaning.

Software systems that perform some operations upon XML require performing lots of documents validation, which means that this process must be optimized and fast.

Talking about existing techniques, the vast majority of them are aimed to solve the problem of full validation, such as incremental validating or even splitting the document into chunks and checking simultaneously several chunks in different threads. Validating small files is a simple task for any of numerous algorithms. However, when the file is big (more than 5 Mb), it may be a problem. Moreover, there are situations when it is required to make several sequential actions on XML and not to break document correctness [2, 3, 4]. It can be some automatic changes (adding information into document by web agents) or making manual changes in some visual editors. This may dramatically increase duration of XML validation.

Thus, the main purpose of this work is considered to be a research of techniques and algorithms of XML validation against XSD schemas [5]. In addition, a new validation technique is proposed. It is based on this research and has lesser algorithm complexity in case of sequential validation of the same XML after some changes.

Research description

Basically, before the process of XML validation against XSD schema, it is required to make two steps: firstly, perform syntax analysis of a document and secondly, perform syntax analysis of its schema.

There are two main techniques of XML processing:

- DOM (Document Object Model);
- SAX (Simple API for XML).

The first technique is parsing and building of full element tree of the document. The majority of XML applications works with documents using DOM. Data types of DOM nodes are abstract; every implementation has its own programming language dependent data types. The main disadvantage of DOM is extensive use of memory, because before any operation upon DOM requires XML to be fully loaded, processed and transformed to the object tree (fig 2).

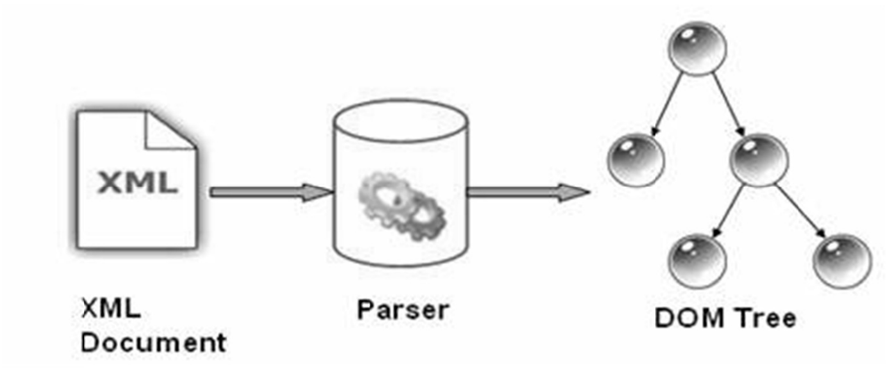
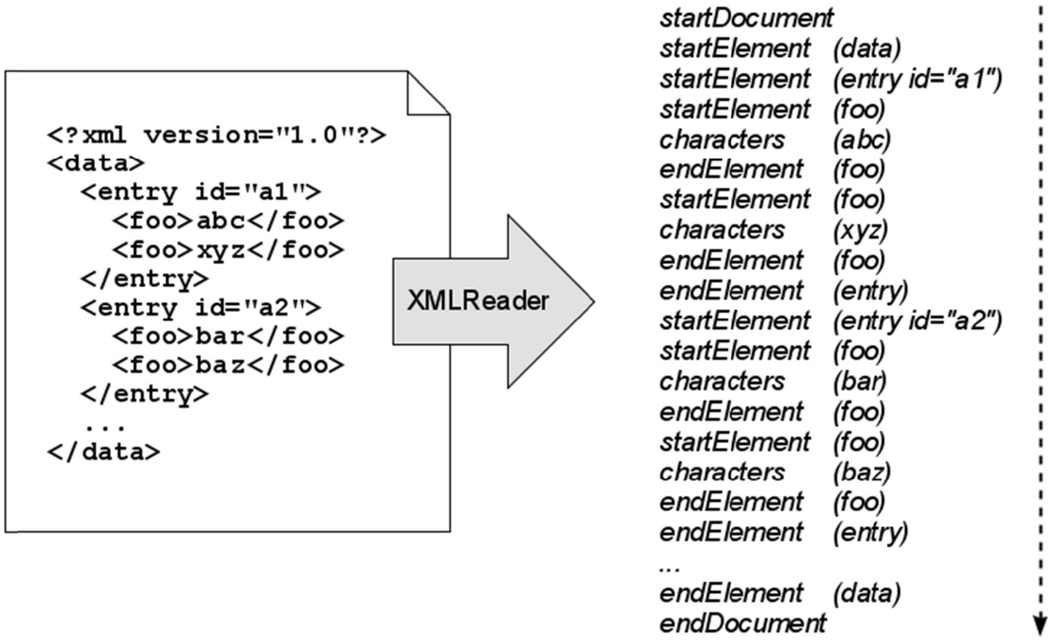


Fig 2 – Parsing XML into DOM

DOM model is very useful for data manipulating due to loading the whole document into memory. However, this can cause excessive use of memory.

SAX solves the problem of memory overuse by scanning the document from the beginning till the end and notifying the application about such events (fig 3) as “start of element” and “end of the element”. This approach allows reducing the use of memory considerably. However, this also means that manipulating the document will be impossible due to absence of elements tree as with DOM.



```
<?xml version="1.0"?>
<data>
  <entry id="a1">
    <foo>abc</foo>
    <foo>xyz</foo>
  </entry>
  <entry id="a2">
    <foo>bar</foo>
    <foo>baz</foo>
  </entry>
  ...
</data>
```

```
startDocument
startElement  (data)
startElement  (entry id="a1")
startElement  (foo)
characters    (abc)
endElement    (foo)
startElement  (foo)
characters    (xyz)
endElement    (foo)
endElement    (entry)
startElement  (entry id="a2")
startElement  (foo)
characters    (bar)
endElement    (foo)
startElement  (foo)
characters    (baz)
endElement    (foo)
endElement    (entry)
...
endElement    (data)
endDocument
```

Fig 3 – SAX XML processing

Finally, let us look through the main opportunities of XML schema:

- strong controlling of data typing document nodes and attributes;
- defining sequence of nodes appearing, observe for the presence of mandatory nodes and attributes;
- demanding the element uniqueness in specified context;
- creating optional nodes that require presence of one or another node, depending on the context;
- fulfilling the requirements of specific predicate on the group of nodes and attributes.

Schema features

XSD describes a document schema by declaring set of definitions (parameters, elements and attributes), which describe its class in terms of syntax restrictions for this document. XSD is developed from DTD (Document Type Definition – previous W3C recommendation for XML schema). Thus, it has common features such as set of regular expressions that can be performed upon atomic terms or elements. However, XML schema also extends functionality of DTD by:

- patterns (any, anyType, anyAttribute) that make possible usage of any element from specified namespace;
- substitution group that defines the group of types that can be used instead of specific one;
- amount of occurrences and opportunity of specifying of minimal and maximal amounts for any element.

Speaking about algorithmic complexity, it should be mentioned that validation upon XML schema is heavier than validation upon DTD. However, the latest XML specification simplifies this process significantly.

Building a validator

As it was said at the beginning of the article, the main use case for this validator is when it is already known that XML is valid and required to be validated again after some small changes.

For the beginning, let us specify the list of simple structure changes that will be checked for correctness:

- Add: creating of new sub-element with the type X on the position N;
- Remove: removing of sub-element from the position N;
- Move: element carrying from the position N to the position M (even if this action reduces to element remove and add, but intermediate state may be inconsistent).

The basis of validating algorithm is in converting XSD model to finite state machine (FSM) which consists of two parts: one is for particles and another is for terms.

To translate a particle to an FSM ending at the state S [6]:

1. Set the start state n to S;
2. If the particles MaxOccurs is infinity:
   2.1. Add a new intermediate state t; which is got as a result of translating term to an FSM;
   2.2. Add lambda (also known as epsilon, or empty) edges from t to n and from n to S (fig 4).
3. If MaxOccurs is numeric:

Build a chain of (MaxOccurs-MinOccurs) copies of the term translation of backwards from S, with lambda transitions from intermediate state on each step to S;

For example, for min=2 and max=4 (fig 5)

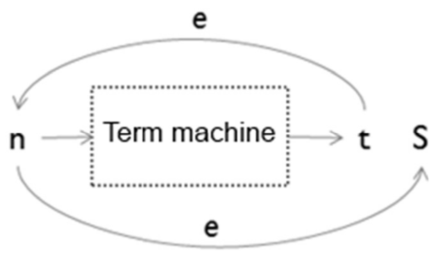


Fig 4 – Particle FSM first step

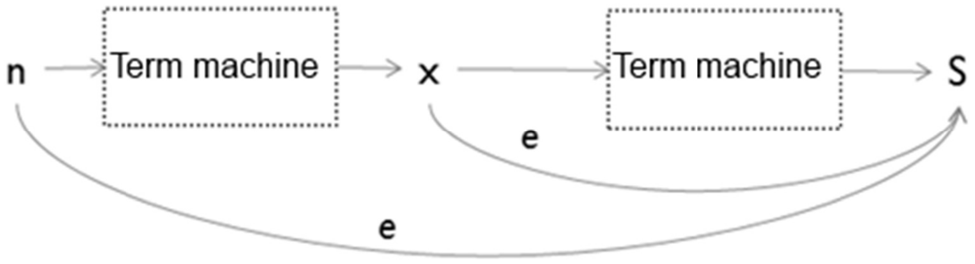


Fig 5 – Particle FSM for min/max values

4. Build a chain of MinOccurs copies of the translation of term from start state n, and as the result of previous steps.

As for the second step, building of FSM with specified final state S for concrete term:

1. If the term is a pattern (any):
   1.1. Create new state b and connect it with edge S, which is labeled with the term type.
   1.2. Return b;
2. If the term is element definition:
   2.1. Create new state b and for each element of the substitution group create edge from b to S, labeled with the type of the element.
   2.2. Return b;
3. If the term is a choice:
   3.1. Create new state b, for every choice element create machine (first step) and connect it with lambda edges with state b and S.
   3.2. Return b;
4. If the term is a sequence:
   4.1. For every sequence element create machine (first step) and connect them in reverse order, starting from the state S.

4.2. Return the first state in received chain.

After applying of proposed algorithm to every type in schema, accordance of type and the validating FSM is received in outcome. In addition, the last task would be selecting of proper FSM for validating of changed node of the document. This could be achieved by use of PSVI (Post Schema Validation Infoset) which is generated by almost every full validator. For every tree node, it points to according schema type. Finally, if there was an element X and it was added a new sub-element B, validation would be performed by FSM of X element (fig 6).

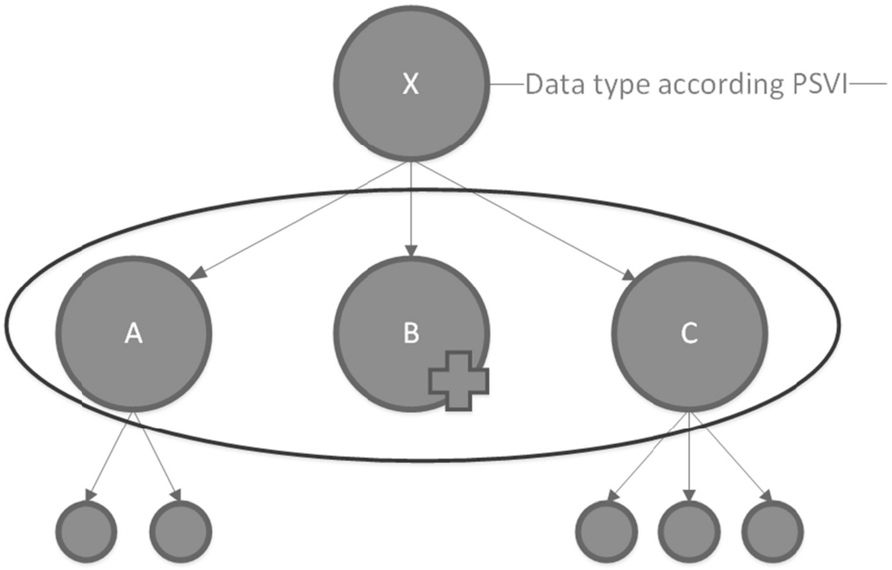


Fig 6 – Validation of X element after B was added

Conclusions

The results of performed tests are below:

Table 1

Validation time tests

Amount of element structure	Nesting level	Amount of types	Average validation time by Xerses	Implementation of algorithm average time
32	4	16	10 ms	<1 ms
32	4	40	17 ms	<1 ms
12000	4	16	47 ms	<2 ms
12000	4	40	56 ms	<2 ms
12000	32	16	2460 ms	<5 ms
12000	32	40	2650 ms	<6 ms

As it can be seen from the table, proposed validation technique shows great performance increase, thus making this algorithm suitable for use in visual editors and for validation of batch operations upon XML documents.

REFERENSES

1. Bray T., Paoli J., Sperberg-McQueen C. M., Maler E. Extensible Markup Language (XML) 1.0 (Second Edition). // W3C. - Boston, 2000.
2. Balmin A., Papakonstantinou Y. Incremental Validation of XML Documents. // University of California, San Diego. - 2002.
3. Wu Y., Qi Z., Zhiqiang Y. A Hybrid Parallel Processing for XML Parsing and Schema Validation. // The Markup Conference. - 2008.
4. Thompson, Henry S., Mendelsohn N. XML Schema Validator. // W3C and University of Edinburgh. - 2003.
5. Thompson, Henry S., Mendelsohn N., Maloney M., Beech D. XML Schema Part 1: Structures. // W3C. - Boston, 2001.
6. Aho A., Ullman J. Principles of Compiler Design // Addison-Wesley. - 1977. Reading, MA.

ISSN 1562-9945