Validating a Document using XML Languages

Mr. Akhilesh Mohan Srivastava*, Ms Sanju Tiwari **

* Associate Professor in Saraswati Institute of Technology and Management
** Sr.Lecturer in in Saraswati Institute of Technology and Management

Abstract- Validation of a document plays a very important role and also solves a number of important issues when working with XML and XML related technologies. Validation allow to work with modules and performing multiple validations in one actions. During the last few years a number of different XML schema languages have appeared as suggested replacements for the ageing Document Type Definition (DTD). Schematron [Schematron] is an XML schema language designed and implemented by Rick Jelliffe at the Academia Sinica Computing Centre, Taiwan. It combines powerful validation capabilities with a simple syntax and implementation framework. In this paper authors explain how a document can be validated through XML Schema and how a rule based validation can be implemented with Schematron. But when several validations is in computing process, performance of Schematron is degraded. To cope this degradation **RELAX**-NG is useful for validating a document.

Index Terms- XMLDTD, XML Schema, Schematron, RELAX-NG.

I. Introduction to XML Validation

XML files are developed for people to be easily read and edit. They are also useful for data exchange among different systems and different applications.

Validation confirms that XML data follows a specific predefined format so that an application can receive it in a predictable way. This format can be provide in a number of different ways including DTD and XML Schemas. A document that has been checked against a DTD or schema in this way is considered a "valid" document.XML validation also confirms that a XML document is well-formed and valid for specific format. A well-formed document can forms to the rules of XML.All elements have start and end tags, all attributes are enclosed in quotes ,all elements are nested correctly. If a document is not well-formed It can't be parsed.

A valid XML documents are always well-formed, but well-formed documents may not be valid. Validity plays important role to specifies that the document is syntactically and structurally correct.Validity must be ensured whenever the document is updated.

XMl not allowed these type of elements…

```
<vehicle>
   <twowheeler>
     ---------
     ---------
   </vehicle>
</twowheeler>
```

Each tag should be closed in proper manner. Nesting rules for tag must be obeyed.

A document can be validated in ORACLE by using schemavalidate() procedure and isschemavalidate() function as below--

```
declare
    xmldoc xmltype;
 begin
  --  validate against XML schema
 xmldoc.schemavalidate();
   If xmldoc.isschemavalidated () = 1 then
        dbms_output.put_line ('Data is valid');
    else
        dbms_output.put_line('Data is invalid');
    end if;
end;
```

isschemavalidate() is a Boolean function that returns 1 or zero for valid or invalid.

This introduction to XML presents the Extensible Markup Language at a reasonably technical level for anyone interested in learning more about structured documents.

XML documents contain elements and attributes. They provide a flexible and powerful way to exchange data between applications and organizations. To specify the allowable structure and content of an XML document, you can write a DTD, an XDR schema, or an XSD schema.

In the creation of a database, using a data model in conjunction with integrity constraints can ensure that the structure and content of the data meet the requirements. But it is very difficult to ensure, using XML, when your data is just text in hand-editable files. Fortunately,validating files and documents can make sure that data fits constraints.This is possible by using Document Type Definition (DTD) or an XML Schema document and also some other languages like Schematron and **RELAX**-NG.

1.1 Features of DTDs:-

XML validation is a crucial part of predictable and efficient processing of XML instances. Knowing the structure of an XML document saves the developer from writing unnecessary conditional application logic. Once a document is identified as belonging to a class of documents, many assumptions about its structure can be made.

A Document Type Definition (DTD) is an optional part of an XML document that defines the document's exact layout and structure. There are two main reasons for an XML author to write a DTD for their document. The first of these is documentation. A developer can look at a document with a DTD and immediately

understand the structure of the data. A DTD can formally state that, for example, a product number contains a manufacturer's code, followed by a batch number, followed by a part number, followed by an optional colour code. The second reason to use a DTD is to enable what is known as validation. The process of document validation involves passing an XML document through a processor which reads the DTD, then examines the XML data to ensure that elements appear in the right order, that mandatory elements and attributes are in place, that no other elements or attributes have been inserted where they shouldn't have been, and so on. Working with validated data makes life much easier for a developer. If data is known to be valid, it is completely predictable. For the product number example given above, a developer can write code to read each of the first 3 pieces of data from a validated document, then do a check to see if the optional color code has been provided before attempting to read that. An XML document can be viewed abstractly as a tree of nested elements. The basic mechanism for specifying the type of XML documents is provided by Document Type Definitions (DTDs) [W3C 1998]. DTDs can be abstracted as extended context-free grammars (CFGs).for example

```
<!DOCTYPE car [
<!ELEMENT car (make, model, year)>
...
...
]>
```

This line says that the car element must contain – and can only contain – a make
element, followed by a model element, followed by a year element, like this:

```
<car>
<make>
...
</make>
<model>
...
</model>
<year>
...
</year>
</car>
```

The make, model and year elements must all appear (i.e. none are optional) and they must appear in that order. No other elements are allowed. To make an element optional, follow it with a question mark in the DTD:

```
<! DOCTYPE car [
<! ELEMENT car (make, model, year?)>
...
]>
```

The next two elements of our DTD might look like this:

```
<!DOCTYPE car[
<!ELEMENT car (make, model, year?)>
<!ELEMENT make (#PCDATA)
<!ELEMENT model (#PCDATA)
...
]>
```

It doesn't matter which order you declare your elements in inside a DTD. The make and model elements have been declared as containing #PCDATA which means parsed character data – in other words, free text. The name parsed character data means just that: the characters will be parsed by the XML parser.

DTDs are the first XML validation mechanism[14] and for all practical purposes and they use a non-XML syntax.DTDs are not written in XML syntax and rely post-processing for validation.DTDs are sufficient for simple XML schemas.

1.2 Limitations [2] of DTDs:-

In spite of their several advantages, DTDs suffer from a number of limitations-

- DTDs has Non-XML syntax. They have angled bracket syntax like(<!element-- >) this is quite different from the XML syntax.DTD does not have the standard <?xml version---- ?> tag etc.
- Each XML document can have only one DTD.Multiple DTDs cannot be used to validate one XML documents.
- DTDs does not support for namespaces.
- DTD have basic data types. In real-life application, it is not applicable and not sufficient also for specific data types.
- There is no inheritance in DTD.But DTD is not object-oriented. They do not allow the designer to create data type and extended them as desired.
- Its very hard to read and maintain large DTDs.
- There are no default values for elements and attribute defaults must be specified when they are declared.
- DTDs have limited set of rules and constraints.

2.1 Features of XMLSchema:-

As we know that a DTD is used for validating the contents of an XML document. A DTD is an important feature of the XML technology. But there are a no of areas in which DTDs are not fit. The main problem is that their syntax is not like that of XML document. Therefore W3C introduces a schema that is an alternate to DTD.

An XML Schema[4] is a document which describes another XML document. XML Schemas are used to validate XML documents. An XML schema itself is an XML document which contains the rules to be validated against a given XML instance document.

There are two forms of word 'Schema',in capital form and in lowercase form.The lowercase form is a generic term and may refer to any type of schema, including DTD, XML Schema (aka XSD), RELAX NG, or others, and should always be written using lowercase except when appearing at the start of a sentence. The form "Schema" (capitalized) in common use in the XML community always refers to W3C XML Schema[15].

DTDs were inherited from XML's origins as SGML[5] (Standard Generalized Markup Language) and, as such, are limited in their expressiveness. DTDs are for expressing a text document's structure, so all entities are assumed to be text. Schemas provide the ability to define an element's type (string, integer, etc.) and much finer constraints (a positive integer, a string starting with an uppercase letter, etc.). DTDs enforce a strict ordering of elements; Schemas have a more flexible range

of options (elements can be optional as a group, in any order, in strict sequence, etc.). Finally schemas are written in XML, whereas DTDs have their own syntax.

II. When need an XML Schema?

When sending data from a sender to a receiver, it is essential that both parts have the same "expectations" about the content. There should be a format to understand for both receiver and sender.With XML Schemas, the sender can describe the data in a way that the receiver will understand.One more need for XML Schema that it can be reused in other schemas.

Assume a function/method for an application that manages employee data. The function is expecting the employee information in the following XML structure:

```
<Student>
  <Name>
    <First>Jacob</First>
    <Middle>V</Middle>
    <Last>Sebastian</Last>
  </Name>
  <!-- Deleted other information for brevity -->
</Student>
```

The function needs to make sure that the caller passes correct XML data. An XML Schema which describes and validates the above XML document is given below.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="First" type=”xs:string”/>
              <xs:element name="Middle" type=”xs:string”/>
              <xs:element name="Last" type=”xs:string”/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Example. Consider this simple instance document:

```
<?xml version="1.0"?>
<Doc xmlns="http://www.demo.org"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://www.demo.org doc.xsd">
    <X>10</X>
    <Y>20</Y>
</Doc>
```

With XML Schemas we can check the following constraints:

- the Doc (root) element contains a sequence of elements, X followed by Y
- the X element contains an integer
- the Y element contains an integer

In fact, here's an XML Schema which expresses these constraints:

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://www.demo.org"
          xmlns="http://www.demo.org"
          elementFormDefault="qualified">
    <xsd:element name="Doc">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="X" type="xsd:integer"/>
          <xsd:element name="Y" type="xsd:integer"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

XML Schemas does **not** give us the capability to express the following constraint:

- the value of X must be lower than the value of Y

So should be there to check this constraint?
There are many other schema languages besides XML Schemas:

- Schematron
- TREX
- **RELAX**
- SOX
- XDR
- HOOK
- DSD
- Assertion Grammars
- Xlinkit

Thus, the first option is to use one (or more) of these schema languages to express the additional constraints.

2.2 Limitation of XMLSchema:

1. W3C XML Schema is complex and hard to learn, although that's partially because it tries to do more than mere validation.
2. There is a problem with XML Schema that it is not supported by all parsers.
3. XML Schema also not supported to entities.
4. W3C XML Schema does not implement most of the DTD ability to provide data elements to a document. While technically a comparative deficiency, it also does not have the problems that this ability can create as well, which makes it strength.

3.1 Features of Schematron:-

Schematron is a rule-based validation language.It is used for making assertions about the presence or absence of patterns in XML trees. Schematron is a structural schema language expressed in XML using a small number of elements and XPath.It differs in basic concept from other schema languages in that it *not based on grammars* but on finding *tree patterns* in the parsed document. XML Schema and DTD is not capable of expressing constraints but Schematron can express constraints in a better way. Schematron can also specify required relationships between multiple XML files.

Schematron is useful for conditional expression. And it is free and open source implementations available.The Schematron is trivially simple to implement on top of XSLT and to customize. A Schematron schema and an XML instance document are sent into a Schematron validator, which validates the instance document against the Schematron schema in figure 1..

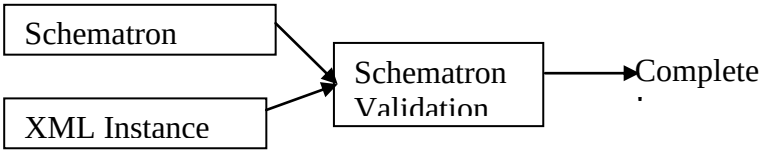

Figure 1

The Schematron can be used to develop and mix two kinds of schemas:

- ***Report*** elements allow you to diagnose which variant of a language you are dealing with.
- ***Assert*** elements allow you to confirm that the document conforms to a particular schema.

It is based on a simple action:

- First, *find* a context nodes in the document (typically an element) based on XPath path criteria;
- Then, *check* to see if some other XPath expressions are true, for each of those nodes.

The Schematron can be useful *in conjunction* with many grammar-based structure-validation languages: DTDs, **XML Schemas**, **RELAX**, **TREX**, etc. Indeed, Schematron is part of an ISO standard (DSDL: Document Schema Description Languages) designed to allow multiple, well-focussed XML validation languages to work together.

For example, using Schematron you can embed the additional constraints within the XSD document (within <appinfo> elements). The XSD document shown earlier has been enhanced (below) with Schematron directives:

```
<?xml version="1.0"?>
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.demo.org"
      xmlns="http://www.demo.org"
      xmlns:sch="http://www.ascc.net/xml/schematron"
      elementFormDefault="qualified">
   <xsd:annotation>
      <xsd:appinfo>
         <sch:title>Schematron validation</sch:title>
         <sch:ns prefix="d" uri="http://www.demo.org"/>
      </xsd:appinfo>
   </xsd:annotation>
   <xsd:element name="Doc">
      <xsd:annotation>
         <xsd:appinfo>
            <sch:pattern name="Check X greater than Y">
               <sch:rule context="d:Doc">
                  <sch:assert test="d:X >d:Y"
                        diagnostics="lessThan">
                     X should be greater than Y.
                  </sch:assert>
               </sch:rule>
            </sch:pattern>
            <sch:diagnostics>
               <sch:diagnostic id="lessThan">
                  Error! X is less than Y
                  X = <sch:value-of select="d:X"/>
                  Y = <sch:value-of select="d:Y"/>
               </sch:diagnostic>
            </sch:diagnostics>
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="X" type="xsd:integer"/>
            <xsd:element name="Y" type="xsd:integer"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Schematron will extract the directives out of the XSD document to create a Schematron schema. Schematron will then validate the instance document against the Schematron schema. Schematron is used for the following checking[11]—

1. **Co-constraint checking**
   Co-constraints are constraints that exist between data (element-to-element co-constraints, element-to-attribute, attribute-attribute). Co-constraints may be "within" an XML document, or "across" XML documents (intra- and inter-document co-constraints).Schematron is very well-suited to expressing co-constraints.

2. **Cardinality checking**

Cardinality constraints are constraints on the occurrence of data. The cardinality constraints may apply over the entire document, or to just portions of the document.Schematron is very well-suited to expressing cardinality checks.Cardinality checking encompasses uniqueness checking.Existence checking is a special case of cardinality checking.The following example will be used to characterize the next category of Schematron usage:

```
<?xml version="1.0"?>
```

```
<ExamResults>
   <ByPercentage>
      <Candidate name="John">61</Candidate>
      <Candidate name="Sara">54</Candidate>
      <Candidate name="Bill">55</Candidate>
   </ByPercentage>
</ElectionResults>
```

3. Algorithmic checking

In the above example the algorithmic constraint is: "the exam results must add up to 100%" (i.e., 61 + 54 + 55 = 100). In general, validity of data in an XML instance document is determined not by student's examination or comparison of the data, but requires performing an algorithm on the data. Schematron is very well-suited to expressing algorithmic checks.

3.2 Limitations of Schematron [12]:-

1. There may be running many validation at a time in Schematron so performance can be degrade and also may be more expensive.

2. It may be difficult to swapping out the particular grammar language that is currently being used and replacing it with a different grammar language.

3. Constraint checking is a big-bang event. All constraints grammar, co-constraints, cardinality, algorithmic are checked at once.

5. Schematron was difficult to place on the graph. Schematron can express content dependent rules across elements and attributes that none of the other methods can. On the other hand, Schematron is not as well suited to basic content model restrictions as the others are.

6. Schematron is best used as supplemental power to the other validation methods.

4.1 Features of RELAX-NG:--

RELAX NG (**REgular LAnguage for XML Next Generation**) is a schema[7] language for XML - a **RELAX**-NG schema define a pattern for the structure and content of an XML document. A **RELAX**-NG schema is itself an XML document but **RELAX**-NG also offers a popular compact, non-XML syntax. Compared to other XML schema languages **RELAX**-NG is considered relatively simple.

It was defined by a committee specification of the OASIS **RELAX** NG technical committee in 2001 and 2002, based on Murata Makoto's **RELAX** and James Clark's TREX and also by part two of the international standard ISO/IEC 19757: Document Schema Definition Languages (DSDL).

RELAX NG is a schema language for XML. The key features of **RELAX** NG are that it:

- is simple.
- is easy to learn.
- has both an XML syntax and a compact non-XML syntax.
- does not change the information set of an XML document.
- supports XML namespaces.
- treats attributes uniformly with elements so far as possible.
- has unrestricted support for unordered content.
- has unrestricted support for mixed content.
- has a solid theoretical basis.
- can partner with a separate data typing language.

Consider a simple XML representation of an email Book Store:

```
<bookstore>
  <name>Mike's Store</name>
  <topic>
    <name>XML</name>
    <book isbn="123-456-789">
      <title>Mike's Guide To DTD's and XML Schemas<</title>
      <author>Mike Jervis</author>
    </book>
  </topic>
</bookstore>
```

The DTD would be as follows:

```
<!DOCTYPE bookstore [
  <!ELEMENT bookstore (topic+)>
  <!ELEMENT topic (name,book*)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT book (title,author)>
  <!ELEMENT title (#CDATA)>
  <!ELEMENT author (#CDATA)>
  <!ELEMENT isbn (#PCDATA)>
  <!ATTLIST book isbn CDATA "0">
  ]>
```

A **RELAX** NG pattern for this could be written as follows:

```
<element name="bookstore"
xmlns="http://RELAXng.org/ns/structure/1.0">
  <ZeroOrMore>
    <element name="topic">
      <element name="name">
        <text/>
      </element>
      <element name="book">
        <element name="title">
        <element name="author">
        <text/>
      </element>
    </element>
  </zeroOrMore>
</element>
```

If the bookstore is required to be non-empty, then we can use oneOrMore instead of zeroOrMore:

```
<element name="bookstore"
xmlns="http://RELAXng.org/ns/structure/1.0">
  <oneOrMore>
    <element name="topic">
```

```
    <element name="name">
      <text/>
    </element>
        <element name="book">
          <element name="title">
          <element name="author">
      <text/>
    </element>
  </element>
 </oneOrMore>
</element>
```

4.2 Limitation of RELAX -NG:

1. **RELAX** - NG[13] really hits a sweet spot for both ease of use and expressive power.
2. **RELAX** NG has no ability to apply default attribute data to an element's list of attributes (i.e., changing the XML info set), while W3C XML Schema does. Again, this design is intentional and is to separate validation and augmentation.

III. Conclusion

XML is a promising standard that will undoubtedly impact positively on the Web community. Although the expressiveness of its data model is limited and it poses scalability issues, its flexibility, its simplicity and its interconnection capabilities make it an excellent candidate as a language for data exchange.

One can create typed XML documents using XSD but still get rich validation of business rules in a declarative manner from the schema language, as well.

Acknowledgement

Thanks to all my friends and collaborators in XML and XML Languages.

References

[1] ^ "Well-Formed XML Documents". *Extensible Markup Language (XML) 1.1*. W3C. 2004.
[2] http://sharat.wordpress.com/2007/05/18/what-are-the-disadvantages-of-dtd/.
[3] http://mikie.iki.fi/teaching/xml_s03/slides-lect4.pdf.
[4] https://www.simple-talk.com/sql/learn-sql-server/introduction-to-xml-schema/.
[5] http://www.javaworld.com/javaworld/jw-08-2005/jw-0808-xml.html.
[6] http://www.xfront.com/ExtendingSchemas.html
[7] http://en.wikipedia.org/wiki/**RELAX**_NG.
[8] http://**RELAX**ng.org/tutorial-20011203.html.
[9] http://www.ldodds.com/papers/schematron_xsltuk.html#c35e2592b5
[10] http://en.wikipedia.org/wiki/Schematron.
[11] http://osdir.com/ml/text.xml.devel/2007-7/msg00082.html.
[12] http://www.dpawson.co.uk/schematron/introduction.html.
[13] http://en.wikipedia.org/wiki/XML_schema_languages.
[14] https://www.simple-talk.com/sql/learn-sql- server/introduction-to-xml-schema/
[15] XML Schema Language Comparison http://en.wikipedia.org/wiki/XML_schema

Authors

First Author – Mr. Akhilesh Mohan Srivastava, (Associate Professor in Saraswati Institute of Technology and Management), akhimohan@yahoo.com 9839397126
Second Author – Ms Sanju Tiwari, (Sr.Lecturer in in Saraswati Institute of Technology and Management), sanju.tiwari.2007@gmail.com,8081634089