Extending XML Schemas

Table of Contents

Issue

What is Best Practice for checking instance documents for constraints that are not expressable by XML Schemas?

Introduction

XML Schemas is very powerful. However, it is not "all powerful". There are many constraints which cannot be expressed with XML Schemas. Here are some examples:

- Ensure that the value of the aircraft <Elevation> element is greater than the value of the obstacle <Height> element.
- Ensure that:
  - if the value of the attribute, mode, is "water" then the value of the element <Transportation> is either airplane or hot-air balloon.
  - if the value of the attribute, mode, is "air" then <Transportation> is either boat or hovercraft.
  - if the value of the attribute, mode, is "ground" then <Transportation> is either car or bicycle.
- Ensure that the value of <PaymentReceived> is equal to the value of <PaymentDue>, where these elements are in separate documents!

To check all these constraints we will need to supplement XML Schemas with another tool.

Example. Consider this simple instance document:

```
<?xml version="1.0"?>
<Demo xmlns="http://www.demo.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.demo.org demo.xsd">
    <A>10</A>
    <B>20</B>
</Demo>
```

With XML Schemas we can check the following constraints:

- the Demo (root) element contains a sequence of elements, A followed by B
- the A element contains an integer
- the B element contains an integer

In fact, here's an XML Schema which implements 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="Demo">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="A" type="xsd:integer"/>
            <xsd:element name="B" 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 A must be greater than the value of B

So what do we do to check this constraint? (Interestingly, for the above instance document, the XML Schema that is shown would accept it as valid, whereas, in fact it is not since the value of A is less than the value of B. We need something else to check this constraint.) There are three options.

Three Options for Extending XML Schemas

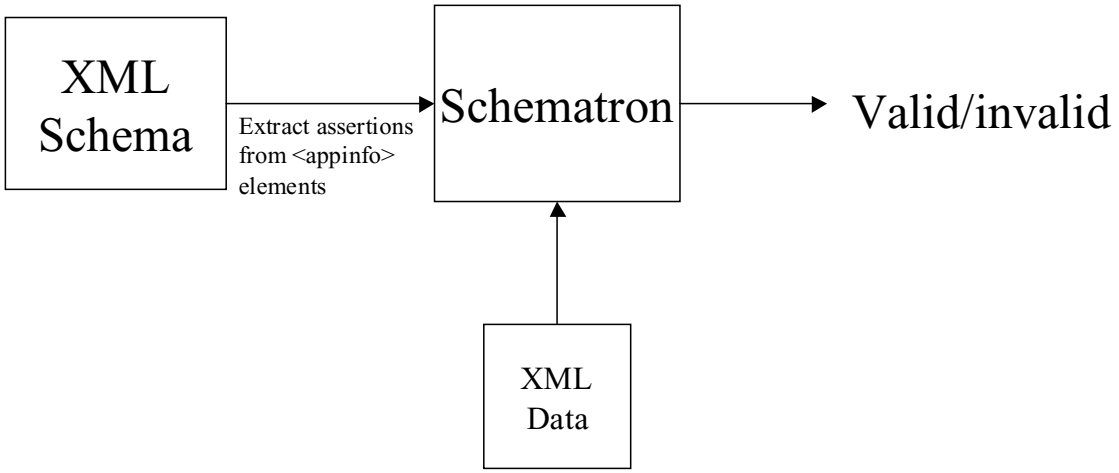
(1) Supplement with Another Schema Language

There are many other schema languages besides XML Schemas:

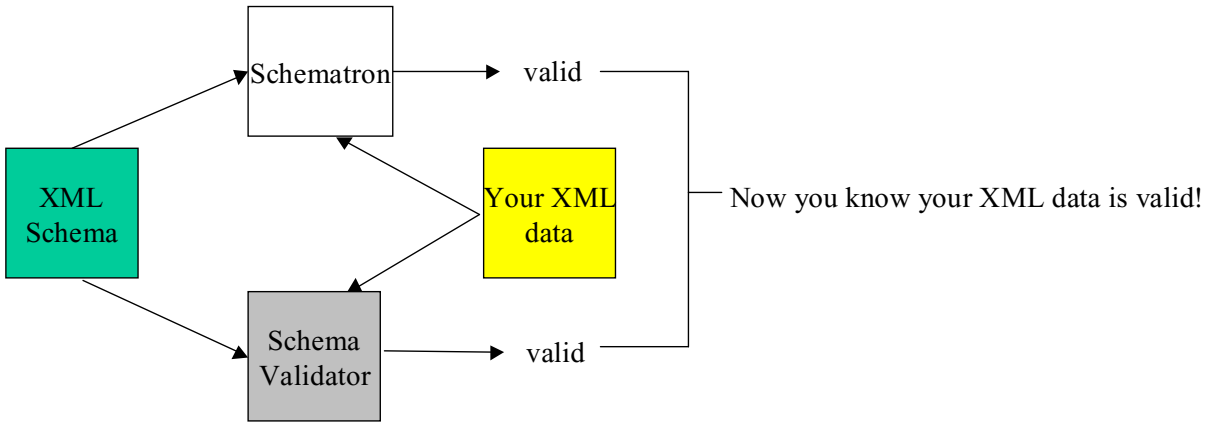
- Schematron
- TREX
- RELAX
- SOX
- XDR
- HOOK
- DSD
- Assertion Grammars
- xlinkit

Thus, the first option is to use one (or more) of these schema languages to express the additional constraints. Let's look at one of these languages - Schematron.

Using Schematron you embed the additional constraints (as "assertions") within the schema document (within <appinfo> elements). A Schematron engine will then extract the assertions and validate the instance document against the assertions.



Thus, here's the architecture for determining if your data meets all constraints:



With Schematron

- state a constraint using an <assert> element
- a <rule> is comprised of one or more <assert> elements
- a <pattern> is comprised of one or more <rule> elements.

```
<pattern>
   <rule>
      <assert> … </assert>
      <assert> … </assert>
   </rule>
</pattern>
```

The <pattern> element is embedded within an XML Schema <appinfo> element.

Here's an example of an assertion:

```
<assert test="d:A &gt; d:B">A should be greater than B</assert>
```

XPath expression Text description of the constraint

In the <assert> element you express a constraint using an XPath expression. Additionally, you state the constraint in natural language. The later helps make the schemas self-documenting.

The <rule> element is used to specify the context for the <assert> elements.

```
<rule context="d:Demo">
   <assert test="d:A &gt; d:B">A should be greater than B</assert>
</rule>
```

This is read as: "Within the context of the Demo element we assert that the A element should be greater than the B element."

You can associate an <assert> with a <diagnostic> element. The <diagnostic> element is used for printing error messages when the XML data fails the assertion. The <diagnostic> element is embedded within a <diagnostics> element, which immediately follows the <pattern> element.

```
<pattern name="Check A greater than B">
    <rule context="d:Demo">
        <assert test="d:A &gt; d:B" diagnostics="lessThan">
            A should be greater than B
        </assert>
    </rule>
</pattern>
<diagnostics>
    <diagnostic id="lessThan">
        Error! A is less than B
        A = <value-of select="d:A"/>
        B = <value-of select="d:B"/>
    </diagnostic>
</diagnostics>
```

To identify the schematron elements, they must be namespace-qualified with sch:.

Here's what demo.xsd looks like after enhancing it with the Schematron elements: (next page)

The schema document shown earlier is enhanced 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="Demo">
      <xsd:annotation>
         <xsd:appinfo>
            <sch:pattern name="Check A greater than B">
               <sch:rule context="d:Demo">
                  <sch:assert test="d:A &gt; d:B" diagnostics="lessThan">A should be greater than B</sch:assert>
               </sch:rule>
            </sch:pattern>
            <sch:diagnostics>
               <sch:diagnostic id="lessThan">
                  Error! A is less than B. A = <sch:value-of select="d:A"/>  B = <sch:value-of select="d:B"/>
               </sch:diagnostic>
            </sch:diagnostics>
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="A" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="B" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Schematron will extract the directives out of the schema document to create a Schematron schema. Schematron will then validate the instance document against the Schematron schema.

The key points to note about using Schematron are:

The additional constraints are embedded in <appinfo> elements within the XML Schema document

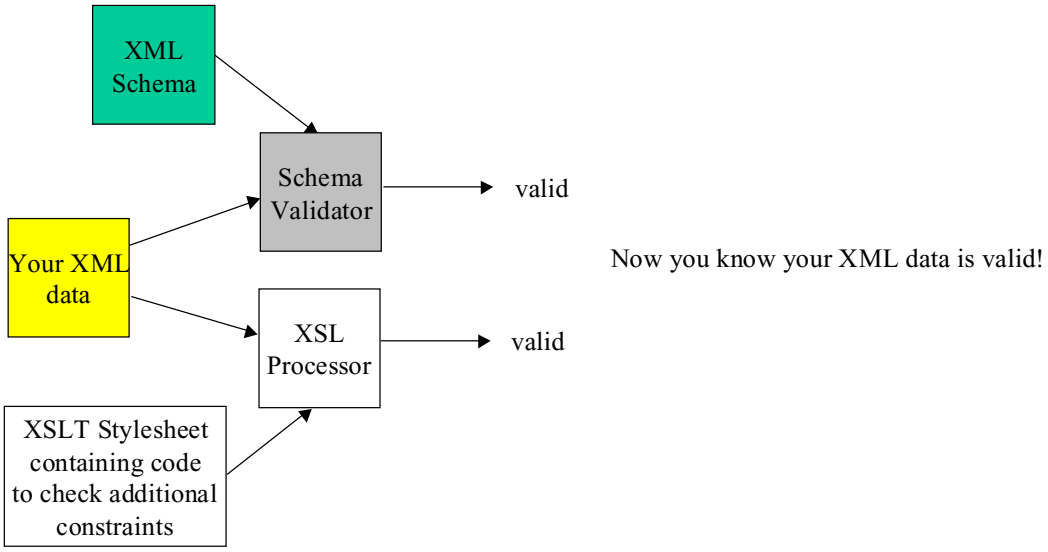
The constraints are expressed using <assert> elements

(2) Write Code to Express Additional Constraints

The second option is to write some Java, Perl, C++, etc code to check additional constraints.

(3) Express Additional Constraints with an XSLT/XPath Stylesheet

The third option is to write a stylesheet to check the constraints.



For example, the following stylesheet checks instance documents to see if the contents of the A element is greater than the contents of the B element:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:d="http://www.demo.org"
                version="1.0">

   <xsl:output method="text"/>

   <xsl:template match="/">
      <xsl:if test="/d:Demo/d:A &lt; /d:Demo/d:B">
         <xsl:text>Error! A is less than B</xsl:text>
         <xsl:text>&#xD;&#xA;</xsl:text>  <!-- carriage return -->
         <xsl:text>A = </xsl:text><xsl:value-of select="/d:Demo/d:A"/>
         <xsl:text>&#xD;&#xA;</xsl:text>   <!-- carriage return -->
         <xsl:text>B = </xsl:text><xsl:value-of select="/d:Demo/d:B"/>
      </xsl:if>
      <xsl:if test="/d:Demo/d:A &gt;= /d:Demo/d:B">
         <xsl:text>Instance document is valid</xsl:text>
      </xsl:if>
   </xsl:template>
</xsl:stylesheet>
```

Upon running this stylesheet on the above XML data the following output is generated:

```
Error! A is less than B. A = 10, B = 20
```

This is exactly what is desired. Thus, the methodology for this third option is:

– check as many constraints as you can using XML Schemas

– for all other constraints write a stylesheet to do the checking

If both the schema validator and the XSL processor generate a positive output then you know that your instance document is valid. This combination of XML Schemas plus stylesheets provides for a powerful constraint checking mechanism.

Advantages/Disadvantages of the Three Options

Advantages

Collocated Constraints: Above we saw how Schematron can be used to express additional constraints. We saw that you embed the Schematron directives within the XML Schema document. There is something very appealing about having all the constraints expressed within one document rather than being dispersed over multiple documents. [This ability to collocate constraints within the schema document is a feature of Schematron. The other schema languages do not have this capability.]

Simplicity: Many of the schema languages were created in reaction to the complexity and limitations of XML Schemas. Consequently, most of them are relatively simple to learn and use.

Disadvantages

Multiple Schema Languages may be Required: Each schema language has its own capabilities and limitations. Multiple schema languages may be required to express all the additional constraints. For example, while Schematron is very powerful it is not able to express all constraints (for an example, see the ISBN simpleType definition on http://www.xfront.com). Also, Schematron forces you to go through many contortions to express your assertion. This is due to the fact that it does not have loops and variables.

Yet Another Vocabulary (YAV): There are many schema languages, each with its own vocabulary and semantics. How do you find a schema language with the capability to express your problem's additional constraints? You have to take the time to learn each of the schema languages. Hopefully, you will find one that supports expression of your constraints. Although relatively easy to learn and use, it still takes time to learn a new vocabulary and semantics.

Questionable Long Term Support: In most cases the schema languages listed above were created by a single author. These authors are busy, very bright people. Someday their interests will move to something else. At that time you may be left with a product which is no longer supported. [Editor's Note: Schematron is basically a few XSLT/XPath stylesheets. Consequently, Schematron will be supported as long as there are XSL processors. Also, the author of RELAX has publically promised to support RELAX for the next five years.]

(2) Write Code to Express Additional Constraints

Advantages

Full Power of a Programming Language: The advantage of this option is that with a single programming language you can express all the additional constraints.

Disadvantages

Not Leveraging other XML Technologies: There are other XML technologies that could be used to express the additional constraints in a declarative manner, without going through the compiling, linking, executing effort.

(3) Express Additional Constraints with an XSLT/XPath Stylesheet

Advantages

Application Specific Constraint Checking: Each application can create its own stylesheet to check constraints that are unique to the application. We can enhance the schema without touching it!

Core Technology: XSLT/XPath is a "core technology" which is well supported, well understood, and with lots of material written on it.

Expressive Power: XSLT/XPath is a very powerful language. Most, if not every, constraint that you might ever need to express can be expressed using XSLT/XPath. Thus you don't have to learn multiple schema languages to express your additional constraints

Long Term Support: XSLT/XPath is well supported, and will be around for a long time.

Disadvantages

Separate Documents: With this approach you will write your XML Schema document, then you will write a separate XSLT/XPath document to express additional constraints. Keeping the two documents in synch needs to be carefully managed.