NEMSIS TAC Whitepaper

NEMSIS V3 Schematron Developer's Guide

Date

November 23, 2011 (FINAL)

Authors

Su Shaoyu – NEMSIS Lead Developer

Introduction

This document will briefly explain the adoption of Schematron validation in NEMSIS v3. To improve data quality, there are two validation stages: general XML validation and NEMSIS business rule validation. Any error in general XML validation will result in the rejection of the submission. Errors in business rule validation will result in rejection or warning, depending on how critical the nature of the error is.

General Description

Stage 1: General XML Validation

The submitted NEMSIS XML file will be validated against a published NEMSIS v3 XSD to check whether it is "well-formed" and conforms to the schema (XSD). This is a well-known procedure and there are many off-the-shelf applications which can be utilized. The NEMSIS TAC customized a Java's SAX parser to execute the validation.

After Stage 1, if it is determined that the submitted XML file is not "valid", the file is rejected and no further information is processed or reported to the end-user. Only valid files will continue to Stage 2.

Stage 2: Business Rule Validation

Stage 1 validation only checks XML file structure and grammar. It does not indicate if the data submitted are "good" or "meaningful". For example, element eMedications.03 – Medication Given is defined as "the medication given to the patient". According to the XSD, this element can accept "NOT Values" (NV), "Pertinent Negative Values" (PN), and "Nillable" (Nil) attributes. The possible combinations for documenting this element include the following options:

X = documented, F = False; T = True

Option	Code	PN	NV	Nil	Example
1	x			F	<eMedications.03 xsi:nil="false">02345</eMedications.03>
2	X	X		F	<eMedications.03 xsi:nil="false" PN="8801009">03456</eMedications.03>
3	X		X	F	<eMedications.03 xsi:nil="false"

					NV="7701003">04567</eMedications.03>
4	X	x	X	F	<eMedications.03 xsi:nil="false" PN="8801009" NV="7701003">05678</eMedications.03>
5	X				<eMedications.03>12345</eMedications.03>
6	X	X			<eMedications.03 PN="8801009">23456</eMedications.03>
7	X		X		<eMedications.03 NV="7701003">34567</eMedications.03>
8	X	x	X		<eMedications.03 PN="8801009" NV="7701003">45678</eMedications.03>
9				T	<eMedications.03 xsi:nil="true"></eMedications.03>
10		X		T	<eMedications.03 xsi:nil="true" PN="8801009"></eMedications.03>
11			X	T	<eMedications.03 xsi:nil="true" NV="7701003"></eMedications.03>
12		x	x	T	<eMedications.03 xsi:nil="true" PN="8801011" NV="7701003"></eMedications.03>

All these combinations will pass Stage 1 XML validation – but those highlighted in yellow are not accurate in the documentation of care in the EMS environment.

In addition, in a NEMSIS XML file, many elements are related. A value in one element may apply some restriction on possible value selections for another element. For example, Times.05 – Unit En Route Date/Time is defined as "The date/time the unit responded; that is, the time the vehicle started moving" and eTimes.03 – Unit Notified by Dispatch Date/Time is defined as "The date/time the responding unit was notified by dispatch". It is reasonable to assume eTimes.03 <= eTimes.05. Any completed record where eTimes.03 > eTimes.05 clearly indicates a documentation error.

The purpose of Stage 2 validation is to capture these potential "violations" by checking the file against a set of pre-defined business rules. This ensures the data we collect are meaningful.

Implementation:

Stage 1: General XML Validation

All modern programming languages and many applications could be used to execute XML validation. There are only two objects needed: the XML file and the schema set (XSD or DTD). For example, several lines of Java code will successfully perform validation using Java's built-in JAXP package:

```
try {
   //create schema W3C factory:
   SchemaFactory factory = SchemaFactory.newInstance("http://www.w3.org/2001/XMLSchema");
   // create schema by reading it from an XSD file:
   Schema schema = factory.newSchema(new StreamSource("sample.xsd"));
   Validator validator = schema.newValidator();
   // perform validation:
   validator.validate(new StreamSource("sample.xml"));
   System.out.println("sample.xml is valid.");
}catch (SAXException ex) {
   System.out.println("sample.xml  is not valid.");
   //process validation error if "sample.xml" file is not valid...
} catch (Exception ex) {
  //process other errors, such as file not exist, etc…
}
```

Certainly, there is more to implement. For example, it is unacceptable to simply say "your file is not valid" – at least, we should provide some details about which parts of the submitted XML file violates the schema. In order to provide useful information to the end user, additional development is necessary for a successful and functional software solution. Details related to the XML file violation should be provided to the end user in a "friendly" message, allowing them to locate and fix the file. The NEMSIS TAC logs all validation errors to a database table for reporting purposes. A report is provided back to the end user related to the errors in the XML file.

XML validation techniques are so mature that there is no point discussing them further – if I do so, I will lose the attention of our software developer community.

Stage 2: Business Rule Validation

As discussed above, XML structure/grammar is only the first step to enforce data integrity. The development of an initial set of national "structural" business rules is the next step.

There are many ways/places to enforce these business rules. Business rules validation could be executed directly on NEMSIS XML files (using Schematron, which will be discussed later), or during/after loading EMS data to a database (using ETL, http://en.wikipedia.org/wiki/Extract,_transform,_load), or even during the export of the final dataset (for example, it might be necessary to trim off offending records when exporting the data from a database to the XML file). And we might need to create several sets of business rules to enforce at various data processing stages. **The NEMSIS TAC requires that all software tested for compliance apply a standard Schematron-compatible validation file during (or before) the process of completing and/or closing a patient record.**

In this document, we will focus on Schematron, a popular rule-based "language" for business rule validation/reporting. Its official web site is http://www.schematron.com/. There are tutorials/samples online. Please reference the tutorial by Dave Pawson as it is very helpful for those not familiar with Schematron.

1. Setup the environment:
    a. Install Java runtime!
    b. Download http://www.schematron.com/tmp/iso-schematron-xslt2.zip and unzip it. It contains 7 XSL files.
    (There are two versions of ISO Schematron released, one is for XSLT1 processors and the other is for XSLT2 processors. As documented on the Schematron official web site, more features are provided in the release for XSLT2, hence it's recommended.)
    (Some advantages to using XSLT 2.0 over 1.0 are outlined at http://www.devx.com/enterprise/Article/43749/1954. I strongly agree with the section talking about using @use-when attribute, which has proven to be very helpful to me.)
    c. Download SAXON HE (home edition) from http://saxon.sourceforge.net/, unzip it. The file saxon9he.jar is included in the zip file. Please note that the name of the downloaded SAXON HE jar file may be different, as in my example below, where the file is named saxon9304he.jar.

d. Put all these files into the same folder. For example, it is C:\ssu\Projects\NEMSIS\v3\schematron\ for my test. Please note that my computer runs on Windows. Mac/Linux/Unix users need to adjust the path name accordingly.
e. Let's create a batch file, mytest.bat, to run the business rules. There is nothing fancy here.

mytest.bat

```
@echo off

echo
echo
echo Usage: mytest %%1 = iso schematron file, no extension.
echo  %%2 is the input xml file, with the extension.
echo E.g. mytest input input.xml will produce input.report.xml as output for input.xml

del stage1.sch
del stage2.sch
del stage3.xsl
echo Generate the stylesheet from %1
echo %time%

echo stage 1----------------
java  -cp .;.\saxon9304he.jar net.sf.saxon.Transform -o:stage1.sch -s:%1.sch -xsl:iso_dsdl_include.xsl
echo %time%

echo stage 2----------------
java  -cp .;.\saxon9304he.jar net.sf.saxon.Transform -o:stage2.sch -s:stage1.sch -xsl:iso_abstract_expand.xsl
echo %time%

echo stage 3----------------
java  -cp .;.\saxon9304he.jar net.sf.saxon.Transform -o:stage3.xsl -s:stage2.sch -xsl:iso_svrl_for_xslt2.xsl
echo %time%

echo Now run the input file %2 against the generated stylesheet stage3.xsl to produce %1%.report.xml
java  -cp .;.\saxon9304he.jar net.sf.saxon.Transform -t -o:%1.report.xml -s:%2 -xsl:stage3.xsl
echo %time%
```

2. Running sample files
    a. In the folder C:\ssu\Projects\NEMSIS\v3\schematron\, create a sample XML file with name “sample1.xml”:

sample1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<EMSDataSet xmlns="http://www.nemsis.org"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.nemsis.org
file:/whatever/EMSDataSet_v3.xsd">
    <Header>
        <dAgency.01>dAgency.01</dAgency.01>
        <dAgency.04>01</dAgency.04>
        <dAgency.11>HAHA</dAgency.11>
        <dConfiguration.ConfigurationGroup>
            <dConfiguration.01>01</dConfiguration.01>
            <dConfiguration.02>FUNNY</dConfiguration.02>
        </dConfiguration.ConfigurationGroup>
    </Header>
</EMSDataSet>
```

And a Schematron file, “sample1.sch”:

sample1.sch

```
<?xml version="1.0" encoding="UTF-8"?>
<iso:schema xmlns:iso="http://purl.oclc.org/dsdl/schematron" queryBinding="xslt">
  <iso:ns prefix="nem" uri="http://www.nemsis.org"/>
  <iso:ns prefix="xsl" uri="http://www.w3.org/1999/XSL/Transform"/>
  <iso:title>Sample Schematron Business Rule Validation</iso:title>
```

```
   <!-- RULE 0: just want to satisfy the assert -->
   <iso:pattern name="RULE00">
      <iso:title>Just want to make sure dAgency.01 has some value</iso:title>
      <iso:rule context="nem:header">
         <iso:assert test="nem:dAgency.01" >
               dAgency.01 should have some value
         </iso:assert>
      </iso:rule>
   </iso:pattern>

   <!-- dConfiguration rules          -->
   <!-- RULE 1: If dAgency.04 equals dConfiguration.01, then the value supplied
   in dAgency.11 must equal one of the values of dConfiguration.02 -->
   <iso:pattern name="RULE01">
      <iso:title>dAgency.11 must equal one of the values of dConfiguration.02</iso:title>
      <iso:rule context="nem:dAgency.11">
         <iso:let name="codeValue" value="normalize-space(.)" />
         <iso:let name="config1" value="normalize-
space(../nem:dConfiguration.ConfigurationGroup/nem:dConfiguration.01/.)" />
         <iso:let name="agency4" value="normalize-space(../nem:dAgency.04/.)" />
         <iso:let name="config2" value="../nem:dConfiguration.ConfigurationGroup/nem:dConfiguration.02" />
         <iso:assert test="if ($config1 = $agency4) then (if ($config2[normalize-space(text()) = $codeValue]) then true() else
false()) else true()" >
               Value '<iso:value-of select="$codeValue" />' supplied in dAgency.11 NOT found in dConfiguration.02
         </iso:assert>
         <iso:assert test="$codeValue = 2" >
               Value '<iso:value-of select="$codeValue" />' supplied in dAgency.11 does not equal to 2
         </iso:assert>
      </iso:rule>
   </iso:pattern>
</iso:schema>
```

Now, open a command window, go to directory C:\ssu\Projects\NEMSIS\V3\schematron, and run this command:

.\mytest.bat sample1 sample1.xml

If you did everything right, the output might look like this:

```
ECHO is off.
ECHO is off.
Usage: mytest %1 = iso schematron file, no extension.
 %2 is the input xml file, with the extension.
E.g. mytest input input.xml will produce input.report.xml as output
Generate the stylesheet from sample1
15:08:42.05
stage 1----------------
Warning: at xslt:stylesheet on line 120 column 3 of iso_dsdl_include.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
15:08:42.96
stage 2----------------
Warning: at xslt:stylesheet on line 73 column 8 of iso_abstract_expand.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
15:08:43.71
stage 3----------------
Warning: at xsl:stylesheet on line 150 column 2 of iso_svrl_for_xslt2.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
15:08:44.64
Now run the input file sample1.xml against the generated stylesheet stage3.xsl to produce sample1.report.xml
Saxon-HE 9.3.0.4J from Saxonica
Java version 1.6.0_07
Warning: at xsl:stylesheet on line 8 column 31 of stage3.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
Stylesheet compilation time: 375 milliseconds
Processing file:/C:/ssu/Projects/NEMSIS/V3/schematron/sample1.xml
Using parser com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
Building tree for file:/C:/ssu/Projects/NEMSIS/V3/schematron/sample1.xml using class net.sf.saxon.tree.tiny.TinyBuilder
```

```
Tree built in 0 milliseconds
Tree size: 25 nodes, 23 characters, 1 attributes
Execution time: 78ms
Memory used: 3006352
NamePool contents: 38 entries in 37 chains. 11 prefixes, 11 URIs
15:08:45.39
```

the generated sample1.report.xml looks like

sample1.report.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<svrl:schematron-output xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        xmlns:xhtml="http://www.w3.org/1999/xhtml"
                        xmlns:schold="http://www.ascc.net/xml/schematron"
                        xmlns:iso="http://purl.oclc.org/dsdl/schematron"
                        xmlns:nem="http://www.nemsis.org"
                        xmlns:svrl="http://purl.oclc.org/dsdl/svrl"
                        title="Sample Schematron Business Rule Validation"
                        schemaVersion=""><!--


                -->
<svrl:ns-prefix-in-attribute-values uri="http://www.nemsis.org"
prefix="nem"/>
   <svrl:ns-prefix-in-attribute-values uri="http://www.w3.org/1999/XSL/Transform"
prefix="xsl"/>
   <svrl:active-pattern
document="file:/C:/ssu/Projects/NEMSIS/V3/schematron/sample1.xml"
                        name="Just want to make sure dAgency.01 has some value"/>
   <svrl:active-pattern
document="file:/C:/ssu/Projects/NEMSIS/V3/schematron/sample1.xml"
                        name="dAgency.11 must equal one of the values of
dConfiguration.02"/>
   <svrl:fired-rule context="nem:dAgency.11"/>
   <svrl:failed-assert test="if ($config1 = $agency4) then (if
($config2[normalize-space(text()) = $codeValue]) then true() else false()) else
true()"
                       location="/*[local-name()='EMSDataSet']/*[local-
name()='Header']/*[local-name()='dAgency.11']">
      <svrl:text>
                              Value 'HAHA' supplied in dAgency.11 NOT found in
dConfiguration.02
            </svrl:text>
   </svrl:failed-assert>
   <svrl:failed-assert test="$codeValue = 2"
                       location="/*[local-name()='EMSDataSet']/*[local-
name()='Header']/*[local-name()='dAgency.11']">
      <svrl:text>
                      Value 'HAHA' supplied in dAgency.11 does not equal to 2
            </svrl:text>
   </svrl:failed-assert>
</svrl:schematron-output>
```

There are several important points we can conclude from this sample exercise.

i. Schematron is nothing more than a multiple-step XSLT transformation, which transforms a XML file to generate a report. In the mytest.bat file, the input sample1.sch file goes through three steps to become a real stylesheet (XSL) file. This generated stage3.xsl file is used to transform sample1.xml to get the result report in XML format. (There is a beautiful diagram in Dave Pawson's tutorial that explains this flow.) Actually, since our sample1.sch Schematron file doesn't

have a include/import/abstract pattern, it is safe to skip the first two transformation steps. If you are very good at writing XSL, especially XPATH, you can directly write something similar to stage3.xsl and totally forget about Schematron!

ii. A Schematron transformation is not schema-aware: you might notice that the sample.xml is not a valid NEMSIS XML file at all. This doesn't matter, as long as it is well-formatted. XML validation is of no concern to Schematron. (You might have noticed the enlarged font of xsi:schemaLocation part in sample1.xml file. The file path is not a valid path!)

iii. The final sample1.report.xml file needs further processing to extract error messages. Nobody is interested in reading a "raw" XML report. This means another step of XML parsing is required in most cases.
(There are some tricks to improving readability. For example, besides the "assert" element, which is used in our validation process, Schematron also defines a "report" element, which can be used for diagnosis. Combined usage of both "assert" and "report" elements can produce "prettier" reports. But this is out of the scope of this document.)

iv. Since Schematron is a XSLT implementation, it uses XPath and XQuery: this means that its memory requirement depends on the size of input XML file. (Check http://download.oracle.com/docs/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/SJSXP2.html for comparison of various JAXP APIs.) Here, "XML file size" is not about the XML file's size as stored on a hard drive, but is about the complexity of the XML structure. "Tree size" is a better measurement for XML "file size" – in the example above, sample1.xml is read into memory and a DOM tree is built to represent the file. More tree nodes means more memory required. If you need further information about DOM parsers, Google is your friend.
For our tiny sample1.xml, memory usage of 3Mb at the last transformation step is not small. To further test performance of the Schematron transformation, we can replicate the header in sample1.xml to have many more nodes. Based on the release XSD candidate file at the time of this writing, I also constructed some sample NEMSIS XML files which have 1 or many NEMSIS EMS records, with all elements fully populated. For these files, only the testing rule of "*dAgency.11 does not equal to 2*" was triggered. On the other hand, every header in sample1.xml triggers two rules. So for the same header count (for example, a 1,000 headers), sample1.xml will have 2,000 errors, while the NEMSIS EMS sample file will have only 1,000 errors.

Table 1

Sample1.xml					
	Header Count	Node Count	Memory Used	Time Used	Report File Size (MB)
	1	25	3 MB	78 ms	2 KB
	1,000	21,170	4 MB	781 ms	741 KB
	10,000	211,670	8 MB	47 sec	7.4 MB
	100,000	2,116,670	136 MB	1h 8m	74.3 MB
NEMSIS EMS, all elements populated					
	Header Count	Node Count	Memory Used	Time Used	Report File Size
	1	5624	3 MB	203 ms	2 KB
	2	11,243	3.9 MB	204 ms	2 KB
	5	28,100	4.3 MB	219 ms	3 KB
	10	56,196	5.6 MB	297 ms	5 KB
	20	112,388	8.5 MB	438 ms	9 KB
	50	280,963	22.8 MB	906 ms	20 KB
	100	561,921	22. 1 MB	1.5 s	39 KB
	500	2,809,588	117 MB	7.2 s	188 KB
	1,000	5,619,171	342 MB	14.4 s	374 KB
	2,000	11,238,338	708 MB	26.7 s	747 KB
	5,000	28,095,838	966 MB	64.8 s	1.86 MB
	10,000	I don't have the values, and all I know is that memory usage is over 1.6 GB, which is the maximum I can allocate on my desktop.			

From table 1, we observe that memory usage of Schematron grows with the number of elements in a XML file (roughly speaking). Also, the execution time depends not only on number of elements, but also on how many errors are in the resulting report file. For sample1.xml, when the header count increases from 10k to 100k, the execution time jumps from 47 seconds to 1 hour and 8 minutes (4108 seconds), which is quite alarming. Apparently, the transformer we used, SAXON, is not good at handling a large number of errors.

(To increase memory allocated for Java, use –mx option.)

b. Let's create another test file, sample2.xml:

Sample2.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<EMSDataSet xsi:schemaLocation="http://www.nemsis.org
file:/watever/EMSDataSet_v3.xsd" xmlns="http://www.nemsis.org"
xmlns:n1="http://www.nemsis.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
    <eMedications.03 xsi:nil="false">02345 Good</eMedications.03>
    <eMedications.03 xsi:nil="false" PN="8801009">03456 Good</eMedications.03>
    <eMedications.03 xsi:nil="false" NV="7701003">04567 Bad Case
30</eMedications.03>
    <eMedications.03 xsi:nil="false" PN="8801009" NV="7701003">05678 Bad case
40</eMedications.03>
    <eMedications.03>12345 Good</eMedications.03>
    <eMedications.03 PN="8801009">23456 Good</eMedications.03>
    <eMedications.03 NV="7701003">34567 Bad case 3</eMedications.03>
    <eMedications.03 PN="8801009" NV="7701003">45678 Bad Case 4</eMedications.03>
    <eMedications.03 xsi:nil="true"></eMedications.03>
    <eMedications.03 xsi:nil="true" PN="8801009 Bad Case 6"></eMedications.03>
    <eMedications.03 xsi:nil="true" NV="7701003"></eMedications.03>
    <eMedications.03 xsi:nil="true" PN="8801011 Bad case 8"
NV="7701003"></eMedications.03>


    <funny.99 xsi:nil="false">02345 Good</funny.99>
    <funny.99 xsi:nil="false" PN="8801009">03456 Good</funny.99>
    <funny.99 xsi:nil="false" NV="7701003">04567 Bad Case 30 funny</funny.99>
    <funny.99 xsi:nil="false" PN="8801009" NV="7701003">05678 Bad case 40
funny</funny.99>
    <funny.99>12345 Good</funny.99>
    <funny.99 PN="8801009">23456 Good</funny.99>
    <funny.99 NV="7701003">34567 Bad case 3 funny</funny.99>
    <funny.99 PN="8801009" NV="7701003">45678 Bad Case 4 funny</funny.99>
    <funny.99 xsi:nil="true"></funny.99>
    <funny.99 xsi:nil="true" PN="8801009 Bad Case 6 funny"></funny.99>
    <funny.99 xsi:nil="true" NV="7701003"></funny.99>
    <funny.99 xsi:nil="true" PN="8801011 Bad case 8 funny"
NV="7701003"></funny.99>

</EMSDataSet>
```

and a Schematron file, sample2.sch:

sample2.sch

```
<?xml version="1.0" encoding="utf-8"?>
<iso:schema xmlns="http://purl.oclc.org/dsdl/schematron"
xmlns:iso="http://purl.oclc.org/dsdl/schematron" xmlns:nem="http://www.nemsis.org"
queryBinding="xslt2">
  <iso:ns prefix="nem" uri="http://www.nemsis.org"/>
  <iso:ns prefix="xsl" uri="http://www.w3.org/1999/XSL/Transform"/>
  <iso:ns prefix="xsi" uri="http://www.w3.org/2001/XMLSchema-instance"/>
  <iso:title>Test ISO schematron file. Introduction mode</iso:title>
  <iso:pattern abstract="true" id="national">
    <iso:rule context="$realElem">
      <iso:let name="nvValue" value="normalize-space(@NV)" />
      <iso:let name="pnValue" value="normalize-space(@PN)" />
      <iso:let name="nilValue" value="normalize-space(@xsi:nil)" />
      <iso:let name="selfValue" value="normalize-space(.)" />
      <iso:let name="nvHasValue" value="if (normalize-space(@NV)) then true() else
false()" />
      <iso:let name="pnHasValue" value="if (normalize-space(@PN)) then true() else
false()" />
      <iso:let name="nilRealValue" value="if (normalize-space(@xsi:nil) = 'true')
then true() else false()" />
      <iso:assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue)
and $nvHasValue)) then false else true()">
```

```
            $PREFIX: the PN, NV, NIL combination for element <iso:name/> is not
valid.
          NV has value of '<iso:value-of select="$nvValue" />';
          PN has value of '<iso:value-of select="$pnValue" />';
          NIL has value of '<iso:value-of select="$nilValue" />';
          Element value is '<iso:value-of select="$selfValue" />';
          hahha; position is '<iso:value-of select="position()" />';
      </iso:assert>
    </iso:rule>
  </iso:pattern>

  <iso:pattern is-a="national" id="eMedication.03" description="PN/NV/NIL
combination not valid"  errorlevel="FATAL">
    <iso:param name="realElem" value="nem:eMedications.03"/>
    <iso:param name="PREFIX" value="National Schematron Error"/>
  </iso:pattern>

  <iso:pattern is-a="national" id="JustForFun" description="habhabahabahaba"
errorlevel="DONOTMATTER">
    <iso:param name="realElem" value="nem:funny.99"/>
    <iso:param name="PREFIX" value="Bunny Error"/>
  </iso:pattern>

</iso:schema>
```

c. Open a command window, go to C:\ssu\Projects\NEMSIS\v3\schematron\, then type in this command:

.\mytest.bat sample2 sample2.xml

d. The output from running the batch file should look like this:

```
ECHO is off.
ECHO is off.
Usage: build %1 = iso schematron file, no extension.
 %2 is the input xml file, with the extension.
E.g. build input input.xml will produce input.report.xml as output
Generate the stylesheet from sample
10:01:10.47
stage 1----------------
Warning: at xslt:stylesheet on line 120 column 3 of iso_dsdl_include.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
10:01:11.39
stage 2----------------
Warning: at xslt:stylesheet on line 73 column 8 of iso_abstract_expand.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
10:01:12.19
stage 3----------------
Warning: at xsl:stylesheet on line 150 column 2 of iso_svrl_for_xslt2.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
10:01:13.14
Now run the input file sample.xml against the generated stylesheet stage3.xsl to produce sample.report.xml
Saxon-HE 9.3.0.4J from Saxonica
Java version 1.6.0_07
Stylesheet compilation time: 390 milliseconds
Processing file:/C:/ssu/Projects/NEMSIS/v3/schematron/sample2.xml
Using parser com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
Building tree for file:/C:/ssu/Projects/NEMSIS/v3/schematron/sample2.xml using class
net.sf.saxon.tree.tiny.TinyBuilder
Tree built in 0 milliseconds
Tree size: 72 nodes, 236 characters, 41 attributes
Execution time: 94ms
Memory used: 2550968
NamePool contents: 35 entries in 35 chains. 13 prefixes, 11 URIs
10:01:13.94
```

e. This is the report file, sample2.report.xml:

sample2.report.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<svrl:schematron-output xmlns:schold="http://www.ascc.net/xml/schematron"
                xmlns:iso="http://purl.oclc.org/dsdl/schematron"
                xmlns:saxon="http://saxon.sf.net/"
                xmlns:nem="http://www.nemsis.org"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                xmlns:svrl="http://purl.oclc.org/dsdl/svrl"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:xhtml="http://www.w3.org/1999/xhtml"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                title="Test ISO schematron file. Introduction mode"
                schemaVersion=""><!--


                        --><svrl:ns-prefix-in-attribute-values uri="http://www.nemsis.org" prefix="nem"/>
  <svrl:ns-prefix-in-attribute-values uri="http://www.w3.org/1999/XSL/Transform" prefix="xsl"/>
  <svrl:ns-prefix-in-attribute-values uri="http://www.w3.org/2001/XMLSchema-instance" prefix="xsi"/>
  <svrl:active-pattern document="file:/C:/ssu/Projects/NEMSIS/V3/schematron/sample2.xml"
                id="eMedication.03"
                name="eMedication.03"/>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
                location="/*:EMSDataSet[namespace-
uri()='http://www.nemsis.org'][1]/*:eMedications.03[namespace-uri()='http://www.nemsis.org'][3]">
    <svrl:text>
        National Schematron Error: the PN, NV, NIL combination for element eMedications.03 is not valid.
        NV has value of '7701003';
        PN has value of '';
        NIL has value of 'false';
        Element value is '04567 Bad Case 30';
        hahha; position is '3';

      </svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
                location="/*:EMSDataSet[namespace-
uri()='http://www.nemsis.org'][1]/*:eMedications.03[namespace-uri()='http://www.nemsis.org'][4]">
    <svrl:text>
        National Schematron Error: the PN, NV, NIL combination for element eMedications.03 is not valid.
        NV has value of '7701003';
        PN has value of '8801009';
        NIL has value of 'false';
        Element value is '05678 Bad case 40';
        hahha; position is '4';

      </svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
                location="/*:EMSDataSet[namespace-
uri()='http://www.nemsis.org'][1]/*:eMedications.03[namespace-uri()='http://www.nemsis.org'][7]">
    <svrl:text>
        National Schematron Error: the PN, NV, NIL combination for element eMedications.03 is not valid.
        NV has value of '7701003';
        PN has value of '';
```

```
            NIL has value of '';
            Element value is '34567 Bad case 3';
            hahha; position is '7';

        </svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
            location="/*:EMSDataSet[namespace-
uri()='http://www.nemsis.org'][1]/*:eMedications.03[namespace-uri()='http://www.nemsis.org'][8]">
    <svrl:text>
            National Schematron Error: the PN, NV, NIL combination for element eMedications.03 is not valid.
            NV has value of '7701003';
            PN has value of '8801009';
            NIL has value of '';
            Element value is '45678 Bad Case 4';
            hahha; position is '8';

        </svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
            location="/*:EMSDataSet[namespace-
uri()='http://www.nemsis.org'][1]/*:eMedications.03[namespace-uri()='http://www.nemsis.org'][10]">
    <svrl:text>
            National Schematron Error: the PN, NV, NIL combination for element eMedications.03 is not valid.
            NV has value of '';
            PN has value of '8801009 Bad Case 6';
            NIL has value of 'true';
            Element value is '';
            hahha; position is '10';

        </svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:fired-rule context="nem:eMedications.03"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
            location="/*:EMSDataSet[namespace-
uri()='http://www.nemsis.org'][1]/*:eMedications.03[namespace-uri()='http://www.nemsis.org'][12]">
    <svrl:text>
            National Schematron Error: the PN, NV, NIL combination for element eMedications.03 is not valid.
            NV has value of '7701003';
            PN has value of '8801011 Bad case 8';
            NIL has value of 'true';
            Element value is '';
            hahha; position is '12';

        </svrl:text>
  </svrl:failed-assert>
  <svrl:active-pattern document="file:/C:/ssu/Projects/NEMSIS/V3/schematron/sample2.xml"
             id="JustForFun"
             name="JustForFun"/>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
            location="/*:EMSDataSet[namespace-uri()='http://www.nemsis.org'][1]/*:funny.99[namespace-
uri()='http://www.nemsis.org'][3]">
    <svrl:text>
            Bunny Error: the PN, NV, NIL combination for element funny.99 is not valid.
            NV has value of '7701003';
```

```
            PN has value of '';
            NIL has value of 'false';
            Element value is '04567 Bad Case 30 funny';
            hahha; position is '15';

        </svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
            location="/*:EMSDataSet[namespace-uri()='http://www.nemsis.org'][1]/*:funny.99[namespace-
uri()='http://www.nemsis.org'][4]">
    <svrl:text>
            Bunny Error: the PN, NV, NIL combination for element funny.99 is not valid.
            NV has value of '7701003';
            PN has value of '8801009';
            NIL has value of 'false';
            Element value is '05678 Bad case 40 funny';
            hahha; position is '16';

        </svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
            location="/*:EMSDataSet[namespace-uri()='http://www.nemsis.org'][1]/*:funny.99[namespace-
uri()='http://www.nemsis.org'][7]">
    <svrl:text>
            Bunny Error: the PN, NV, NIL combination for element funny.99 is not valid.
            NV has value of '7701003';
            PN has value of '';
            NIL has value of '';
            Element value is '34567 Bad case 3 funny';
            hahha; position is '19';

        </svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
            location="/*:EMSDataSet[namespace-uri()='http://www.nemsis.org'][1]/*:funny.99[namespace-
uri()='http://www.nemsis.org'][8]">
    <svrl:text>
            Bunny Error: the PN, NV, NIL combination for element funny.99 is not valid.
            NV has value of '7701003';
            PN has value of '8801009';
            NIL has value of '';
            Element value is '45678 Bad Case 4 funny';
            hahha; position is '20';

        </svrl:text>
  </svrl:failed-assert>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:fired-rule context="nem:funny.99"/>
  <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
false else true()"
            location="/*:EMSDataSet[namespace-uri()='http://www.nemsis.org'][1]/*:funny.99[namespace-
uri()='http://www.nemsis.org'][10]">
    <svrl:text>
            Bunny Error: the PN, NV, NIL combination for element funny.99 is not valid.
            NV has value of '';
            PN has value of '8801009 Bad Case 6 funny';
            NIL has value of 'true';
            Element value is '';
```

```
                    hahha; position is '22';

                </svrl:text>
            </svrl:failed-assert>
            <svrl:fired-rule context="nem:funny.99"/>
            <svrl:fired-rule context="nem:funny.99"/>
            <svrl:failed-assert test="if (($nilRealValue and $pnHasValue) or (not($nilRealValue) and $nvHasValue)) then
        false else true()"
                    location="/*:EMSDataSet[namespace-uri()='http://www.nemsis.org'][1]/*:funny.99[namespace-
        uri()='http://www.nemsis.org'][12]">
                <svrl:text>
                    Bunny Error: the PN, NV, NIL combination for element funny.99 is not valid.
                    NV has value of '7701003';
                    PN has value of '8801011 Bad case 8 funny';
                    NIL has value of 'true';
                    Element value is '';
                    hahha; position is '24';

                </svrl:text>
            </svrl:failed-assert>
</svrl:schematron-output>
```

f. This example demonstrates a very powerful mechanism of Schematron: abstract patterns. The basic idea is to think about the properties of a set of "similar" XML elements. In sample2.sch, we design an abstract "national" pattern to check valid combinations of NV/PN/Nil. For any real NEMSIS element, we can create a solid pattern to instantiate the abstract pattern and "pass" it in the NEMSIS element as parameter. The example shown above is to check eMedications.03:

```
<iso:pattern is-a="national" id="eMedication.03" description="PN/ "/> NV/NIL combination not valid"
errorlevel="FATAL">
    <iso:param name="realElem" value="nem:eMedications.03
    <iso:param name="PREFIX" value="National Schematron Error"/>
  </iso:pattern>
```

The pattern for a make-up element "funny.99" is added to show how to reuse the abstract pattern.

This simple and flexible mechanism opens up a great deal of power in adopting Schematron. For the NEMSIS elements, many of them share similar constraints – in another words, the same business rules apply. One abstract pattern can be created to represent one business rule, then a NEMSIS element can be associated dynamically with this business rule by instantiating the abstract patterns. By modifying the syntax of one abstract pattern, the report for all NEMSIS elements on which that business rule applies can be controlled easily. This not only helps increase code reusability, but it also decreases the chance of errors during a copy/paste procedure.

g. You might notice that although we put something into pattern's "errorlevel" and "description" attributes, these values are nowhere to be found in the resulting report XML file, and only "id" and "name" attributes are included in the report file.

Discussions

To improve data integrity in NEMSIS v3, we will be using business rule validation together with XML validation. Schematron transformation is a powerful mechanism to help business rule validation and will be required by NEMSIS compliance process. It has already been adopted by some states/vendors. In the two examples shown above, we demonstrate the pros and cons of Schematron. The following items are some observations from playing with simple Schematron rules at the NEMSIS TAC. We welcome any input from the NEMSIS software community.

1. Schematron goes beyond XML schema validation. It provides the mechanism to diagnose/check/report business constraints applied to XML element(s). In conjunction with XML schema validation and other tools, it helps improve data integrity.
2. Schematron is straightforward and simple to implement. (On the official website, Schematron authors use word of "trivial" to define implementation – I would not go that far.) It is just a multiple-step XSLT transformation, and most modern computer languages can be adopted. Many developers are familiar with XSLT.
3. Schematron provides several powerful tools, including: diagnostics, keys, includes, extends, phases, abstract patterns, etc.
4. It is possible to embed Schematron rules directly into XSD. The benefit is that we can combine XML XSD validation and business validation into one step (see http://msdn.microsoft.com/en-us/library/aa468554.aspx for an example.)
5. The XPath expression is ***THE*** central part of Schematron. This is a double-edge sword: in one hand, Schematron can take advantage of the full power of XPath. On the other hand, as demonstrated above, during execution, its memory requirement depends on the size of the input XML file. The execution time will vary based on the size of the output. These scalability limitations make it ill-suited for big input XML files and/or lots of errors in the output.
6. It is necessary to further process the raw report from Schematron. The feedback from Schematron should be a document easy to understand/interoperate by users.
7. An interesting observation is that the Schematron report doesn't include the position (the line number and column number) of XML file on which an error is reported. This is because Schematron is a XSLT transformation. Standard XSLT processors do not provide such information. It is possible to identify the "position" of element where error occurs, but it is the position in the parsed DOM tree. In many cases, we cannot easily pinpoint the "position" in the DOM tree and associate it to a line number in an XML file. Some XML processors do provide such capacity. For example, in professional/enterprise editions of Saxon, one can use "saxon:line-number()" extension to report line numbers.

We consider Schematron to be a good candidate for business rule validation, especially when the input XML file is reasonably small. That is why we require its use at the single NEMSIS record level. NEMSIS TAC will release sample codes to further demonstrate the adoption of Schematron in V3.

Schematron will not fix all data integrity issues. Not all business rules can be enforced at a single XML file level: for example, if agency “Elmo” submits its EMS data to State “Sesame Street” every day, about 10 records per day. At the end of the month, the State EMS manager is surprised to find that all 300 records from agency “Elmo” are for female patients! This finding is not impossible, but still raises a big question mark. Another example could be variation in the number of submitted EMS records. It is hard to believe that for one state, the first month records 1,000 EMS events, and in the next month the state records 10,000 EMS events. These kinds of business rule violations could only be caught after we aggregate enough data, to possibly detect the violation at database/warehouse/report level.

Conclusion

1. It is absolutely necessary to validate NEMSIS XML files against the XSD schema. Data processing will stop if the file fails XSD validation.
2. Software solutions are required to execute business rule validation. But NEMSIS TAC will not enforce which technique to use. Schematron is one good option. Depending on different usage scenarios, other techniques could be implemented. However, use of a Schematron-compatible validation file is required.
3. Different vendors/states/agencies could develop their own business rules and implementations. They should design their own reject/accept criteria.
4. The NEMSIS TAC will publish a set of **critical** business rules in clear text, together with Schematron syntax. Any XML file submitted to the NEMSIS TAC that fails one or more critical business rules will be rejected.
5. NEMSIS TAC will publish a set of **optional** business rules in clear text, together with Schematron syntax. Failing one or more of these business rules will result in a warning. The submitted file will be accepted if no critical rules are violated.