Introduction to Schematron

Wendell Piez and Debbie Lapeyre

Mulberry Technologies, Inc.
17 West Jefferson St.
Suite 207
Rockville MD 20850
Phone: 301/315-9631
Fax: 301/315-8285
info@mulberrytech.com
http://www.mulberrytech.com

Version 90-1.0 (November 2008)

Introduction to Schematron

Exhibit

Appendix

Introduction to Schematron

slide 1

Administrivia

- Who are you?
- Who are we?
- Timing

slide 2

Schematron is a ...

- Way to test XML documents
- Rules-based validation language
- Way to specify and test statements about your XML document
  - elements
  - attributes
  - content
- Cool report generator

All of the above!

slide 3

Reasons to use Schematron

- Business/operating rules other constraint languages can't enforce
- Different requirements at different stages of the document lifecycle
- Local or temporary requirements (not in the base schema)
- Unusual (but not illegal) variations to manage
- No DTD or schema (but some need for consistency)
- Need *ad hoc* querying and discovery

slide 4

What Schematron is used for

A few use cases

- QA / Validation
  - run reports for checking by human agents
    (Display all figure `titles` and `captions` for cross-checking)
  - validate things schemas can't express
    (If `owner-type` attribute is "consultant",
    value must be either "Mulberry" or "Menteith",
    otherwise value is unconstrained)
  - find patterns in documents
    (Show me all the `authors` who have no `bio`)
- Check element values against a controlled vocabulary
  (could be maintained externally)
- Validate output of a program against its input (or the reverse)

slide 5

Schematron is an XML vocabulary

- A Schematron "program" is a well-formed XML document
- Elements in the vocabulary are "commands" in the language
- The program is called a "schema" (sadly)

(schema, specification, rule set, program, pattern set,
assertion set, potato, potahto)

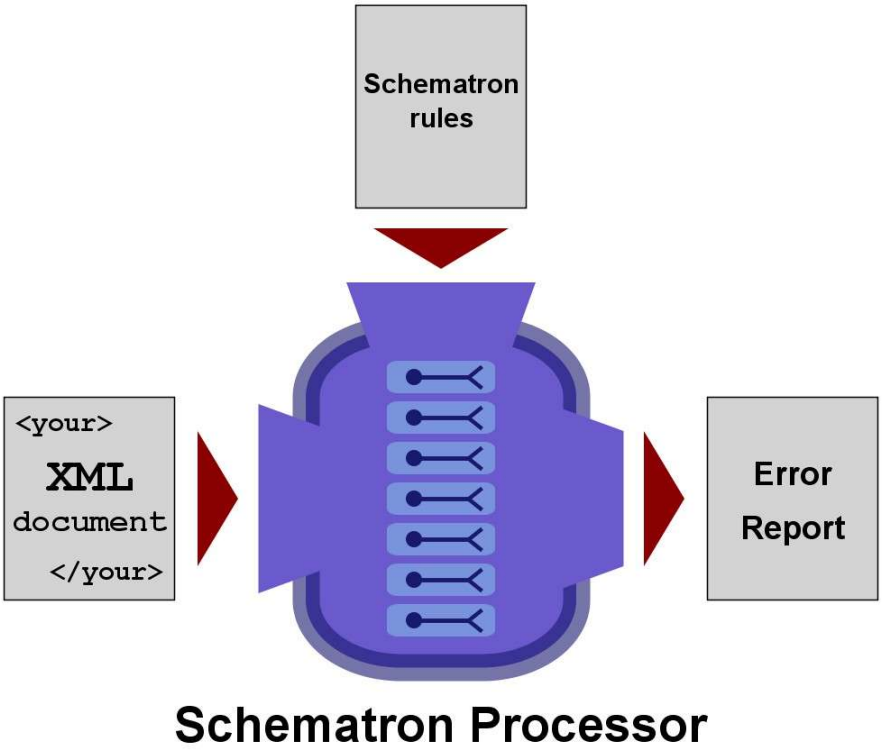
slide 6

Schematron *specifies*, it does not *perform*

- A Schematron "schema" specifies tests to be made on your XML
- A set of *declarations* for a process
  ("test this; tell me that")
- A Schematron processor is necessary to make anything happen
  - reads and interprets your Schematron rules
  - applies the tests to your documents
  - reports back with any messages

Simple Schematron processing architecture

Easily scales up to accommodate more than one XML document, or more than one Schematron



slide 8

Schematron validation in action

(a short demonstration on real data)

- We have XML data borrowed from PubMed Central
  - journal articles
  - multiple source files
- We have a Schematron rule set
- We can show the messages generated

slide 9

Basic Schematron building blocks

- *Assertions* —
  - are to be tested
  - describe conditions you'd like to be told about
- *Messages* — you get them back when tests succeed or fail
- *Rules* — tests are collected into rules, which apply to particular XML elements (context)
- *Patterns* — Rules are grouped into families called patterns
- *Phases* — Activate different families at different times

slide 10

How Schematron works

A rule (a collection of constraints)

- Declare its context (where it applies; usually an element)
- *In that context*, performs a series of tests

(Programmer-speak, simplified version: For every element in the document described as the context of a rule, the rule's tests will be made with that element as context)

```
 1 <?xml version="1.0" encoding="utf-8" ?>
 2 <schema xmlns="http://purl.oclc.org/dsdl/schematron" >
 3 <title>Check Sections 12/07</title>
 4 <pattern id="section-check">
 5   <rule context="section">
 6     <assert test="title">This section has no title</assert>
 7     <assert test="para">This section has no paragraphs</assert>
 8   </rule>
 9 </pattern>
10 </schema>
```

slide 11

Outline of a simple Schematron rule set

```
schema
  title
  pattern+
    rule+
      (assert or report)+
```

`schema`	The document element (contains all others)
`title`	A descriptive human readable title
`pattern`	Set of related rules
`rule`	One or more assertions that apply *in a given context*
`assert`, `report`	Tests: Declare conditions to be tested (in their attributes) and provide messages to be returned (in their content)

slide 12

A simple demonstration XML document

(know your document structure!)

```
<dog>
 <flea/>
 <flea/>
 <bone/>
</dog>
```

slide 13

A simple Schematron rule set

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron">

 <title>Dog testing 1</title>

 <pattern id="obedience-school">
   <rule context="dog">
      <assert test="bone">Give that dog a bone!</assert>
      <report test="flea">Your dog has fleas!</report>
  </rule>
 </pattern>

</schema>
```

We thank Roger Costello for the “dogs and fleas” example (which we will elaborate)

slide 14

This Schematron translated into English

- There is a pattern with one rule
- The rule contains two tests
  (We can have as many as we need)
- The rule applies to `dog` elements (that's the *context*)
- The rule for dogs is that each dog:
  - Must have at least one bone
    (In the context of a `dog`,
    a `bone` element must be present or
    the assertion fails and you get a message.)
  - May have a flea, but if so we want to know
    (In the context of a `dog`,
    if any `flea` elements are present,
    a report will be given

slide 15

When does a rule fire?

- Context determines when to try the tests
- `context` attribute on `<rule>` sets the context

`<rule context="dog">...</rule>`	For any `dog` element, do these tests
`<rule context="section">...</rule>`	For any `section` element, do these tests
`<rule context="html:body">...</rule>`	For any `html:body` element, do these tests

slide 16

Assertions and reports are tests

Tests are expressed in two forms:

- `<assert>`: *a statement about an expectation*
  - "a section must have a title"
    tell me if you don't find one

```
<rule context="section"
   <assert test="title">Section has no title.</assert>
</rule>
```

- `<report>`: *a circumstance of interest*
  - "notes might turn up inside notes (but that's bizarre)"
    tell me if you see one

```
<rule context="note">
  <report test="ancestor::note">A note appears
  inside a note</report>
</rule>
```

slide 17

`assert` and `report`

- In the Schematron specification are called (confusingly) *assertions*
  But they work oppositely
  - `<assert>` means tell me if it is *not* true
  - `<report>` means tell me if it *is* true

Mnemonic:
`report` means "ho hum, show me where this is true";
`assert` means "it better be true, or else!"

slide 18

Context and tests are stated in attributes

- A rules's `context` attribute **sets the context**
- The `test` attribute of an `assert` or `report` **expresses the test**

```
<rule context="dog">
  <assert test="bone">This dog has no bone.</assert>
</rule>

<rule context="note">
  <report test="ancestor::note">A note appears
   inside a note</report>
</rule>
```

slide 19

Just Enough XPath

XPath is the *query syntax* used for

- The `context` for `rules`*
  a *context* identifies a class of nodes (elements)
- the `tests` (for `assert` and `report`)*
- For example

```
<rule context="child::note">
   <report test="ancestor::note">A note appears inside a
    note</report>
</rule>
```

(In XPath, `child::note` is the same as plain `note`)

(*This could be done in another query language, but XPath is usual.)

slide 20

What is XPath?

- A language for addressing parts of an XML document
- A W3C Recommendation in 1999 (`http://www.w3.org/TR/xpath`)
- Named because it uses a path notation with slashes
  like UNIX directories and URLs

  ```
  invoice/customer/address/zipcode
  ```
- A lightweight query language
  ("Addressing" really means "querying")
  XPath expressions return data objects and values from XML documents
- Used by XQuery, XSLT, and XPointer (among others)
- Widely implemented in many languages
  (Perl, Python, Java, Javascript....)

slide 21

Faking it in XPath

- XPath to say *where* to test (*context*)
  - `<rule context="dog">...` applies to all elements named `dog`
  - `<rule context="chapter">...` applies to all the `chapter elements`
- XPath to say *what* to test
  - `test="bone"` is `true` if
    there is at least one `bone` element inside the given context
    (and `false` if no `bones`)
  - `test="flea"` is `true` if
    there is at least one `flea` element inside the context
    (and `false` if not)

slide 22

A (slightly) more complex example

The XML document:

```
<dog bark="11" bite="10">
  <bone/>
</dog>
```

(Test with `obedience-school/dogtest-2.sch`)

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron">

 <title>Dog testing 2</title>

 <pattern id="obedience-school">
    <rule context="dog">
      <report test="@bark > @bite">
        This dog's bark is worse than his bite.
      </report>
    </rule>
 </pattern>

</schema>
```

- This time, the test compares the `bark` and `bite` *attributes*
- The > test compares two values, and is True if:
  - both are present
  - both are numbers
  - the first is greater than the second
- We get our message when the test is True (this is a `report`)

slide 23

Playing with dogs and bones

(a short demonstration)

Let's see how this works, with dogs and bones and fleas.

slide 24

Schematron context expressed in XPath

Value of `context` attribute on `<rule>` is an XPath expression

Rule with XPath Context	What the Context Means
`<rule context="figure">`	For any `figure` element
`<rule context="section/figure">`	For any `figure` whose parent is a `section`
`<rule context="section/figure/title">`	For any `title` whose parent is a `figure` whose parent is a `section`
`<rule context="/">`	For the document (the root node)
`<rule context="name [@title='Mr.']">`	For any `name` element with a `title` attribute with a value of "Mr."

slide 25

Schematron tests expressed in XPath

Value of `test` attribute on `assert` and `report` is an XPath expression

`<assert test="title">`	The context node has a `title` child (short form) (I.e., there must be a `title` child)
`<assert test="child::title">`	The context node has a `title` child (long form)
`<assert test="@float ='0'">`	The `float` attribute on the context node has a value of zero
`<rule test="count(*) > 20">`	There are more than 20 element children inside the context node ("*" means the all the element children)
`<assert test="string-length(.) > 25">`	The length of the context node is greater than 25 characters ("`.`" means "`self`")
`<report test="parent::section">`	The parent of the context node is a `section`
`<report test="parent::footnote and string-length(.) > 300">`	The parent of the context node is a `footnote` *and* the length of the node's content is longer than 300 characters

slide 26

Evaluating XPath expressions

```
<rule context="para">
  <assert test="string-length(.) &gt; 20">
    Paragraph seems awfully short.
  </assert>
</rule>
```

To evaluate an XPath test, we need to know at least two things:

1. The XPath expression itself

   ```
   string-length(.) &gt; 20
   ```

2. The *evaluation context*: a node in the document
   - Set in Schematron by the `context` attribute
   - Here, the *context* is a `para` element
   - Any `para` element in the document will match this context, so the test will be performed for each

`string-length(.) > 20` will be answered
`true` or `false` for every paragraph

slide 27

Organizing the Schematron schema

- `schema` is the document element
- Elements allowed at the top level inside `schema` include:
  - `title` - A title for the rule set. Optional, but a good idea.
  - `pattern` - A set of related rules: where the action is
  - `p` - A paragraph of human-readable documentation. (Never a bad idea!)
  - `ns` - Namespace declaration for your XML data so you can test it
  - `let` - A variable declaration (covered later)

slide 28

Other top-level elements

There are other top level elements inside `schema` that we won't cover today

- `include` - For calling in external Schematron modules (simple, not very common)
- `phase` - A collection of patterns to be run together (useful for some applications)
- `diagnostics` - Extended diagnostic information for rules to refer to (optional in Schematron; not all processors support this one)

slide 29

How Schematron performs its tests

- Any element in the document may be tested in each pattern
- In a pattern
  - the first rule with a `@context` matching the element is applied
  - subsequent rules in the same pattern that match the element are ignored
- In the rule, all assertions are tested with the matched element as context

```
<pattern>
  <rule context="p">
    <assert test="string-length(.) &gt; 20">
     paragraph is awfully short
    </assert>
  </rule>

  <rule context="title">
    <report test="string-length(.) &lt; 5">
      title is really short
    </report>
  </rule>
</pattern>
```

slide 30

Relations between patterns and rules

- `pattern` elements contain one or more `rule` elements
- `rule` elements group the tests for a single context
- Rules in a pattern are generally related in some way
  - functional
  - typically what messages do you want to see at the same time

If you need to look at the same context twice, you need another pattern

slide 31

Make your tests work together

- Remember, the first `rule` that matches in a `pattern` fires, no other matching `rule` does
- Don't just add rules without thinking
- Here, the second `rule` never fires:

```
<pattern>
   <rule context="sec">
     <assert test="title">Section does not have a
      title</assert>
   </rule>

   <rule context="sec[not(subsec)]">
     <assert test="p">Section without subsections has
     no  paragraphs</assert>
   <rule>
</pattern>
```

- Fix this as:

```
<pattern>
   <rule context="sec">
     <assert test="title">Section does not have a
       title</assert>
     <assert test="subsec|p">Section has neither
      paragraphs nor subsections</assert>
   <rule>
</pattern>
```

slide 32

Summarizing `assert`

Making sure things are as expected

- For validation, `assert` is bread and butter
  - You are looking for errors
  - You assert that something *is true*
  - If it is true, fine
  - If it is not, you get your message
- The message is the content of the `assert` element
- For reports or queries, you may use `report` more often
  `assert` is useful to know when things are *not* true

```
<rule context="dog">
  <assert test="bone">This dog has no bone.</assert>
</rule>
```

slide 33

A realistic example of `assert`

Context is `xref[@ref-type='fig-ref']`
(cross-reference of type `fig-ref`)

```
<assert test="@rid = //fig/@id">
    xref of type 'fig-ref' does not point to a
   figure in the document
</assert>
```

- Returns `true` if any `fig` element is found
  whose `@id` is the same as the `xref/@rid`
- The test is true if `@rid` equals the value of any
  `//fig/@id`
  (by existential quantification)

slide 34

A fancier realistic example

Context is `caption` (of a figure, table, etc.)

```
<assert
  test="*[1][self::p[(*|text()[normalize-space()])[1][self::b]]">
  Caption does not begin with bold inside p
</assert>
```

In other words: "The first element inside a `caption` must be a paragraph (`p`) which starts with bold (`b`)"

- Passes this:

```
<caption>
  <p><b>Two men and a balloon.</b>
   The Montgolfier brothers as depicted in 1785.</p>
</caption>
```

- Fails this:

```
<caption>
   <p>Two men and a balloon. <b>The Montgolfier brothers
   as depicted in 1785.</b></p>
</caption>
```

(This test could have been simpler if it didn't allow for comments, PIs, or whitespace-only text)

slide 35

Hints for writing assertions

- If your test is getting *really* complex
  - use more than one assertion
  - move some of the logic into the `context`
- If you have a context clash, you need more than one pattern

For example

```
<rule context="citation">
  <report test="article-title and not(source)">
    citation has an article-title, but no source
  </report>
</rule>
```

can be rewritten as

```
<rule context="citation[article-title]">
  <assert test="source">
    citation with an article-title has no source
  </assert>
</rule>
```

slide 36

Summarizing `report`

Finding things of interest

- Schematron `reports` are not just for "errors"
- With `report` you can
  - locate elements of interest
  - describe the features of a document
  - find conditions in the document
    - things you expect, but want to know about
    - things you don't expect, but expect enough to ask about
- If the test is true, you get your message
- The message is the content of the `<report>` element

```
<rule context="dog">
   <report test="flea">
     This dog has fleas!
   </report>
</rule>
```

slide 37

Some examples of `report`

```
<rule context="sec">
   <report test="count(//sec[title=current()/title]) > 1">
     Section has the same title as another section
   </report>
</rule>
```

```
<rule context="back">
   <report test="5 > count(.//citation)">
     Fewer than five citation elements appear in the
     back matter
   </report>
</rule>
```

```
<rule context="lpage[../fpage]">
   <report test=". &lt; ../fpage">
     Last page given is lower than first page
   </report>
</rule>
```

slide 38

When being right is enough

- Sometimes the fact that a rule matches is useful in itself
- For example, maybe you want to find all your titles so you can look at them:

```
<rule context="title">
   <report test="true()">title found</report>
</rule>
```

- (We also sometimes write `test="."` — how does this work?)
- This is especially useful when running Schematron in interactive batch mode over a set of documents
- Development technique: locate elements like this when you need to inspect them to write your actual validation rules

slide 39

Schematron for Real

(a short demonstration)

Watch it go!

slide 40

Making better error messages (Advanced)

- `value-of` puts values into messages
- `name` puts elements names into messages

slide 41

`value-of` puts values into messages

- Provide generated text in your messages
- Empty element `value-of` is replaced with its value in your message
- Value is determined by `@select`
- Value of `@select` is an XPath expression that depends on the context of the message

For example:

```
<assert test="@ref-type='fig' or
   @ref-type='sec' or @ref-type='fn'>
   Unknown 'ref-type' value ('<value-of select="@ref-type"/>');
   Value must be 'fig', 'sec', or 'fn'
</assert>

<report test="string-length(.) > 20">
   The element is too long (only 20 characters are allowed);
   <value-of select="string-length(.)"/> characters present)
</report>
```

slide 42

`name` puts elements names into messages

- `<name/>` is short for `<value-of select="name()"/>`
- Let's you put the name of *the matched element* into your message
- For example, to tell sections from appendices when the context is either:

```
<rule context="sec | app">
   <assert test="title"><name/> must have a title</assert>
   <report test="count(title) > 1">
      <name/> has too many titles:
      <value-of select="count(title)"/> are present
   </report>
 </rule>
```

slide 43

Wrapup: Taking advantage of Schematron

(if time permits)

- Download a Schematron processor or
  use an editor/environment that runs Schematron
- Write out your assertions in your native language
- Turn your assertions into XPath statements
  (yes, this is the hard part)
- Write the message you would like to get back

slide 44

Remember what Schematron can do

- Validate/report on document structure
  - presence/absence of elements
  - location of elements
- Validate/report on document content
  - there must be some content
  - there must be some particular content
  - content must follow some rule
- Validate/report on attributes
  - presence/absence of attributes
  - content of attributes
- Check co-occurrence constraints
  - if X is true, then Y should be true
  - A, B, C, and W must all be present (somewhere)

slide 45

Schematron gives you the world's best error messages

You write them!

- You specify where to test your documents
- You specify the test
- The test can be positive or negative
  - you want to know *if*
  - you want to know *unless*
- *You write the message you want to come back*
- Message can be
  - as specific or general as you need
  - in the language of your users
  - in the jargon of your audience

slide 46

Schematron allows "soft validation"

- Schematron allows a stretchy definition of an "error"
- Can find actual *and potential* problem areas
  - even if they are valid
  - even if they won't stop the system from working correctly
  - even if the rules are blurry
    - maybe we *can* fix it, maybe not
    - maybe we *should* or shouldn't
    - maybe we'll need to consider it (or get authority)

slide 47

Why Schematron?

- Handy: Really useful, fast-to-write constraint language
- Lightweight: Lets you test as much or as little as you like
- Flexible: Tests constraints other constraint languages can, and some they cannot

"Schematron is a feather duster that reaches areas other schema languages cannot." — Rick Jelliffe

slide 48

Colophon

- Slides and handouts created from a single XML source
- Slides projected in HTML (created from XML using XSLT)
- Handouts distributed in PDF
  - source XML transformed to Open Office XML with XSLT
  - pagination and tables hand-adjusted
  - Open Office made PDF
- Slideshow tools available at `http://www.mulberrytech.com/slideshow`

slide 49

Reference Slides (Homework)

Advanced concepts for answering questions and at-home reference

- Namespaces in Schematron
- Naming and reusing XPath expressions (variables)
- Schematron using XPath 2.0
- Abstract Rules (declaring and reusing Schematron rules)

slide 50

Namespaces in Schematron (Reference)

- Document element of the Schematron schema is `schema`
- <schema> *requires* a namespace declaration

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
 ...
</schema>
```

- ISO (DSDL) Schematron is designated by `http://purl.oclc.org/dsdl/schematron`
- Other versions have other namespace bindings (The `URI` for Version 1.6 is `http://www.ascc.net/xml/schematron`)

slide 51

Using a prefix on Schematron instructions

- In this class, the ISO namespace has been bound to the default (prefixless) namespace

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
```

- If you prefer, a name prefix may be used ("`sch`" is traditional)

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron">
 ...
</sch:schema>
```

  - Works exactly the same way
  - All Schematron elements use the prefix (`<sch:rule>`, `<sch:assert>`, `<sch:report>`, etc.)
  - This is most helpful when mixing Schematron with other vocabularies(such as other schema languages, or XSLT)

slide 52

Namespaces in your XML documents

- When you have namespaces in your XML documents
  - for element and attribute names
  - that you need to use in `@context` and `@test` expressions
- Use the `ns` element to enable Schematron to see these namespaces (this declares the namespace to the processor)
- Each `<ns>` specifies a prefix/URI pair
- For example

```
<ns prefix="xlink" uri="http://www.w3.org/1999/xlink"/>
```

  - allows recognition of elements and attributes in the XLink namespace using the `xlink` prefix
  - lets you write code like this to test for an `xlink:href` attribute

```
<assert test="@xlink:href">Make this into a live link</assert>
```

(By convention, `<ns>` elements are placed just after the `<title>` inside the schema.)

slide 53

Using Variables (Advanced)

- Let you use the same XPath expressions
  - more than once
  - and give them a name
- Can make your Schematron logic
  - easier to read and understand
  - more efficient to process
  - more general and reusable (as opposed to element-or-attribute-name-specific) with abstract patterns
- Are used in XPath expressions

For example, once *$attribute* has been established:

```
<report test="not($attribute)">
   Expected attribute is missing
</report>
```

slide 54

`let`: Declaring variables

- `<let>` declares a variable
  - gives it a name using `@name` (required)
    - a good name describes the variable
    - variable names follow the rules of XML names
  - gives it a value using `@value` (required)
    - the value is an XPath expression
- Once defined, a variable reference looks like `$name`
  - *name* is the variable name
  - the `$` says that it is a variable

```
<let name="figures" value="//fig"/>

<rule context="xref[@ref-type='fig-ref']">
   <assert test="@rid = $figures/@id">
     xref of type 'fig-ref' does not point to a figure
   </assert>
</rule>
```

slide 55

Scope of variable bindings

- A variable created with `let` has *scope*
- Scope means the context where it can be used
- Scope is set by where the `let` declaration appears
- If `let` is inside a `rule`
  - variable is calculated with that `context`
  - scope is only inside the `rule`
- Any other `let` location creates a global variable (context = document root)

slide 56

Example of scope

```
<rule context="sec">
   <let name="section-bibrefs" select=".//xref[@ref-type='bibl']"/>

   <assert test="count($section-bibrefs) >= 5">
     section contains fewer than 5 bibliographic references
   </assert>

   <report test="$section-bibrefs[not(@rid = //ref/@id)]">
     section contains one or more broken bibliographic references
   </report>
</rule>
```

The variable `$section-bibls` is scoped to the rule matching `sec`

- variable is recalculated for each `sec` that matches the rule (with the `sec` as context node)
- variables can be used anywhere inside this rule, but not outside

slide 57

Tips when using variables

Schematron `let` may appear

- At the top level (reuse inside any rule)
- Inside a `pattern` (reuse inside any rule)
- Inside a `rule` (for use only in its assertions)

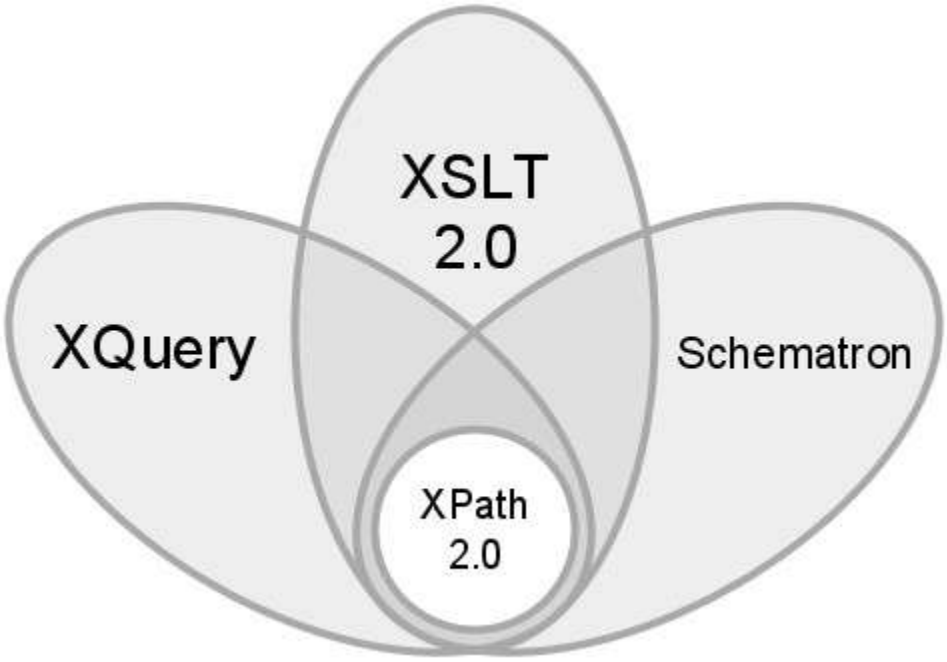
A variable may be useful any time

- You use the same XPath expression several times
- You collect many nodes, especially repeatedly

slide 58

XPath 2.0 in Schematron

XQuery, XSLT, and Schematron can all use XPath 2.0



slide 59

From XPath 1.0 to XPath 2.0

XPath 2.0 is considerably more complex than XPath 1.0

- Closer to "complete" query language
- More functions, operators, functionality
- Real datatypes (XSD)
- Regular expressions on strings

and much much more!

slide 60

Schematron using XPath 2.0

The Exhibit below shows some more complex Schematron, using the power that XPath 2.0 can provide.

Exhibit 1

Schematron using XPath 2.0

(Note new namespace `URI` and `@queryBinding`)

```
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
 queryBinding="xslt2" >

<title>Sample Schematron using XPath 2.0</title>

<ns prefix="xlink" uri="http://www.w3.org/1999/xlink"/>
<ns prefix="xs" uri="http://www.w3.org/2001/XMLSchema"/>

<!-- Uses XPath 2.0 and XML Schema datatypes to do
 some fancy checking -->

<pattern id="uri-testing">
   <rule context="*[@xlink:href]">
     <assert test="normalize-space(@xlink:href)">
      @xlink:href is given with no value
     </assert>
     <assert test="@xlink:href castable as xs:anyURI">
      Broken URI in @xlink:href
     </assert>
   </rule>
</pattern>

<pattern id="preformat-testing">
   <rule context="preformat">
     <assert
      test="every $line in tokenize(.,'&#xA;')
      satisfies string-length($line) le 72">
      preformat is wider than 72 characters
     </assert>
   </rule>
</pattern>

</schema>
```

("
" is a linefeed character)

slide 61

Abstract rules and patterns (Advanced)

- For reusing Schematron logic (not just XPath logic)
- Write a rule or an entire pattern as an *abstract* rule or pattern
- `@abstract="true"` identifies it as an abstract pattern
- context and tests will be *parameterized*
  (not all explicit element or attribute names)
- an `@id` identifies the pattern
- Invoking patterns
  - use an `@is-a` to point to the named `pattern` (it's `@id`)
  - are therefore an instance of the abstract pattern
  - set up the parameters for the abstract pattern and then invoke it

slide 62

An abstract pattern

(invoking patterns in bold)

```
<pattern id="non-ws-attribute" abstract="true">
 <rule context="$my-element[$my-attribute]">
 <assert test="normalize-space($my-attribute)">
 <name/> has no <value-of select="name($my-attribute)"/>
given</assert>
 </rule>

 <rule context="$my-element">
 <report test="not($my-attribute)">
 <name/> is missing an expected attribute</report>
 </rule>
</pattern>

<pattern id="xref-rid" is-a="non-ws-attribute">
 <param name="my-element" value="xref"/>
 <param name="my-attribute" value="@rid"/>
</pattern>

<pattern id="ext-link-href" is-a="non-ws-attribute">
 <param name="my-element" value="ext-link"/>
 <param name="my-attribute" value="@xlink:href"/>
</pattern>
```

Appendix 1

Resources

Schematron Resources

- The source for all things Schematron — **http://www.schematron.com/resources.html**
- The ISO Schematron Specification — **http://www.schematron.com/spec.html**
- Tools — **http://www.eccnet.com/schematron/index.php/Tools-url**
- Eric van der Vlist has written an O'Reilly Shortcut on Schematron **http://www.oreilly.com/catalog/9780596527716/cover.html**
- Tutorials (some of these are very good but not ISO Schematron)
  - Roger Costello (both) — **http://www.xfront.com/schematron/**
  - Uche Ogbuji (non-ISO) — **http://www.ibm.com/developerworks/edu/x-dw-xschematron-i.html**
  - Zvon — (beginner non-ISO) **http://www.zvon.org/xxl/SchematronTutorial/General/contents.html**
  - Dave Pawson — (advanced ISO) **http://www.dpawson.co.uk/schematron/index.html**

XPath Resources

- XPath 1.0 Recommendation **http://www.w3.org/TR/xpath**
- XPath 2.0 Recommendation **http://www.w3.org/TR/xpath20/**
- XPath tutorials (find one that works for you)
  - (XPath 1.0) **http://www.zvon.org/xxl/XPathTutorial/General/examples.html**
  - (XPath2.0) **http://www.w3schools.com/xpath/**
  - (XPath 1.0) **http://www.ibm.com/developerworks/edu/x-dw-xxpath-i.html**
  - (XPath 1.0) **http://www.tizag.com/xmlTutorial/xpathtutorial.php**
- XPath Books
  - *XSLT 2.0 and XPath 2.0: Programmer's Reference*, by Michael Kay (4th Edition, Wiley, 2008; 0-470-19274-0)
  - *XPath Essentials*, by Andrew Watt (John Wiley & Sons, 2002; ISBN: 0471205486)
  - *XPath: Navigating XML with XPath 1.0 and 2.0*, by Steven Holzner (SAMS Kick Start, 2004; 0-672-32411-3)
  - *XPath and XPointer*, by John E. Simpson (O'Reilly 2002; ISBN: 0-596-00291-2)