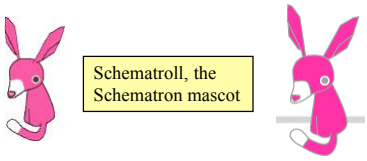
Schematron



Schematroll, the Schematron mascot

"Sounds like a particle accelerator for XML Schemas" – Marlon Burney

Outline

1. Overview and Background
2. Basic Features
3. Advanced Features
4. New Features (not well supported yet)
5. Implementations

1

Overview

- Rule-based rather than grammar-based
  - DTD, XML Schema and RELAX NG are all grammar-based
  - grammar-based approaches take a closed approach
    - everything not explicitly allowed is treated as invalid
  - rule-based approaches take an open approach
    - everything not explicitly disallowed is treated as valid
- Not typically the only validation method used
  - use one grammar-based method for structure and value constraints
  - use Schematron for constraints that can't be described in grammar-based methods
    - such as constraints between multiple elements/attributes and validation across documents (using `document` function)
- Designed by Rick Jelliffe
- Main web site: http://www.schematron.com

1

Overview (Cont'd)

- Syntax is XML
  - described by
    - RELAX NG compact schema (Annex A)
    - Schematron schema (Annex B) for constraints that RELAX NG can't describe
- Rules use XPath expressions
  - can validate **anything that can be expressed as a boolean XPath expression**

see **XPath overview** on next page

- Rule location
  - can be in a separate file, typically with a "`.sch`" extension
  - can be embedded within other schema files, such as RELAX NG
- Implementations
  - designed so that XSLT-based implementations are easy to create
  - can also been implemented without using XSLT for better performance
  - see list of implementations later

1

XPath Overview

- XPath is to XML what regular expressions are to strings
  - result is a node set or a value (boolean, string or number)
  - composed of "steps" separated by slashes
  - steps navigate through XML hierarchy
- Step syntax
  - `axis::node-test[predicate-1]...[predicate-n]`
  - `axis` can be one of
    - `child`, `descendant`, `parent`, `ancestor`, `attribute`, `namespace`, `following-sibling`, `preceding-sibling`, `following`, `preceding`, `self`, `descendant-or-self`, `ancestor-or-self`
    - defaults to `child` when `axis::` is omitted
  - `node-test` can be one of
    - an element name, `*` (for any element), `node()` (for any node), `text()`, `comment()`, `processing-instruction('pi-name')`
  - **predicates** are optional and each further reduces the result set

an XPath expression that begins with "`/`" starts at "document root"; makes it absolute instead of relative to context node

many examples will be shown later

1

XPath Overview (Cont'd)

- Supports a set of **operators**
  - **arithmetic:** `+`, `-`, `*`, `div`, `mod`
  - **relational:** =, !=, <, >, <= and >=
  - **boolean:** `and`, `or`, `not()`
  - **node-set union:** `|`
- Supports a set of **functions**
  - **string functions** include:
    `concat``(value1, value2, ...)`
    `contains``(value, substring)`
    `format-number``(value, format)`
    `starts-with``(value, substring)`
    `string-length``([value])`
    `substring``(value, start[, length])`
    `substring-after``(value, substring)`
    `substring-before``(value, substring)`
  - **math functions** include:
    `count``(node-set)`
    `sum``(node-set)`

many examples of XPath expressions will be shown later

1

XPath Examples

- Use XML Spy to practice writing XPath expressions
  - start XML Spy
  - open `labs/Schematron/MusicCollection/music-collection.xml`
  - on the XML menu, select "Evaluate XPath…"
  - Enter the following XPath expressions
    - To see the root element, **`/*`**
    - To see the direct children of the root element, **`/music-collection/*`**
    - To see the name of every artist, **`/music-collection/artist/name`**
    - To see the name of all artists whose name begins with "C", **`/music-collection/artist[starts-with(name, 'C')]/name`**
    - To see the year of every CD, **`//cd/@year`**
    - To see the title of all CDs from 1993, **`//cd[@year=1993]/title`**
    - To verify the number of CDs from 1993, **`count(//cd[@year=1993]) = 5`**

1

Document Schema Definition Languages (DSDL)

- DSDL is defined by ISO/IEC 19757
  - "The main objective of DSDL is to **bring together different validation-related tasks and expressions to form a single extensible framework** that allows technologies to work in series or in parallel to produce a single or a set of validation results."
    - don't have to use one schema language to perform all the validation on a given document
  - see http://dsdl.org/
- Schematron is undergoing standarization as one part of this
  - "Rule-based validation – Schematron" – **part 3**
    - see http://dsdl.org/0524.pdf

1

Document Schema Definition Language (Cont'd)

- Other parts include
  - "Regular-grammar-based validation – **RELAX NG**" – **part 2**
  - "Namespace-based Validation Dispatching Language – **NVDL**" – **part 4**
    - allows validation to be dispatched to a different schema for each namespace used in a document
  - "**Data types**" – **part 5**
    - such as **XML Schema** data types
  - "Path-based Integrity Constraints" – **part 6**
    - such as **XPath**

2

Main Concepts

- Assertions
  - conditions to be tested such as existence and values of elements/attributes
- Rules
  - groups of assertions (`assert` and `report` elements)
  - selects the set of context nodes under which they are evaluated
- Patterns
  - groups of rules with an id (used by phases)
  - each node being tested will only be used as the context node of a single rule within the pattern (more on page 18)
- Phases
  - named groups of patterns (specified by their id) that allow evaluating only the rules in those patterns

2

Schema

- All elements of a Schematron schema are wrapped in a `schema` root element
- Example

in latest spec. the Schematron namespace is **`http://purl.oclc.org/dsdl/schematron`**, but the ref. impl. and Jing still require this previous namespace

```
<?xml version="1.0"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>schema title goes here</title>
  <ns prefix="prefix" uri="namespace-uri"/>

  ... phases go here ...

  ... patterns go here ...

  ... diagnostics go here ...

</schema>
```

optional; not used by ref. impl. or Jing

can have any number of these;
the prefixes are used in
`rule` element `context` attributes and
`assert`/`report` element `test` attributes

patterns contain rules;
rules contain assert and report elements

2

Validation Steps

(not necessarily in this order)

For each **context node** in the document being validated
  For each **`phase`** being evaluated
    For each **`pattern`** in the phase

if phases aren't used then all patterns are evaluated

      Find the first **`rule`** that matches the context node
      For each **`assert`** and **`report`** in the rule
        Perform the **`test`**
        If an **`assert test`** fails or a **`report test`** passes
          Output the message inside it
          If diagnostics are enabled
          and there are associated diagnostics
            Output the **`diagnostic`** messages

diagnostics provide information beyond messages in `assert` and `report` elements such as actual/expected values and hints to repair the document

2

Order

- Only the order of `rule` elements is significant
  - for each context node,
    only the first matching `rule` within a pattern is used
- The order of other things is implementation dependent
  - order in which context nodes are validated
  - order in which `phase`s are evaluated
  - order in which `pattern`s within a phase are evaluated
  - order in which `assert`s and `report`s within a `rule` are tested

2

Assertions

- Conditions to be tested such as
  - existence of elements/attributes
  - values of elements/attributes
- Positive assertions
  - specified with `<assert test="boolean-xpath">message</assert>`
  - message is output if test evaluates to false
- Negative assertions
  - specified with `<report test="boolean-xpath"/>message</report>`
  - message is output if test evaluates to true

2

Assertions (Cont'd)

- `name` element
  - used in messages to output name of context node being tested
  - doesn't output namespace prefixes or URIs
  - optional `path` attribute is used to output the name of a node found relative to the context node
    - for example, `<name path="*[1]"/>` outputs name of first child element of context node
  - see example on page 20
- `value-of` element
  - used in messages to output the value of other nodes found relative to the context node being tested
    - allowed in `assert`, `report` and `diagnostic` elements
      - some implementations, such as Jing 20030619, only support `value-of` in `diagnostic` elements
  - see example on page 27

In both the ref. impl. and Jing, `value-of` seems to only be able to get values of attributes!

2

Rules

- Groups of assertions
- Selects set of context nodes under which they are evaluated
- Syntax

```
<rule context="xpath">
  ... assertions go here ...
</rule>
```

- Example

```
<rule context="candidates">
  <assert test="sum(candidate/@percentage) = 100">
    The sum of the percentage attributes for each candidate
    must be 100.
  </assert>
  <report test="count(candidate) < 2">
    There must be at least two candidate elements inside candidates.
  </report>
</rule>
```

XML document to be validated

```
<root>
  ...
  <candidates>
    <candidate name="John Doe" percentage="51"/>
    <candidate name="Joe Blogs" percentage="49"/>
  </candidates>
  ...
</root>
```

2

Rules (Cont'd)

- One rule per pattern is evaluated
  - for each node being tested,
    the first rule within a pattern that matches it
    is the only one within that pattern
    that will be evaluated

2

Patterns

- Groups of rules
  - ordered so the first matching rule is the one that should be used
- Can choose to test only the rules within specified patterns
  - using "phases" (described next)
- Syntax

used to refer to the pattern from a phase

```
<pattern name="pattern-name" [id="pattern-id"]>
  ... rules go here ...
</pattern>
```

- Example

```
<pattern name="artist rules" id="artists">
  <rule context="artist[@vocals='female']">
    ... assertions specific to artists with female vocals go here
   ...
  </rule>
  <rule context="artist">
    ... assertions for all other artists go here ...
  </rule>
```

2

Music Collection XML

```
<?xml version="1.0" encoding="UTF-8"?>
<music-collection xmlns="http://www.ociweb.com/music">
  <owner>Mark Volkmann</owner>
  <artist type="solo" vocals="female">
    <name>Yamagata, Rachel</name>
    <cd category="pop" year="2004">
      <title>Happenstance</title>
    </cd>
  </artist>
  <artist type="group" vocals="male">
    <name>Cake</name>
    <cd category="pop" year="1996">
      <title>Fashion Nugget</title>
    </cd>
    <cd category="pop" year="1998">
      <title>Prolonging The Magic</title>
    </cd>
  </artist>
</music-collection>
```


2

Music Collection Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.ascc.net/xml/schematron">
  <title>Music Collection Schema</title>
  <ns prefix="m" uri="http://www.ociweb.com/music"/>

  <pattern name="all">

    <rule context="/">
      <assert test="m:music-collection">
        Root element must be music-collection.
      </assert>
    </rule>

    <rule context="m:music-collection">
      <assert test="count(m:owner) = 1">
        The element <name/> must have one owner child element.
      </assert>
      <assert test="count(*) = count(m:owner|m:artist)">
        The only valid child elements of <name/> are owner and artist.
      </assert>
    </rule>
```

phases aren't used

the only pattern

This example demonstrates using Schematron to validate everything about an XML document except child element order.

Typically a grammar-based schema language would be used to validate structure/value constraints.

In that case, Schematron would only be used to validate things that can't be described in a grammar-based language, such as constraints between multiple elements/attributes.

2

Music Collection Schema (Cont'd)

```
<rule context="m:artist">
  <assert test="parent::m:music-collection">
    The parent of <name/> elements must be music-collection.
  </assert>
  <assert test="count(*) = count(m:name|m:cd)">
    The only valid child elements of <name/> are name and cd.
  </assert>
  <assert test="count(m:name) = 1">
    The element <name/> must have one name child element.
  </assert>
  <assert test="@type">
    The element <name/> requires a type attribute.
  </assert>
  <assert test="@vocals">
    The element <name/> requires a type attribute.
  </assert>
  <assert test="count(@*) = count(@type|@vocals)">
    The only valid attributes of <name/> are type and vocals.
  </assert>
```


2

Music Collection Schema (Cont'd)

```
  <assert test="@type='solo' or @type='group'">
    The type attribute of the <name/> element
    must have a value of "solo" or "group".
  </assert>
  <assert test="@vocals='female' or @vocals='male' or
                @vocals='mixed' or @vocals='none'">
    The vocals attribute of the <name/> element
    must have a value of "female", "male", "mixed" or "none".
  </assert>
</rule>
```

2

Music Collection Schema (Cont'd)

```
<rule context="m:cd">
  <assert test="parent::m:artist">
    The parent of <name/> elements must be artist.
  </assert>
  <assert test="count(m:title) = 1">
    The element <name/> must have one title child element.
  </assert>
  <assert test="count(*) = count(m:title)">
    The only valid child element of <name/> is title.
  </assert>
  <assert test="@category">
    The element <name/> requires a category attribute.
  </assert>
  <assert test="count(@*) = count(@category|@import|@year)">
    The only valid attributes of <name/> are category, import and year.
  </assert>
```


2

Music Collection Schema (Cont'd)

```
  <assert test="@category='alternative' or
                @category='classical' or
                @category='country' or
                @category='folk' or
                @category='jazz' or
                @category='pop' or
                @category='rock' or
                @category='other'">
    The category attribute of the <name/> element
    must have a value of "alternative", "classical",
    "country", "folk", "jazz", "pop", "rock" or "other".
  </assert>
  <assert test="not(@import) or @import='true' or @import='false'">
    The import attribute of the <name/> element
    must have a value of "true" or "false".
  </assert>
  <report test="@year &lt; 1990 or @year > 2010">
    The year attribute of a <name/> element
    must be between 1990 and 2010.
  </report>
</rule>
```

2

Music Collection Schema (Cont'd)

```
<rule context="m:name">
  <assert test="parent::m:artist">
    The parent of <name/> elements must be artist.
  </assert>
  <assert test="count(*) = 0">
    The element <name/> doesn't contain child elements.
  </assert>
</rule>

<rule context="m:owner">
  <assert test="parent::m:music-collection">
    The parent of <name/> elements must be music-collection.
  </assert>
  <assert test="count(*) = 0">
    The element <name/> cannot contain child elements.
  </assert>
  <assert test="count(preceding-sibling::*) = 0">
    The <name/> element must be the first child element its parent.
  </assert>
</rule>
```


2

Music Collection Schema (Cont'd)

```
<rule context="m:title">
  <assert test="parent::m:cd">
    The parent of <name/> elements must be cd.
  </assert>
  <assert test="count(*) = 0">
    The element <name/> cannot contain child elements.
  </assert>
</rule>

<rule context="*">
  <report test="true()">
    The element <name/> is not a valid element of the music namespace.
  </report>
</rule>

</pattern>

</schema>
```

catches all nodes not matched by previous rules within the current pattern

2

Diagnostics

- Optional descriptions of validation errors that provide information beyond what is in `assert/report` messages
  - such as actual/expected values and hints to repair the document
- Also useful when the same diagnostic message is desired for multiple `assert/report` elements
- Example

```
<diagnostics>
  <diagnostic id="artistDetail">
    in artist named <value-of select="name"/>
    with vocals of <value-of select="@vocals"/>
  </diagnostic>
</diagnostics>
```

- Implementations aren't required to support these

spec. doesn't say whether these messages should be output before or after message in `assert/report` (in Jing it is after)

a child of `schema` that follows `phase` elements

to include line breaks under Windows, use `
`

evaluated relative to current context node

2

Diagnostics (Cont'd)

- Referred to by `assert` and `report` elements
  - using a space-separated list of `diagnostic` element ids in the `diagnostics` attribute
  - example

```
<assert test="boolean-xpath"
  diagnostics="artistDetail">message</assert>
```

- Diagnostic messages are only output if enabled
  - details are implementation specific
  - with reference implementation
    - set "`diagnose`" stylesheet parameter to "`yes`" when generating new stylesheet from schema and ref. impl. XSLT
    - when running Xalan from command line, add "`-PARAM diagnose yes`"
  - with Jing
    - add "`-d`" command-line option

support for diagnostics seems to be broken in the ref. impl.

2

Checking For Duplicates -
the XML

```
<movies xmlns="http://www.ociweb.com/movies">
  <movie>
    <title>Elf</title>
    <actor name="Will Ferrell" role="Buddy"/>
    <actor name="James Caan" role="Walter"/>
    <actor name="Bob Newhart" role="Papa Elf"/>
    <actor name="Edward Asner" role="Santa"/>
    <actor name="Mary Steenburgen" role="Emily"/>
    <actor name="Zooey Deschanel" role="Jovie"/>
    <actor name="Mark Volkmann" role="Buddy"/>
    <actor name="Edward Asner" role="Mr. Grant"/>
  </movie>
</movies>
```

duplicate

2

Checking For Duplicates -
the schema

```
<schema xmlns="http://www.ascc.net/xml/schematron">
  <ns prefix="m" uri="http://www.ociweb.com/movies"/>

  <pattern name="all">
    <rule context="m:actor">
      <report test="@role=preceding-sibling::m:actor/@role"
        diagnostics="duplicateActorRole">
        Duplicate role!
      </report>
    </rule>
  </pattern>

  <diagnostics>
    <diagnostic id="duplicateActorRole">
      More than one actor plays the role <value-of select="@role"/>.
      A duplicate is named <value-of select="@name"/>.
    </diagnostic>
  </diagnostics>
</schema>
```

this `actor` can't play the same `role` as any that is a sibling (in same `movie`) and comes before it

can get name of first actor playing the role, "Will Ferrell" in this case, with
`<value-of select="preceding-sibling::m:actor[./@role = @role]/@name"`

Lab #1

The Windows executable version of Jing used for the RELAX-NG labs doesn't support Schematron, but the Java JAR version in this directory does.

- Setup
  - copy `labs\Schematron` from the instructor PC to your directory
- Create an XML document that conforms to a given Schematron schema
- Steps
  - study `movies.sch`
  - rename the solution from `movies.xml` to `solution.xml`
  - create your own `movies.xml` that conforms to the supplied `movies.sch`
  - validate by running the supplied script `jing.bat`
- What does this schema do that other schema languages cannot?
  - verifies that no two actors have the same name
  - verifies that no two actors play the same role

contains a couple of validation errors just to demonstrate that the schema is working

The schema requires the elements in the XML document to be in a certain **namespace**. Declare this as the default namespace on the root element.

Lab #2

- Write your own Schematron schema that validates cards in a poker hand
  - for root element `hand`
    - has no attributes
    - contains five `card` elements
    - has no other elements
  - for each `card`
    - parent is `hand`
    - has `rank` and `suit` attributes
    - has no other attributes
    - has no child elements
    - validate `suit` – heart, diamond, club or spade
    - validate `rank` – 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King or Ace
    - check for duplicate cards - **can't do this** until we have support for XPath 2!

```
<report test="concat(@rank,@suit)=
  preceding-sibling::card/concat(@rank,@suit)"
  diagnostics="duplicateCard"/>
```

Example XML document

```
<hand>
  <card rank="King" suit="heart"/>
  ...
</hand>
```

see test for valid `vocals` attribute on page 22

Lab #2 (Cont'd)

- Steps
  - rename the solution from `hand.sch` to `solution.sch`
  - create your own `hand.sch`
  - validate the supplied XML document `hand.xml` by running the supplied script `jing.bat`

The elements in the XML document are not in any **namespace**, so the schema doesn't need to map a prefix to a namespace with an `ns` element or use prefixes on elements in `test` attributes.

3

Abstract Rules

- Rules can be abstract, in which case they have no context
  - example

```
<rule abstract="true" id="team">
  ... assertions that apply to all kinds of teams ...
</rule>
```

  - **abstract rules** require an `id` attribute
  - **non-abstract rules** require a `context` attribute
    - either have no `abstract` attribute or `abstract="false"`
- Rules can extend abstract rules to add their assertions
  - `extends` element is replaced by content of referenced rule
  - example

```
<rule context="homeTeam">
  <extends rule="team"/>
  ... can add more assertions here ...
</rule>
```

`vistingTeam` rule would also extend the `team` abstract rule (see example on page 35)

3

Allowed Values From Document -

the XML documents

schedule.xml

```
<schedule xmlns="http://www.ociweb.com/football">
  <game date="12/27/2004" time="9ET">
    <homeTeam name="Rams"/>
    <visitingTeam name="Eagles"/>
  </game>
</schedule>
```

teams.xml - used to check for valid team names

```
<teams>
 <team name="Bills"/>
 <team name="Dolphins"/>
 <team name="Patriots"/>
 <team name="Jets"/>
 <team name="Ravens"/>
 <team name="Bengals"/>
 <team name="Browns"/>
 <team name="Steelers"/>
 <team name="Texans"/>
 <team name="Colts"/>
 <team name="Jaguars"/>
 <team name="Titans"/>
 <team name="Broncos"/>
 <team name="Chiefs"/>
 <team name="Raiders"/>
 <team name="Chargers"/>
 <team name="Cowboys"/>
 <team name="Giants"/>
 <team name="Eagles"/>
 <team name="Redskins"/>
 <team name="Bears"/>
 <team name="Lions"/>
 <team name="Packers"/>
 <team name="Vikings"/>
 <team name="Falcons"/>
 <team name="Panthers"/>
 <team name="Saints"/>
 <team name="Buccaneers"/>
 <team name="Cardinals"/>
 <team name="Rams"/>
 <team name="49ers"/>
 <team name="Seahawks"/>
</teams>
```


3

Allowed Values From Document -

the schema

```
<schema xmlns="http://www.ascc.net/xml/schematron">
  <ns prefix="f" uri="http://www.ociweb.com/football"/>

  <pattern name="all">
    <rule context="f:homeTeam">
      <extends rule="team"/>
    </rule>
    <rule context="f:visitingTeam">
      <extends rule="team"/>
    </rule>
    <rule abstract="true" id="team">
      <assert test="@name = document('teams.xml')//team/@name"
        diagnostics="badTeamName">
        An invalid team name was found.
      </assert>
    </rule>
```

abstract rule

the name attribute must match some team element name attribute in teams.xml

3

Allowed Values From Document -
the schema (cont'd)

```
    <rule context="f:game">
      <report test="f:homeTeam/@name = f:visitingTeam/@name"
         diagnostics="listTeams">
        A team can't play itself.
      </report>
    </rule>
  </pattern>

  <diagnostics>
    <diagnostic id="badTeamName">
      <value-of select="@name"/> is not a valid team name.
    </diagnostic>
    <diagnostic id="listTeams">
      Home team is <value-of select="f:homeTeam/@name"/>.
      Visiting team is <value-of select="f:visitingTeam/@name"/>.
    </diagnostic>
  </diagnostics>
</schema>
```


3

Phases

- Optional, named groups of patterns
- Can evaluate only the rules of specific patterns instead of evaluating all rules in all patterns
  - by specifying a phase id

in most cases it will be desirable to evaluate all the patterns

- Options for specifying the phase to evaluate include
  - command-line option
  - selection in a GUI
  - parameter in API call
- Syntax

```
<phase id="phase-id">
  <active pattern="pattern-id"/>
  ... more active elements go here ...
</phase>
```

4

Let

The ref. impl. and Jing do not support this!

- Used to define variables that can be used in XPath expressions
- Syntax
  - `<let name="name" value="value"/>`
- Can appear as a child of `schema`, `phase`, `pattern` or `rule`
  - when a child of `rule`, it is evaluated relative to the rule context
  - otherwise it is evaluated relative to document root
- Example

```
<rule context="box">
  <let name="volume" value="width * length * height"/>
  <assert test="$volume > 10">box has insufficient volume</assert>
</rule>
```


4

Abstract Patterns

The ref. impl. and Jing do not support this!

- Patterns can be abstract
  - allows a set of rules to be parameterized to support reuse for similar XML structures
- Patterns can incorporate rules of abstract patterns
  - using `is-a` attribute and `param` child elements
  - allows one pattern to "inherit" the assertions of another

4

Abstract Patterns (Cont'd)

The ref. impl. and Jing do not support this!

example XML

the **table** and **worksheet** elements have similar structure

```
<root>
  <table>
    <tr>
      <th>Player</th> <th>Number</th>
    </tr>
    <tr>
      <td>Wayne Gretzky</td> <td>99</td>
    </tr>
  </table>
  <worksheet>
    <row>
      <cell>Player</cell> <cell>Number</cell>
    </row>
    <row>
      <cell>Wayne Gretzky</cell> <cell>99</cell>
    </row>
  </worksheet>
</root>
```


4

Abstract Patterns (Cont'd)

The ref. impl. and Jing do not support this!

example schema

```
<schema xmlns="http://www.ascc.net/xml/schematron">
  <pattern abstract="true" id="table">
    <rule context="$table">
      <assert test="$row">tables must contain row elements</assert>
    </rule>
    <rule context="$row">
      <assert test="$entry">rows must contain entry elements</assert>
    </rule>
  </pattern>
  <pattern is-a="table" id="html">
    <param name="table" value="table"/>
    <param name="row" value="tr"/>
    <param name="entry" value="th|td"/>
  </pattern>
  <pattern is-a="table" id="spreadsheet">
    <param name="table" value="worksheet"/>
    <param name="row" value="row"/>
    <param name="entry" value="cell"/>
  </pattern>
</schema>
```

5

Implementations

- Reference implementation - Schematron 1.5
  - free from Academia Sinica Computing Centre
    - http://xml.ascc.net/schematron/1.5/
  - implemented as an XSLT stylesheet
- Jing
  - free from James Clark
    - http://www.thaiopensource.com/relaxng/jing.html jing-20030619.zip
  - supports RELAX NG and Schematron
- Topologi Schematron Validator
  - commercial from topologi - $495
    - http://www.topologi.com/
    - Rick Jelliffe is C.T.O. of this company
  - supports DTD, Schematron, RELAX NG and XML Schema

5

Implementations (Cont'd)

- ZVON Schematron - based on XSLT
- 4Suite - from FourThought; for Python
- XML::Schematron - from Kip Hampton; for Perl
- Xmlform - from Ivelin Ivanov; for C++
- Schematron.NET - from Daniel Cazzulino; for .NET

see http://xml.ascc.net/schematron/ for URLs of these

5

Using an XSLT-based Implementation

- The reference implementation is based on XSLT
  - see `skeleton1-5.xsl`
- Steps to use
  - apply implementation XSLT to Schematron schema to produce new XSLT
  - apply new XSLT to an instance document to output validation errors
- Can use Xalan from a script

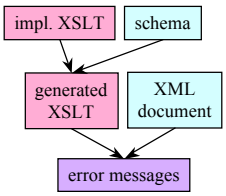
```
java org.apache.xalan.xslt.Process
  -IN mySchema.sch
  -XSL skeleton1-5.xsl
  -OUT generated.xslt
java org.apache.xalan.xslt.Process
  -IN myDocument.xml
  -XSL generated.xslt
  -TEXT
```

Schematron schema → `-IN mySchema.sch`

instance document → `-IN myDocument.xml`

on a single line

on a single line



5

Using Jing

- Everything needed is in a single Java Archive (JAR) file
  - must install Java
- Command-line usage

```
set JING_HOME=/XML/Jing/jing-20030619
java -cp %JING_HOME%/bin/jing.jar
  [-d]
  [-p phase-id]
  com.thaiopensource.relaxng.util.Driver
  schematron-schema-name.sch document-name.xml
```

[-d] enables diagnostics

on a single line

- Can also be used from Java applications
  - Java API for RELAX Verifiers (JARV)
    - see http://iso-relax.sourceforge.net/JARV/
    - not well supported by Jing yet
  - native API
    - see example on next page

5

Jing Native API

- Java code to validate against a Schematron schema

```
import com.thaiopensource.validate.ValidationDriver;

...

ValidationDriver driver = new ValidationDriver();

driver.loadSchema(ValidationDriver.fileInputSource(schemaPath));

boolean valid =
  driver.validate(ValidationDriver.fileInputSource(xmlPath));
```


Summary

- Schematron can validate things that can't be validated in other XML schema languages
- Effective use of Schematron requires becoming proficient with XPath
- Use it in conjunction with DTD, XML Schema or RELAX NG