Advanced topics in ODD

Sebastian Rahtz

Advanced topics in ODD

Sebastian Rahtz

October 2014

Reminder: the inline documentation elements

<code>	literal code from some formal language: `count = 56;`
<att>	attribute name: *@target*
<gi>	element name: <table>
<ident>	identifier or name for an object of some kind in a formal language: $content
<tag>	identifier or name for an object of some kind in a formal language.: `<ptr target="http://www.bbc.co.uk"/>`
<val>	attribute value: ‘unknown’

More on Schematron constraints

- We have seen that an element specification can contain a <constraintSpec> element which contains rules about its content expressed as ISO Schematron *constraints*

```
<elementSpec ident="div"
 module="teistructure" mode="change"
   xmlns:s="http://purl.oclc.org/dsdl/schematron">
 <constraintSpec ident="div"
   scheme="isoschematron">
   <constraint>
    <s:assert test="@type='prose' and .//tei:p">prose must
include a paragraph</s:assert>
   </constraint>
 </constraintSpec>
</elementSpec>
```

A <constraintSpec> in TEI ODD

The rule applies to the context of the element in which it is defined.

- It must have a *@scheme* to identify the constraint language (‘isoschematron’)
- It must have a unique identifier
- It contains one or more <constraint>
- Each <constraint> has an <assert> or <report> in the `http://purl.oclc.org/dsdl/schematron` namespace
- The *@test* attribute is an XPath expression. The prefix `tei` is defined in the TEI for you

Writing Schematron XPath expressions

- The <assert> element prints its body text if the expression resolves to false
- The <report> element prints its body text if the expression resolves to true
- You can use <name> in the message text, to give the context, and <value-of> to show values of eg attributes

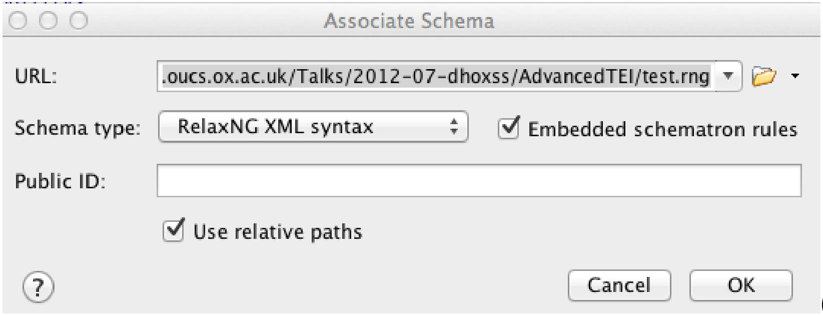
There are other Schematron facilities to help give more useful reports, but the XPath expression is the key tool.

```
http://www.schematron.com/
```

Using the Schematron rules

You have various ways of using the rules:

1. Ask oXygen to use the Schematron embedded in a RELAX NG schema:



URL: .oucs.ox.ac.uk/Talks/2012-07-dhoxss/AdvancedTEI/test.rng

Schema type: RelaxNG XML syntax

[✓ Embedded schematron rules]

Public ID:

[✓ Use relative paths]

(r

that it must be an XML, not Compact, schema)

2. Ask Roma to extract the Schematron rules into a file, and compile that into XSLT

Amongst the sort of things you can check with Schematron

- Co-occurrence constraints: ‘if there is an attribute X, there must also be a Y’
- Contextual counting: ‘there can only be one <title> child of a <titleStmt>’
- Text content: ‘The word SECRET cannot appear in an author name’
- Contextual constraint: ‘Words in English (xml:lang=’en’) cannot occur inside Latin phrases (xml:lang=’la’)’
- Referential integrity: ‘a pointer URL starting with a # must have a corresponding xml:id somewhere in the document’

Complex schematron

```
<constraintSpec ident="validtarget"
 scheme="isoschematron">
 <constraint>
  <s:rule context="tei:*[@target]">
   <s:let name="results"
    value="for $t in
tokenize(normalize-space(@target),'\s+') return
starts-with($t,'#') and not(id(substring($t,2)))"/>
   <s:report test="some $x in $results satisfies $x">Error:
Every local pointer in "<s:value-of select="@target"/>"
must point to an ID in this document
(<s:value-of select="$results"/>)</s:report>
  </s:rule>
 </constraint>
</constraintSpec>
```

Techniques used in that example

- `normalize-space(@target)`: make sure there is no trailing space
- `tokenize(normalize-space(@target),'\s+')`: break the attribute up into a sequence of space-separated tokens
- `starts-with($t,'#')`: only local references of interest
- `not(id(substring($t,2)))`: is there something with an *@xml:id* attribute which is same as the current value without the first character
- `some $x in $results satisfies $x`: XPath expression to check a sequence of true/false values

Copying the Schematron approach

You could also write a simple XSLT of your own to test your document, simply using <xsl:message>.

```
<xsl:template match="q">
 <xsl:if test="count(ancestor-or-self::q)>3">
  <xsl:message>Quotes nested 3 deep?
      really????</xsl:message>
 </xsl:if>
</xsl:template>
```

Processing ODD

An ODD processor:

- assembles all the components referenced or directly provided
- resolves multiple declarations
- may do some validity checking
- can produce a schema in one or more formal languages
- can produce a ”plain” XML document with selected documentary components

http://www.tei-c.org/Roma/

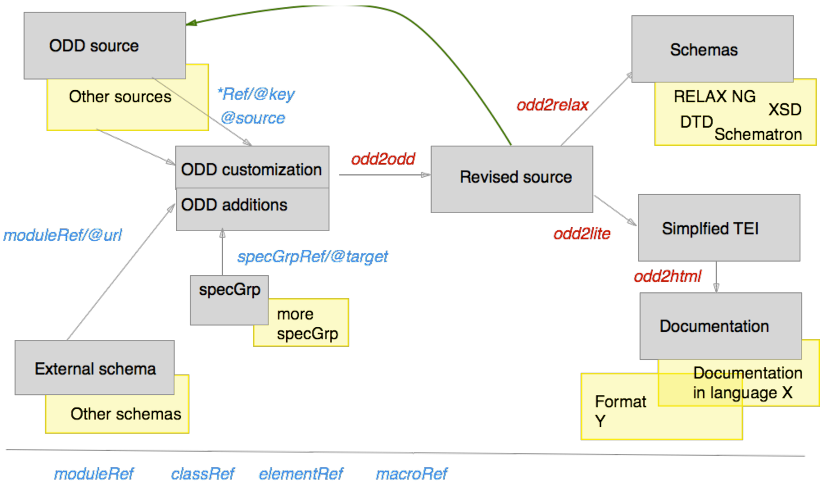
http://tei.it.ox.ac.uk/Byzantium/

http://oxgarage.oucs.ox.ac.uk:8080/ege-webclient/

How TEI ODD customization documentation is made

- assemble merge of customization and TEI into new master copy
- transform ODD-specific elements into TEI Lite, eg build reference documentation as TEI tables
- transform TEI Lite to (eg) Word, LaTeX, XSL FO, Markdown (some work to do there....)

ODD flow diagram



The attributes of <schemaSpec>

source	specifies the source from which declarations and definitions for the components of the object being defined may be obtained.
start	specifies entry points to the schema, i.e. which patterns may be used as the root of documents conforming to it.
prefix	specifies a default prefix which will be prepended to all patterns relating to TEI elements, unless otherwise stated.
targetLang	specifies which language to use when creating the objects in a schema if names for elements or attributes are available in more than one language
docLang	specifies which languages to use when creating documentation if the description for an element, attribute, class or macro is available in more than one language

What about this *@prefix* thing?

Schema languages describe a flat set of objects (‘patterns’ in RELAXNG, ‘entities’ in DTD) which other objects point to. Typically, named the same as elements. So what if we have two of the same name?
in TEI:

```
<rng:define name="tei_list">
 <rng:element name="list">
<!-- ... -->
 </rng:element>
</rng:define>
```

and in MathML

```
<rng:define name="mathml.list">
 <rng:element name="list">
  <rng:ref name="mathml.attlist-list"/>
  <rng:ref name="mathml.ContentExpression"/>
 </rng:element>
</rng:define>
```

so prefixes solve the problem of conflict.

Examples of *@source*

@source tells the processor where to read.

```
<schemaSpec ident="test1"
 prefix="tei_" start="TEI"
 source="http://www.tei-
c.org/Vault/P5/current/xml/tei/odd/p5subset.xml">
<!-- ... -->
</schemaSpec>
<schemaSpec ident="test2"
 prefix="tei_" start="TEI" source="tei:1.5.0">
<!-- ... -->
</schemaSpec>
<schemaSpec ident="test3"
 prefix="tei_" start="TEI"
 source="http://www.tei-
c.org/Vault/P5/1.5.0/xml/tei/odd/p5subset.xml">
<!-- ... -->
</schemaSpec>
<schemaSpec ident="test4"
 source="file:///home/lou/Public/TEI-
SF/P5/Source/Guidelines/en/guidelines-en.xml">
<!-- ... -->
</schemaSpec>
```

Note the use of the private `tei:` URI protocol.

Chaining ODDS - building your own TEI base

Normally, an operation like an ‘ODD to RELAX NG’ transform does a sequence of work for you (‘odd to odd’ followed by ‘odd to relax’); to make a new source for chaining, you need to stop after the first stage

- The command-line roma has a `--compile` option
- The command-line script teitoodd does the same job
- OxGarage can convert from format ‘ODD Document’ to ‘Compiled ODD Document’
- in oXygen you have to build a new transform using the XSLT odd2odd.xsl; make this an Ant-based task, and pick up the file stylesheets/odd/build-to.xml

Chaining ODDS - customizing from a new source

@source on <schemaSpec> is optional and defaults to http://www.tei-c.org/Vault/P5/current/xml/tei/odd/p5subset.xml. So all you need to do is specify it explicitly:

```
<schemaSpec ident="test-pure"
 source="/Users/rahtz/TEI/tei.oucs.ox.ac.uk/Talks/2014-10-
odds/examples/alienbase.odd" start="TEI" prefix="alien_">
 <moduleRef key="tei"/>
 <moduleRef key="header"/>
 <moduleRef key="core"/>
 <moduleRef key="aliens"/>
 <moduleRef key="textstructure"/>
</schemaSpec>
```

Note the module aliens there; the customization from which is is derived has

```
<elementSpec module="aliens"
 ident="alien" ns="http://www.dixit.eu/ns/"
 mode="add">
<!-- ... -->
</elementSpec>
```

Mixing in foreign schema components

What if we want to extend TEI's <formula> to permit MathML content? We have to do three things:

1. Pull in the MathML schema
2. Adjust the content model of <formula>
3. Generate the TEI schema in a such a way that it doesn't conflict with MathML

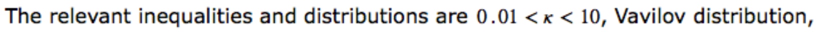
TEI + MathML

```
<schemaSpec ident="tei_math"
 prefix="tei_" start="TEI teiCorpus">
 <moduleRef url="http://www.tei-
c.org/release/xml/tei/custom/schema/relaxng/mathml2-
main.rng"/>
 <moduleRef key="header"/>
 <moduleRef key="core"/>
 <moduleRef key="tei"/>
 <moduleRef key="textstructure"/>
 <moduleRef key="figures"/>
 <elementSpec module="figures"
  ident="formula" mode="change">
  <content>
   <rng:ref name="mathml.math"/>
  </content>
 </elementSpec>
</schemaSpec>
```

TEI and Math example

That customization means that the following is now valid:

```
<p>The relevant inequalities and distributions are
<formula notation="MathML">
  <m:math overflow="scroll">
   <m:mn>0</m:mn>
   <m:mo>.</m:mo>
   <m:mn>0</m:mn>
   <m:mn>1</m:mn>
   <m:mo><</m:mo>
   <m:mi>κ</m:mi>
   <m:mo><</m:mo>
   <m:mn>1</m:mn>
   <m:mn>0</m:mn>
  </m:math>
 </formula>, Vavilov distribution,
and ...
</p>
```



The relevant inequalities and distributions are $0.01 < \kappa < 10$, Vavilov distribution,

Merging SVG uses a different technique

```
<moduleRef url="http://www.tei-
c.org/release/xml/tei/custom/schema/relaxng/svg11.rng">
 <content>
  <rng:define name="tei_model.graphicLike"
   combine="choice">
   <rng:ref name="svg"/>
  </rng:define>
 </content>
</moduleRef>
```

In a rather unstable way, we add an SVG pattern to a TEI model class.

This technique may well not work in the Pure ODD world.

ODD by Example

What if you want to create an ODD by examining your current practice?

oddbyexample is your friend.

On a command line:

```
java -Xmx12000m -jar /usr/share/saxon/saxon9he.jar \
 -it:main -
xsl:/usr/share/xml/tei/stylesheet/tools/oddbyexample.xsl \
corpus=/tmp/texts
```

with many parameters you can set to vary its behaviour.

oddbyexample needs plenty of memory provided to Java on your computer to operate on big corpora.

What does oddbyexample actually do?

1. create a variable and copy in all of the TEI
2. read the corpus and get a list of all the elements and their attributes that it uses
3. process the new corpus list and compare with the TEI section. if an element or attribute is not present in the corpus section, put out a delete customization; if the attributes of an attribute class are never used that class may be deleted only if it doesn't claim membership in any other class or, if it does, none of the attributes from that other class is used too
4. for every attribute which is of type "enumerated", construct a valList

Parameters in oddbyexample

name	default	explanation
schema	oddbyexample	name of result
keepGlobals	false	whether to do all the global attributes
corpus	./	the path to document corpus
prefix		file names starting with what prefix?
suffix	xml	file names ending with what suffix?
includeHeader	true	should elements in teiHeader be included
defaultSource	http://www.tei-c.org/Vault/P5/current/xml/tei/odd/p5subset.xml	
enumerateRend	false	should we make valList for @rend and @rendition
enumerateType	false	should we make valList for @type
processNonTEI	false	should we deal with non-TEI namespaces
attributeList		which attributes should be make valLists for, regardless
method	include	moduleRef generated with @include or @except?
debug	false	turn on debug messages
verbose	false	turn on messages
corpusList		provide specific list of files
processP4	false	should P4 files be considered?
processP5	true	should P5 files be considered?

Setting up oddbyexample in oXygen

- Open any TEI XML file in the target collection of texts
- Choose Transformation -> Configure Transformation Scenario(s) from theDocument menu
- Click New and choose "XML Transformation with XSLT"
- Give your scenario a name ("oddGenerator" for example)
- Leave XML URL as it is. Change XSL URL to point to the stylesheet oddbyexample.xsl in your TEI Framework directory directory. Enter `${frame-works}/tei/xml/tei/stylesheet/tools/oddbyexample` to find it
- Choose Saxon-PE 9.4.0.4 as processor
- Click the little yellow wheel next to this window to select Advanced Options: you need to set `Template("-it")` to `main`

Setting up oddbyexample in oXygen (cont.)

- Click the Parameters button : you need to set the `corpus` parameter to contain the full name of the folder which you want to analyse. Assuming you opened one of its files in step 1 above, just set the parameter to `${cfdu}` and click OK
- Now select the Output tab ...
  - In the Save as window supply an output filename such as generated.odd
  - Tick the Open in editor box
  - Select the XML radio button underneath Show in results view as and click OK
- Launch the transformation by clicking the Apply Associated button
- If everything works, you should see the resulting ODD file generated.odd

Other ODD outputs

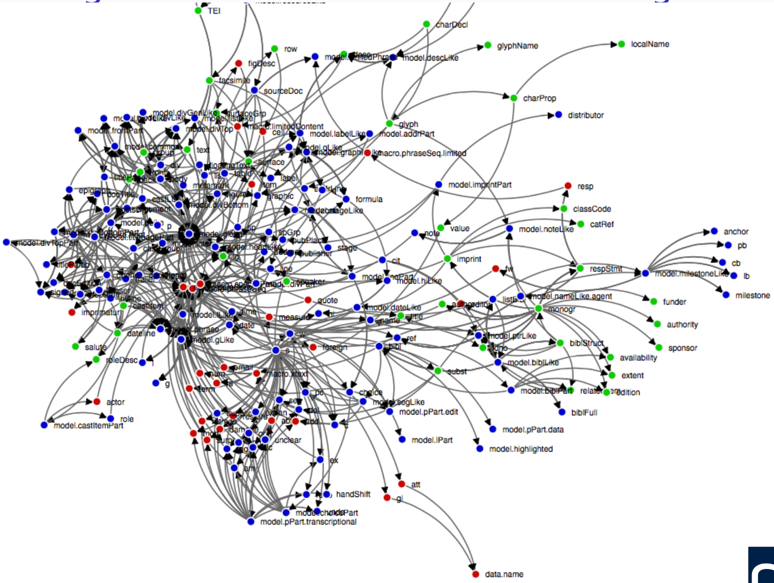
You are already aware that the current ODD tools can produce

schemas	DTD, XSD, RELAX NG (XML and compact), ISO Schematron
documentation	HTML, LaTeX (and thence PDF), XSL FO, Word etc)

but there are at least two other transformations available

JSON	`odds/odd2json.xsl` produces a generic JSON output which you can use with any of the myriad Javascript libraries
JSON	`tools/odd-to-tree.xsl` produces a generic JSON output which you can use with any of the myriad Javascript libraries

which gives us the chance to consider things like...



Making your own dancing graphs

There are a selection of force-directed graphs for TEI customizations available at `http://tei.oucs.ox.ac.uk/Talks/2014-10-teimm-visualization/showviz.html`, using CSV files which are generated by running the stylesheet `https://github.com/TEIC/Stylesheets/blob/master/tools/odd-to-csv.xsl` on a compiled ODD.