Advanced TEI customisation and documentation

TEI @ Oxford

September 2014

Going deeper into ODD

- Overview of the ODD language
- Roma, OxGarage and oXygen: processing your ODD
- Using ODD and Schematron to get tight constraints

Why might you need ODD?

- You need to define an XML schema to describe your resource
- You need to provide documentation about
  - the semantics of your XML schema
  - constraints, usage notes, examples
- You need to keep the two in step
- You want to share the results
  - with others
  - with yourself, long term
- you don't want to reinvent the wheel

The design of ODD in the TEI

A special XML vocabulary for defining....

- schemas
- XML element types independent of a particular schema language
- public or private groups of such elements
- patterns (macros)
- classes (and subclasses) of element

And also for defining references which can pull into a schema

- named components from the above list
- objects from other namespaces

All embedded within conventional document markup elements

The basic idea (2)

An ODD processor:

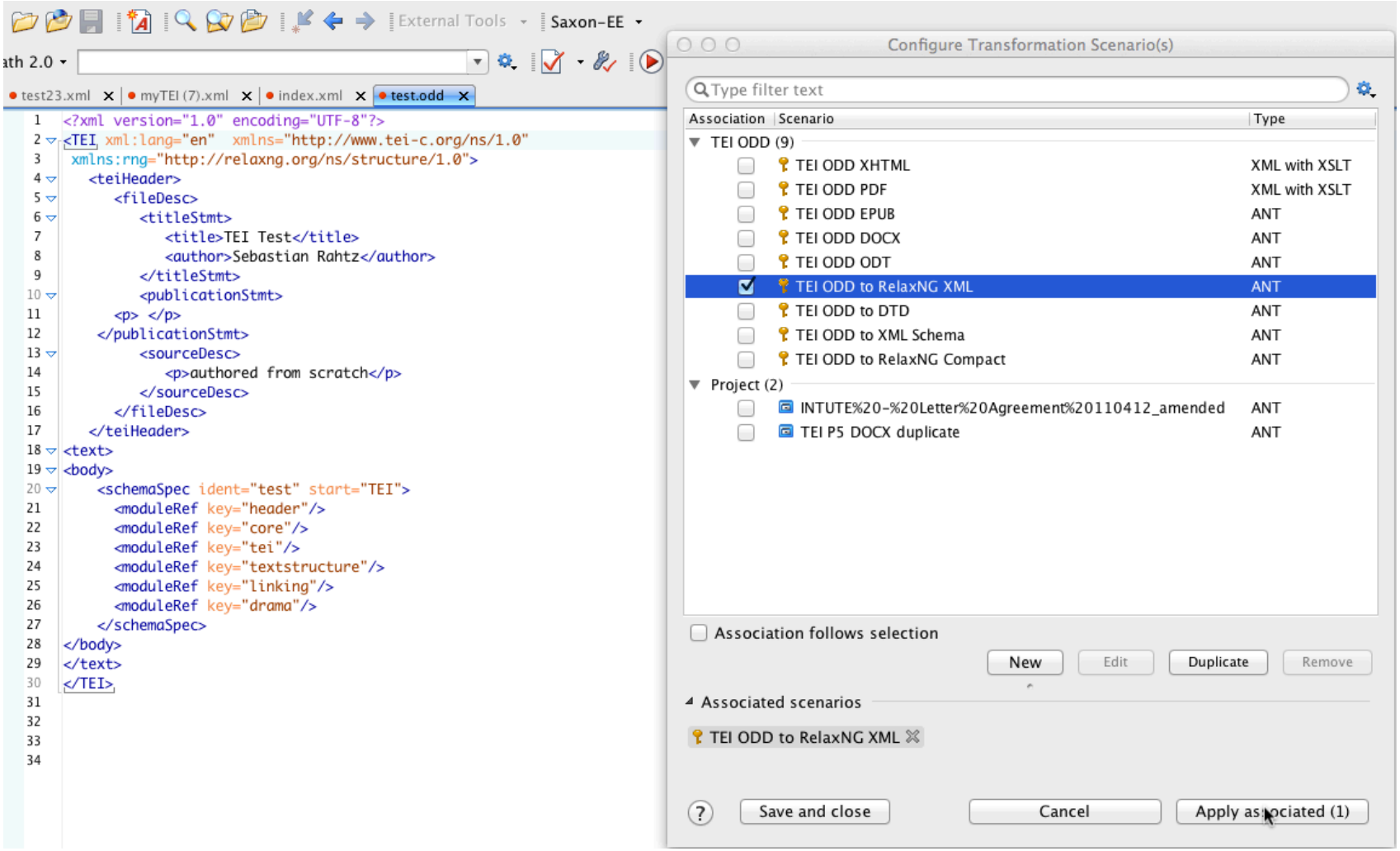
- assembles all the components referenced or directly provided
- resolves multiple declarations
- may do some validity checking
- can produce a schema in one or more formal languages
- can produce a "plain" XML document with selected documentary components

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

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

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

ODD conversion in oXygen



ODD conversion in Oxgarage

Select file to convert: ?

Choose File test.odd

+ Show advanced options ?

Upload images: ?

You can upload image files and .zip files containing images

Choose File No file chosen - Remove

+ Add more images

Convert Reset

Convert from: ?

Documents

- Compiled TEI ODD
- DocBook Document
- Microsoft Word (.doc)
- Microsoft Word (.docx)
- ODD Document
- OpenOffice 1.0 Text (.sxw)
- OpenOffice Text (.odt)
- Plain Text (.txt)
- Rich Text Format (.rtf)
- TCP XML Document
- TEI P4 XML Document
- TEI P5 XML Document
- TEI Tite XML Document
- WordPerfect (.wpd)
- xHTML

Presentations

Convert to: ?

- Comma-Separated Values (.csv)
- Compiled ODD Document
- DTD created from ODD
- ePub
- ISO Schematron constraints
- LaTeX
- Microsoft Excel (.xls)
- Microsoft Word (.doc)
- Microsoft Word (.docx)
- National Library of Medicine (NLM) DTD 3.0
- ODD documentation as HTML
- ODD documentation as TEI Lite
- OpenOffice (.ods)
- OpenOffice 1.0 (.sxc)
- OpenOffice 1.0 Text (.sxw)
- OpenOffice Text (.odt)
- PDF
- Plain text
- RDF XML
- RELAX NG compact schema
- RELAX NG schema

A simple example

We have <book>, which contains a mixture of <para>s and <picture>s. We have never heard of the TEI and we don't want to use it.

```
<schemaSpec  ns=""  start="book"
  ident="bookschema">
  <elementSpec  ident="book">
    <desc>Root element for a very simple schema</desc>
    <content>
      <rng:oneOrMore>
        <rng:choice>
          <rng:ref  name="para"/>
          <rng:ref  name="picture"/>
        </rng:choice>
      </rng:oneOrMore>
    </content>
  </elementSpec>
<!-- ...  continues on next slide -->
</schemaSpec>
```

A simple example, contd.

```
<!-- ... contd --><elementSpec  ident="picture">
  <desc>Empty element to point at a picture</desc>
  <content>
    <rng:empty/>
  </content>
  <attList>
    <attDef  ident="href">
      <desc>supplies the URI of the graphic pointed at</desc>
      <datatype>
        <rng:data  type="anyURI"/>
      </datatype>
    </attDef>
  </attList>
</elementSpec>
<elementSpec  ident="para">
  <desc>textual element in a very simple schema (may have pictures in it)</desc>
  <content>
    <rng:zeroOrMore>
      <rng:choice>
        <rng:text/>
        <rng:ref  name="picture"/>
      </rng:choice>
    </rng:zeroOrMore>
  </content>
</elementSpec>
```

So what?

- We can now build a schema in RELAX NG, W3C schema, or DTD language by a simple XSLT transformation
- We can also extract documentary fragments (e.g. the descriptions of elements and attributes)

TEI provides a special element for the latter purpose:

```
<specList>
  <specDesc  key="para"/>
  <specDesc  key="picture"/>
</specList>
```

which would generate something like

<para> textual element in a very simple schema (may have pictures in it)

<picture> Empty element to point at a picture

inside our running text

What else might you want to say about your elements?

- alternative <desc>s in different languages maybe?
- some reference usage examples
- Schematron constraints
- value lists
- class memberships

Alternative description example in TEI

```
<elementSpec module="core" ident="p">
  <gloss>paragraph</gloss>
  <gloss version="2007-05-02"
    xml:lang="zh-tw">段落</gloss>
  <desc>marks paragraphs in prose.</desc>
  <desc version="2007-05-02"
    xml:lang="zh-tw">標記散文的段落 。</desc>
  <desc version="2008-04-05" xml:lang="ja"> 散文の段落を示す.  </desc>
  <desc version="2009-01-06" xml:lang="fr">marque les paragraphes dans un
texte en
     prose.</desc>
  <desc version="2007-05-04" xml:lang="es">marca párrafos en prosa.</desc>
  <desc version="2007-01-21" xml:lang="it">indica i paragrafi in
prosa</desc>
<!-- ... -->
</elementSpec>
```

Usage examples

The <exemplum> element combines an XML example with some discussion of it.

```
<exemplum xml:lang="en">
<egXML xmlns="http://www.tei-c.org/ns/Examples">
<langUsage>
<language ident="en">English</language>
</langUsage></egXML>
<p>Note the use of ISO language codes</p>
</exemplum>
```

Note the use of the <egXML> element which declares itself and its content as belonging to the namespace http://www.tei-c.org/ns/Examples. This is to let the content of the element be validated independently

Defining the content of an element

- We can use a set of special TEI elements to describe content for elements and attributes
- or we use can RELAX NG directly to define content
- Generated patterns are uniquified by means of an automatic prefix, which can be switched on or off
- Content can be constrained by means of a <valList> element ...
- ... or by means of a <datatype> element (which uses RELAX NG)
- Generic constraints can be expressed by means of <constraint> elements (which use e.g. ISO Schematron)

Do you really need to do this?

The TEI does actually define elements very like yours. Why not just use them?

```
<schemaSpec source="http://www.tei-c.org/release/xml/tei/odd/p5subset.xml"
  start="div" ident="simpleS-2">
  <elementRef key="div"/>
  <elementRef key="p"/>
  <elementRef key="graphic"/>
</schemaSpec>
```

The *@source* attribute is a URI of any kind, from which specifications are available. It could be a file name or a URL

Recap

- The TEI encoding scheme defines a set of *elements*
- An element definition (<elementSpec>) specifies:
  - a name (<gi>) for the element, and optionally other names in other languages
  - a description (<desc>, also possibly translated) of its function
  - a declaration of the *classes* to which it belongs (<classes>)
  - a definition for each of its *attributes* (<attList>)
  - a definition of its *content model* (<content>), ie what can appear inside it)
  - usage examples (<exemplum>) and notes (<remarks>
- *modules* are used to group together sets of elements
- a TEI *schema* specification (<schemaSpec>) is made by selecting modules or elements and (optionally) modifying their contents
- a TEI document containing a schema specification is called an *ODD* (One Document Does it

Assembling a TEI customization

A customization is described by a <schemaSpec> which can specify any or all of:

- a source from which to get components referred to
- references (<moduleRef>) to modules (which pulls in all members of those modules
- specifications for new modules (<moduleSpec>)
- references to particular elements, classes, or macros (<elementRef>, <classRef> and <macroRef>)
- specifications for new elements, classes, or macros (<elementSpec>, <classSpec>, <macroSpec>)
- specifications to change elements, classes, or macros which been referred to (<elementSpec>, <classSpec>, <macroSpec> with `mode="change"`)
- named groups of specifications which can be referred to later (<specGrp>, <specGrpRef>)

A complete TEI <elementSpec>

```
<elementSpec  module="core"  ident="add">
  <desc  versionDate="2013-04-13"
    xml:lang="en">contains letters, words, or phrases inserted in the source
     text by an author, scribe, or a previous annotator or corrector.</desc>
  <desc  versionDate="2007-05-02"
    xml:lang="zh-TW">包含由作者、抄寫者、註解者、或更正者在文本中插入的字母、單字或詞
彙。</desc>
  <classes>
    <memberOf  key="att.global"/>
    <memberOf  key="model.pPart.transcriptional"/>
    <memberOf  key="att.transcriptional"/>
    <memberOf  key="att.placement"/>
    <memberOf  key="att.typed"/>
  </classes>
  <content>
    <rng:ref  name="macro.paraContent"/>
  </content>
  <exemplum  xml:lang="en">
<!-- example omitted -->
  </exemplum>
  <remarks  xml:lang="ja"
    versionDate="2008-04-05">
    <p> 要素<gi>add</gi>は，編集者や符号化する人による追加情報を符号化する
        ために使うべきではない．この場合は，要素<gi>corr</gi>または
    <gi>supplied</gi>を使うべきである．
    </p>
  </remarks>
</elementSpec>
```

Recap: the TEI Class System

- The TEI distinguishes over 500 elements,
- Having these organised into classes aids comprehension, modularity, and modification.
- *Attribute class*: the members share common attributes
- *Model class*: they can appear in the same locations (and are often semantically related)
- Classes may contain other classes
- An element can be a member of any number of classes, irrespective of the module it belongs to.

Specifying a class

The <classSpec> element is used to declare a class. Its *@type* attribute indicates whether this is an attribute or a model class

```
<classSpec module="tei" type="model"
  ident="model.qLike">
  <desc versionDate="2005-10-10"
    xml:lang="en">groups elements related to highlighting which can appear
either within or between
     chunk-level elements.</desc>
  <classes>
    <memberOf key="model.inter"/>
  </classes>
</classSpec>
```

An attribute class

For an attribute class, the specificaton contains an <attList>, which specifies the attributes it provides:

```
<classSpec module="tei" type="atts"
 ident="att.responsibility">
 <desc versionDate="2009-11-02"
  xml:lang="en">provides attributes indicating who is responsible for
   something asserted by the markup and the degree of certainty
   associated with it.</desc>
 <classes>
  <memberOf key="att.source"/>
 </classes>
 <attList>
  <attDef ident="cert" usage="opt">
   <desc>
<!-- omitted description -->
   </desc>
   <datatype>
    <rng:ref name="data.certainty"/>
   </datatype>
  </attDef>
 </attList>
</classSpec>
```

Referring to classes

Elements are classified (i.e. classes are referenced) by means of the <memberOf> child of the <classes> element inside an <elementSpec> (and classes can also be *members-of* other classes)

Elements can also reference model classes in their content model.

```
<elementSpec  module="core"  ident="gap">
  <desc  xml:lang="en">indicates a point where material has been
     omitted in a transcription, whether for editorial
     reasons...</desc>
  <classes>
    <memberOf  key="att.global"/>
    <memberOf  key="model.global.edit"/>
    <memberOf  key="att.duration"/>
    <memberOf  key="att.editLike"/>
  </classes>
  <content>
    <rng:zeroOrMore>
      <rng:choice>
        <rng:ref  name="model.descLike"/>
        <rng:ref  name="model.certLike"/>
      </rng:choice>
    </rng:zeroOrMore>
  </content>
</elementSpec>
```

Selecting elements (1)

You can specify elements to be excluded from those provided by a module:

```
<schemaSpec start="TEI"
  ident="testSchema-4a">
  <moduleRef key="core"
    except="mentioned quote said"/>
  <moduleRef key="header"/>
  <moduleRef key="textstructure"/>
</schemaSpec>
```

This is equivalent to the following:

```
<schemaSpec start="TEI"
  ident="testSchema-4b">
  <moduleRef key="core"/>
  <moduleRef key="header"/>
  <moduleRef key="textstructure"/>
  <elementSpec ident="mentioned"
    mode="delete"/>
  <elementSpec ident="quote" mode="delete"/>
  <elementSpec ident="said" mode="delete"/>
</schemaSpec>
```

The *@mode* attribute instructs an ODD processor how to resolve multiple declarations.

Selecting elements (2)

Or you can specify just the elements you want to include:

```
<schemaSpec start="TEI"
 ident="testSchema-4b">
 <moduleRef key="core"/>
 <moduleRef key="header"/>
 <moduleRef key="textstructure"
  include="body div"/>
</schemaSpec>
```

This is equivalent to the following:

```
<schemaSpec start="TEI"
 ident="testSchema-4b">
 <moduleRef key="core"/>
 <moduleRef key="header"/>
 <elementRef key="div"/>
 <elementRef key="body"/>
</schemaSpec>
```

Unifying multiple declarations

As noted above, the *@mode* attribute controls what an ODD processor should do when it find multiple instances of some component.

mode value	existing declaration	effect
add	no	add new declaration to schema; process its children in add mode
add	yes	raise error
replace	no	raise error
replace	yes	retain existing declaration; process new children in replace mode; ignore existing children
change	no	raise error
change	yes	process identifiable children according to their modes; process unidentifiable children in replace mode; retain existing children where no replacement or change is provided
delete	no	raise error
delete	yes	ignore existing declaration and its children

@mode is for all changes

To specify changes to a TEI element, use *@mode* everywhere:

```
<elementSpec ident="div" mode="change">
  <desc mode="replace">My kind of division</desc>
  <attList>
    <attDef mode="change" ident="type"
      usage="req"/>
    <attDef mode="delete" ident="subtype"/>
    <attDef mode="add" ident="colour">
      <valList type="closed">
        <valItem ident="red"/>
        <valItem ident="green"/>
        <valItem ident="orange"/>
      </valList>
    </attDef>
  </attList>
</elementSpec>
```

Reading the Guidelines to understand *@mode*

Consider

```
http://www.tei-c.org/release/doc/tei-p5-doc/en/html/
ref-add.html
```

If you want to change the behaviour of its attributes, note that they all come from attribute classes. You need to look up the attribute to see whether you'll be doing an ‘add’, ‘delete’ ‘change’ or ‘replace’ of particular features.

Specifying elements and modules

The '*Spec' elements are all members of a class att.identifiable which provides an attribute *@ident* that is used (rather than *@xml:id*) as a unique identifier for them.

To reference such a declaration, we use the *@key* attribute:

```
<elementRef key="bar"/>
<!-- implies the presence elsewhere of ... -->
<elementSpec ident="bar">
<!-- .... -->
</elementSpec>
```

Similarly:

```
<moduleRef key="foo"/>
<!-- implies the presence elsewhere of ... -->
<moduleSpec ident="foo"/>
```

Elements in modules

But note that elements indicate the module they belong to by means of their *@module* attribute:

```
<elementSpec ident="bar" module="foo">.... </elementSpec>
```

so the <moduleSpec> is largely documentary.

Elements not declared by the TEI can be assigned to a user-defined module; its name is defaulted.

Specification of attributes

There should be an element called <attSpec> but actually it is called <attDef>. Within an <elementSpec> or a <classSpec>, you can supply an <attList> containing one or more <attDef> elements, each with an *@ident*:

```
<attList>
  <attDef ident="who">....</attDef>
</attList>
```

Specifying value lists and datatypes

In general, the legal values for an attribute are defined by means of a <datatype> element, see later.

A common case, however, is to supply an *enumeration* (a list, open or closed, of legal values). This is done using the <valList> element, which groups a bunch of identifiable <valItem> elements: like this

```
<attDef  ident="status">
  <desc>indicates the state of the system using a predefined set of colour
     codes</desc>
  <defaultVal>green</defaultVal>
  <valList  type="closed">
    <valItem  ident="red">
      <desc>all systems shut down</desc>
    </valItem>
    <valItem  ident="orange">
      <desc>systems shut-down imminent</desc>
    </valItem>
    <valItem  ident="green">
      <desc>system status normal</desc>
    </valItem>
    <valItem  ident="white">
      <desc>system status
            unrecorded</desc>
    </valItem>
  </valList>
</attDef>
```

Datatypes

Typically used to constrain attribute values:

```
<attDef ident="status">
  <datatype>
    <rng:ref name="data.enumerated"/>
  </datatype>
<!-- ... implies that a vallist is supplied -->
</attDef>
<attDef ident="lastUpdated">
  <datatype>
    <rng:ref name="data.temporalExpr.w3c"/>
  </datatype>
</attDef>
```

TEI-defined datatypes are actually patterns, defined by a <macroSpec>

Specifying a pattern

The <macroSpec> element is an identifiable element used to associate a name with any string. It has two typical uses in the TEI scheme:

- defining common content models
- defining TEI-specific datatypes

```
<macroSpec  ident="data.foo">
  <desc>a new datatype i just invented</desc>
  <content>
<!-- RELAX NG pattern defining the datatype -->
  </content>
</macroSpec>
<macroSpec  ident="macro.foo">
  <desc>a content model I plan to reuse often</desc>
  <content>
<!-- RELAX NG pattern defining the content model -->
  </content>
</macroSpec>
```

W3C datatypes in RELAX NG

The macro data.numeric looks like this

```
<rng:choice>
  <rng:data  type="double"/>
  <rng:data  type="token">
    <rng:param  name="pattern">(\-?[\d]+/\-?[\d]+)</rng:param>
  </rng:data>
  <rng:data  type="decimal"/>
</rng:choice>
```

For that *@type* attribute on <data> you can also use any of ‘string’, ‘boolean’, ‘decimal’, ‘float’, ‘double’, ‘duration’, ‘dateTime’, ‘time’, ‘date’, ‘gYearMonth’, ‘gYear’, ‘gMonthDay’, ‘gDay’, ‘gMonth’, or ‘anyURI’

Or you narrow down the definition with a <param>, eg a regular expression.

```
http://relaxng.org/
```

Defining the content model for an element

The <content> element can have either

- rules using special TEI elements:

```
<content>
  <sequence>
    <elementRef  key="head"/>
    <classRef  key="model.pLike"
      maxOccurs="infinity"  minOccurs="1"/>
  </sequence>
</content>
```

or

- rules in the RELAX NG schema language:

```
<content>
  <rng:group>
    <rng:ref  name="head"/>
    <rng:oneOrMore>
      <rng:ref  name="model.pLike"/>
    </rng:oneOrMore>
  </rng:group>
</content>
```

Pure ODD

The content-defining elements are a recent addition to the TEI, aiming to

- remove the dependency on a particular schema language
- make the TEI ODD language consistent
- provide a framework to specify rules which we may not actually be able to express in schema languages yet

Content model elements

You can use (and combine with the '*Ref' elements):

- <sequence> to indicate that child elements form a sequence within a content model
- <alternate> to indicate that child elements can be alternated within a content model

On the referring elements (eg <elementRef>) you can use the attributes:

- *@key* to point to an element in your current set
- *@preserveOrder*, to indicate whether the order in which component elements of a sequence appear in a document must correspond to the order in which they are given in the content model
- *@minOccurs* to say how many times it *must* occur
- *@maxOccurs* to say how many times it *could* occur

Default for both *@minOccurs* and *@maxOccurs* is 1.

A content model example

This content element defines a content model allowing either a sequence of paragraphs or a series of msItem elements optionally preceded by a summary:

```
<content>
  <alternate>
    <classRef  key="model.pLike"
      maxOccurs="unbounded"/>
    <sequence>
      <elementRef  key="summary"  minOccurs="0"
        maxOccurs="1"/>
      <elementRef  key="msItem"
        maxOccurs="unbounded"/>
    </sequence>
  </alternate>
</content>
```

Expanding class references

If you say `<classRef key="model.divLike"/>`, what does it mean? all of the members in any order? only one?

The *@expand* attributes indicates how references to this class within a content model should be interpreted. Legal values are:

alternate	any one member of the class may appear (default)
sequence	a single occurrence of all members of the class may appear in sequence
sequenceOptional	a single occurrence of one or more members of the class may appear in sequence
sequenceOptionalRepeatable	one or more occurrences of one or more members of the class may appear in sequence.
sequenceRepeatable	one or more occurrences of all members of the class may appear in sequence

What about simple text?

Where a content model wants to allow text, the attribute *@allowText* on <content> is set to 'true':

```
<content allowText="true">
  <alternate minOccurs="0"
    maxOccurs="unbounded">
    <classRef key="model.gLike"/>
    <classRef key="model.phrase"/>
    <classRef key="model.inter"/>
    <classRef key="model.global"/>
    <elementRef key="lg"/>
  </alternate>
</content>
```

This allows text to be mixed in with any of the elements.

NOTE: this may change. An explicit <textNode> may be supported in future.

Schematron constraints

- 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>
```

Schematron isn't a universal panacea

- You can only add Schematron rules by editing your ODD file: Roma doesn't know about this.
- You have to learn a new syntax
- Not all schema languages can implement these constraints.
- Not all validating software pays attention to Schematron

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, but not other markup

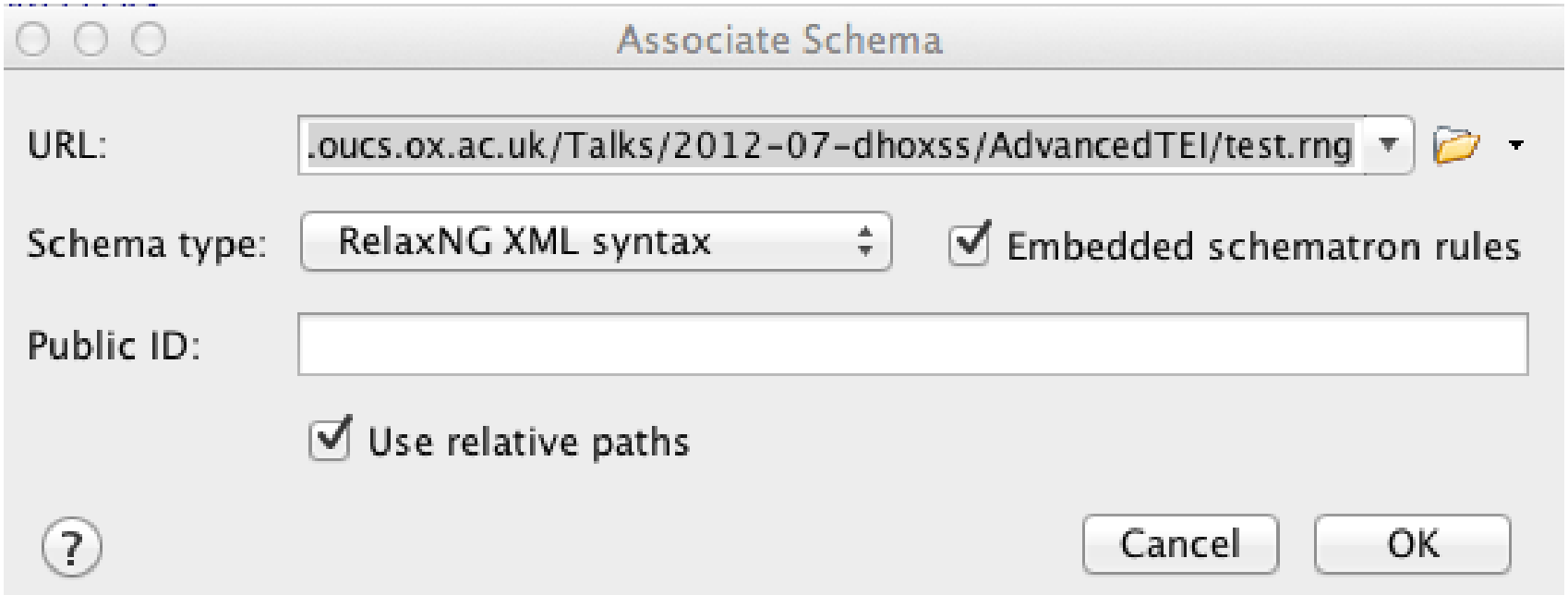
There are other Schematron facilities to help give more useful reports, but the XPath expression is the key tool.

```
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:



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’

Aside: copying the Schematron approach

You could also write a simple XSLT of your own to test your document:

```
<xsl:template match="q"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:if test="count(ancestor-or-self::q)>3">
    <xsl:message>Quotes nested 3 deep?
        really????</xsl:message>
  </xsl:if>
</xsl:template>
```

ODD features I had no space for in the margin

- mixing and matching with external vocabularies (eg MathML)
- using <equiv> to map TEI elements to other schemes
- the *@source* attribute on '*Ref' to specify where to harvest specfications from
- use of <specGrp> and <specGrpRef>
- working with declarations of the same element name in different namespaces
- how to process ODD