Tools, Documentation techniques, examples, and writing new ODDs

TEI@Oxford

November 2010

ODD processing tools

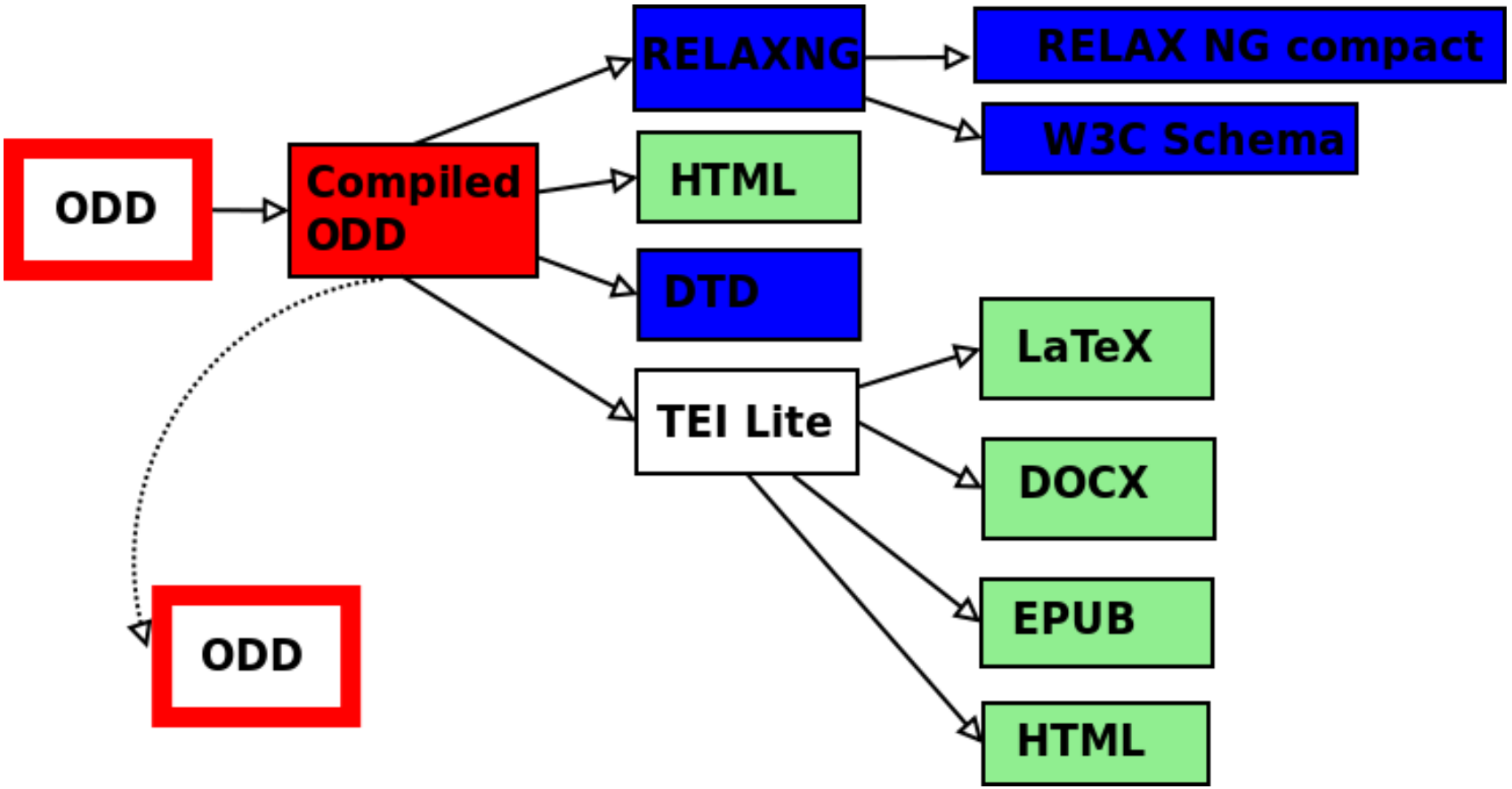
In ascending order of complexity for the user (and control):

- Web Roma: traditional at `http://www.tei-c.org/Roma/`
- Web Roma: new style at `http://tei.oucs.ox.ac.uk/Roma/` (uses improved XSLT 2.0 transformations by calling OxGarage
- Desktop Vesta application
- OxGarage web site and web service
- Command line shell script
- Run XSLT transformations by hand

What you need to know about ODD processing

- There is only one implementation
- It is written as a set of XSLT transforms
- The implementation has little error reporting (eg no if warning if you declare something twice)
- Not everything that is theoretically possible is implemented

The current ODD processing model



What is with the 'compiled ODD'?

Your ODD is rewritten so that

- all the layout <specGrpRef> (see later) and so on are expanded
- reading the specified TEI source, all the modules and/or elements and classes you asked for are pulled in
- your change, add and delete specifications are applied
- content models are simplified to remove reference to non-existent objects

The resulting intermediate file can be saved, and used as the source for a second round of writing an ODD spec against it (in theory — not to be trusted at present).

Existing conversions from compiled/flattened ODD

- ODD to DTD
- ODD to RELAX NG (the program trang is then used to RELAX NG compact, and W3C XSD)
- ODD to HTML (slightly better rendering than via Lite)
- ODD to TEI Lite

The TEI Lite version can be used with existing transformations to make HTML, LaTeX, DOCX, ODT, etc. The TEI Guidelines are made with Lite to LaTeX, and then processed with XeLaTeX.

OxGarage

OxGarage is a new web interface to the ODD processing tools.
http://oxgarage.oucs.ox.ac.uk:8080/ege-webclient

OxGarage lets you:

- generate schemas
- convert documentation to HTML, ePub, and DOCX
- convert between TEI XML and Word DOCX
- perform all the ODD tasks using web services
- chain sets of transformations together

OxGarage web service examples

Process ODD to compiled ODD, then to TEI Lite, then to DOCX

```
curl -s  -F upload=@test.odd  -o test.docx
http://oxgarage.oucs.ox.ac.uk:8080/
ege-webservice/Conversions/
ODD%3Atext%3Axml/
ODDC%3Atext%3Axml/
TEI%3Atext%3Axml/
docx%3Aapplication%3Avnd.openxmlformats-officedocument.wordprocessingml.docume t/
```

ODD to HTML, in French

```
curl -s  -F upload=@test.odd  -o test.html
http://oxgarage.oucs.ox.ac.uk:8080/ege-webservice/Conversions/
ODD%3Atext%3Axml/
ODDC%3Atext%3Axml/
oddhtml%3Aapplication%3Axhtml%2Bxml/
?properties=<conversions><conversion%20index='1'>
<property%20id='oxgarage.lang'>fr</property></conversion></conversions>
```

On the command line (Linux or Mac

Maximum power can be obtained by using a command-line Roma

```
Usage: roma [options] schemaspec [output_directory]
options, shown with defaults:
--xsl=/usr/share/xml/tei/stylesheet
--localsource=  # local copy of P5 sources
options, binary switches:
--compile          # create compiled file odd
--debug            # leave temporary files, etc.
--docflags=STRING  # pass parameters to doc creation transforms"
--dochtml          # create HTML version of doc
--doclang=LANG     # language for documentation
--docpdf           # create PDF version of doc
--lang=LANG        # language for names of attributes and elements
--nodtd            # suppress DTD creation
--norelax          # suppress RELAX NG creation
--noteic           # suppress TEI-specific features
--noxsd            # suppress W3C XML Schema creation
--schematron       # extract Schematron rules
--isoschematron    # extract ISO Schematron rules
--useteiversion    # use version data from TEI P5
--parameterize     # create parameterized DTD
--patternprefix=STRING # prefix RELAX NG patterns with STRING
--schema=NAME      # select named schema spec
```

Useful facilities with command-line Roma

1. You can get debugging information during the build process
2. You can choose which <schemaSpec> to process
3. You can change parameters for the documentation creation
4. You can get multiple outputs from one call

Documentation techniques using ODD

It makes a lot of sense to have all your documentation in one place, closely linked to your schema:

1. you can avoid duplication
2. you can avoid contradictions
3. you can generate documented schemas and gain help in editing
4. the chances of you maintaining it are higher if it all uses the same technology
5. you can use validation techniques across the board

Using <specGrp> and <specGrpRef>

Instead of including all your work inside <schemaSpec>, you can put your code into containers of <specGrp> and refer to them inside <schemaSpec>

```
<specGrp xml:id="coremods">
  <moduleRef key="tei"/>
  <moduleRef key="header"/>
  <moduleRef key="core"/>
</specGrp>
<schemaSpec ident="test">
  <specGrpRef target="#coremods"/>
</schemaSpec>
```

Embedded <specGrp> in normal prose

Now we can start adding normal sections of documentation

```
<div>
  <head>Introduction</head>
  <p>In which we meet each other</p>
</div>
<div>
  <head>TEI customization</head>
  <p>We want the main TEI modules with no change</p>
  <specGrp xml:id="coremods">
    <moduleRef key="tei"/>
    <moduleRef key="header"/>
    <moduleRef key="core"/>
  </specGrp>
</div>
<div>
  <head>The schema</head>
  <schemaSpec ident="test">
    <specGrpRef target="#coremods"/>
  </schemaSpec>
</div>
```

Enhancing your prose with <specList>

Sometimes you would like to summarize some elements in the prose description of your encoding:

```
<p>Our main structural elements are <gi>div</gi> and <gi>p</gi>:
<specList>
   <specDesc  key="div"/>
   <specDesc  key="p"/>
  </specList>
</p>
```

1 Intro to my project

Here is where I talk about my project

Our main structural elements are div and p:

< **div** > (text division) contains a subdivision of the front, body, or back of a text.
< **p** > (paragraph) marks paragraphs in prose.

<specList> can do attributes as well

Normally <specList> does not list the attributes of an element

```
<specList>
  <specDesc  key="graphic"/>
</specList>
```

< graphic > indicates the location of an inline graphic, illustration, or figure.

but you can specify which ones to show:

```
<specList>
  <specDesc  key="graphic"  atts="url"/>
</specList>
```

< graphic > indicates the location of an inline graphic, illustration, or figure.

@url (uniform resource locator) A URL which refers to the image itself.

Now for some changes to the documentation

The commonest requirement is to change the description of an element or attribute, to better fit your project, perhaps combined with a limitation on the content model

```
<elementSpec ident="graphic" mode="change">
  <desc>Include a picture of the page; always use PNG or
     JPEG and supply height and width</desc>
  <attList>
    <attDef ident="scale" mode="delete"/>
    <attDef ident="width" mode="change" usage="req"/>
    <attDef ident="height" mode="change" usage="req"/>
    <attDef ident="url" mode="change">
      <desc>Full URL of picture file</desc>
    </attDef>
  </attList>
</elementSpec>
```

Implementing another restriction

What if you wanted to limit the measurements to positive measures in centimetres only? The current pattern is in the macro data.outputMeasurement:

```
        <content>
  <rng:data type="token">
    <rng:param name="pattern">
    [\-+]?\d+(\.\d+)?(%|cm|mm|in|pt|pc|px|em|ex|gd|rem|vw|vh|vm)</rng:param>
  </rng:data>
</content>
```

We could change that to

```
        <content>
  <rng:data type="token">
    <rng:param name="pattern">\d+(cm|mm)</rng:param>
  </rng:data>
</content>
```

Note that you will have to follow the Guidelines carefully to get here, and to understand regexp patterns in W3C Schema.

… and the examples too

Once we have changed the description, we probably want to put in local examples as well:

```
<elementSpec mode="change" ident="graphic">
  <exemplum xml:lang="en">
    <egXML xmlns="http://www.tei-c.org/ns/Examples">
     <graphic ="http://www.example.com/Pages/P1.png" width="4cm" height="10cm" >
    </egXML>
  </exemplum>
</elementSpec>
```

By the way — don't put several <elementSpec mode="change"> stanzas in your ODD for the same element. Combine all the changes into one.

Example gotchas

- Note the use of <egXML> in its own namespace to encompass examples
- All exempla must have indicate their language, using *@xml:lang*
- When you processs an ODD for documentation, only examples from the chosen language are shown, except that
  - if there is no example from your language, the English one(s) are shown
  - examples in language und are always shown
- You cannot selectively remove or change existing examples, you can only replace the whole set for a language
- The TEI Guidelines use *@corresp* on <egXML> to link to a bibliographical entry for the example

egXML and bibl linking

```
<egXML xmlns="http://www.tei-c.org/ns/Examples" corresp="#COPA-eg-1">
<body>
<p>I fully appreciate Gen. Pope's splendid achievements with their
invaluable results; but you must know that Major Generalships in the
Regular Army, are not as plenty as blackberries.</p>
</body>
</egXML>
....
<bibl xml:id="COPA-eg-1">
<author>Lincoln, Abraham</author>. <title level="a">A. Lincoln to
Richard Yates and William Butler</title>, 10 Apr 1862.
In <editor>Fehrenbacher, Don E.</editor>
ed. <title level="m">Lincoln: Speeches and Writings, </title>
<biblScope type="vol">2</biblScope> (1859-1865).
<series>Library of America</series>
<biblScope type="vol">46</biblScope>, <date>1989</date>,
<biblScope type="pp">p.315</biblScope>.</bibl>
```

While we are about it, let's rename elements too

A more controversial technique is to rename TEI elements. It is quite easy:

```
<elementSpec mode="change" ident="div">
  <altIdent>section</altIdent>
</elementSpec>
```

which may make the markup and editing easier for some people (you may like to have all the element names in Latin), but it causes problems with portability.

Writing a transformation script to normalize the XML which uses the new names would not be very hard.

Which leads us to <equiv>

What if we want to go further than map to <section>, perhaps to use the value of the *@type* attribute to decide how to map it.

Example: instead of with a type attribute, use <chapter>, <section>, <subsection> etc

<equiv> example

```
<elementSpec ident="chapter" mode="add">
<!-- ... -->
</elementSpec>
```

define a new element

```
<equiv filter="mymap.xsl" mimeType="text/xsl" name="chapter"/>
```

specify an external routine which will do the mapping

```
<xsl:template name="chapter">
  <div type="chapter">
    <xsl:apply-templates
       select="@*|*|text()|comment()|processing-instruction"/>
  </div>
</xsl:template>
```

write the external routine(s)

create a transformation using the mapping See next slide

Transformation script

```
<XSL:stylesheet version="2.0"
   xpath-default-namespace="http://www.tei-c.org/ns/1.0">
  <XSL:import href="mymap.xsl"/>
  <XSL:template match="*">
    <XSL:copy>
      <XSL:apply-templates
         select="*|@*|processing-instruction()|comment()|text()"/>
    </XSL:copy>
  </XSL:template>
  <XSL:template
     match="text()|comment()|@*|processing-instruction()">
    <XSL:copy/>
  </XSL:template>
  <XSL:template match="chapter">
    <XSL:call-template name="chapter"/>
  </XSL:template>
</XSL:stylesheet>
```

Making a new ODD

Why might you write your own schema in ODD?

- to do a better job of documenting an existing vocabulary
- to define a genuinely new language

Examples:

- W3C ITS: `http://www.w3.org/TR/its/`
- TEI ISO: `http://tei.oucs.ox.ac.uk/TEIISO/`
- Music Encoding Initiative: `http://www2.lib.virginia.edu/innovation/mei/`

Fundamentally the same as a TEI customization

Write a TEI document with a <schemaSpec> and add some elements.

```
<TEI>
<!-- … -->
  <text>
    <body>
      <schemaSpec  ident="mySchema">
        <elementSpec  ident="a">
          <desc>My first element</desc>
          <content>
            <rng:text/>
          </content>
        </elementSpec>
      </schemaSpec>
    </body>
  </text>
</TEI>
```

But remember

- Think about the namespace to use (if any)
- You do not need `mode="add"`
- You do not get att.global (*@xml:id*, *@xml:lang* etc)
- You have to specify a start element (with *@start* attribute
- You can pull in TEI elements using <elementRef>, but watch for the consequences
- You can use ODD as the input to a second stage of customizations if you are careful (in this case you do not need the <schemaSpec>)
- You cannot edit or process these ODDs with Web Roma

A small ODD (part 1)

```
<TEI>
  <teiHeader>
<!-- ... -->
  </teiHeader>
  <text>
    <body>
      <schemaSpec ident="new" start="people" ns="">
        <elementSpec ident="people">
          <desc>Catalogue of people</desc>
          <content>
            <rng:oneOrMore>
              <rng:ref name="person"/>
            </rng:oneOrMore>
          </content>
        </elementSpec>
<!-- next page -->
      </schemaSpec>
    </body>
  </text>
</TEI>
```

A small ODD (part 2)

```
<elementSpec ident="person">
  <desc>A human bean</desc>
  <content>
    <rng:text/>
  </content>
  <attList>
    <attDef ident="sex">
      <valList type="closed">
        <valItem ident="1">
          <gloss>male</gloss>
        </valItem>
        <valItem ident="2">
          <gloss>female</gloss>
        </valItem>
      </valList>
    </attDef>
    <attDef ident="born">
      <datatype>
        <rng:data type="date"/>
      </datatype>
    </attDef>
  </attList>
</elementSpec>
```