One Document Does it all

Lou Burnard and Sebastian Rahtz

TEI

October 2005

This talk gives an overview of the ODD (One Document Does it all) XML documentation system developed as a part of TEI P5, explaining the motivation and development of this system.

Literate programming ODD-style

The TEI Guidelines, its DTD, and its schema fragments, are all produced from a single XML resource containing:

1. Descriptive prose (lots of it)
2. Examples of usage (plenty)
3. Formal declarations for components of the TEI Abstract Model:
   - elements and attributes
   - modules
   - classes and macros
4. We call this resource an ODD (One Document Does it all) although the master source is instantiated as many XML mini-documents.

So what?

The TEI scheme can only be used by customizing it.
Customizations are also expressed in the ODD language
For example:

```
<schemaSpec ident="myTEIlite">
<desc>This is TEI Lite with simplified heads</desc>
  <moduleRef name="teistructure"/>
  <moduleRef name="linking"/>
  <moduleRef name="core"/>
  <moduleRef name="teiheader"/>
  <elementSpec ident="head" mode="change">
    <content><rng:text/></content>
  </elementSpec>
</schemaSpec>
```

produces something like TEI Lite, with a slight change

ODD processors

- We supply a library of XSLT scripts that can generate
  - The book in canonical TEI XML format
  - The book in HTML or PDF
  - RelaxNG, DTD, or W3C schema fragments
- The same library is used by the new customization layer to generate
  - project-specific documentation
  - project-specific schemas
  - translations into other (human) languages
- We use eXist as database for extracting material from the P5 sources

The TEI abstract model

- The TEI abstract model sees a markup scheme (a schema) as consisting of a number of discrete modules, which can be combined more or less as required.
- A schema is made by combining references to modules and optional element over-rides.
- Each element declares the module it belongs to: elements cannot appear in more than one module.
- Each module extends the range of elements and attributes available by adding new members to existing classes of elements, or by defining new classes.

The TEI class system

- Class membership can do two distinct things for an element:
  1. give it some attributes
  2. allow it to join a ‘club’
- Content models reference ‘clubs’ rather than specific elements (wherever possible)
- Content models are named patterns, distinct from element names
- (There are also special named patterns for common content models such as macro.phraseSeq)

Expression of TEI content models

Beyond the class system, TEI elements have to be defined. How?

1. continue (as in P4) to use ‘raw’ XML DTD language
2. maintain in DTD language but transform to some other schema language at the point of delivery
3. transform to some other schema language for maintenance and delivery
4. invent an entirely new abstract language for later transformation to some schema language

We chose a combination of 3 and 4 — revise our abstract language to use RelaxNG for content modelling (only).

Why that combination?

- Expressing constraints in XML language is too attractive to forego
- We knew we would want namespaces sooner rather than later
- A clamour for better datatyping
- The schema languages are so good, it is silly to reinvent them
- But we like our class system and literate programming

DTD vs Relax NG vs W3C Schema

- DTDs are not XML, and need specialist software
- W3C schema is not consistently implemented, is poorly documented, and looks over-complex
- Relax NG on the other hand...
  - uncluttered design
  - good documentation
  - multiple open source 100%-complete implementations
  - ISO standard
  - useful features for multipurpose structural validation
  - Compelling leadership (can James Clark do wrong?)

No contest. . .

What does an ODD look like?


```
<elementSpec module="spoken" ident="pause">
 <classes>
   <memberOf key="model.divPart.spoken"/>
   <memberOf key="att.timed"/>
   <memberOf key="att.typed"/>
 </classes>
 <content>
   <rng:empty xmlns:rng="\protect .\kern \fontdimen 3\fo
 </content>
  <attList>
    <attDef ident="who" usage="opt">
    <datatype>
        <rng:ref name="data.pointer"/>
    </datatype>
      <valDesc>A unique identifier</valDesc>
      <desc>supplies the identifier of the
      person or group pausing.
      Its value is the identifier of a <gi>person</gi>
      or <gi>persGrp</gi> element in the TEI header.
      </desc>
```

... from which we generate


```
element pause { pause.content, pause.attributes }
pause.content = empty
pause.attributes =
  att.global.attributes,
  att.timed.attributes,
  att.typed.attributes,
  att.ascribed.attributes,
  [ a:defaultValue = "pause" ] attribute TEIform { text
model.divPart.spoken |= pause
att.timed |= pause
att.typed |= pause
att.ascribed |= pause
```

.. which translates to


```
<!ELEMENT %n.pause; %om.RR; EMPTY>
<!ATTLIST %n.pause;
 %att.global.attributes;
 %att.timed.attributes;
 %att.typed.attributes;
 %att.ascribed.attributes;
 TEIform CDATA 'pause'  >
<!ENTITY % model.divPart.spoken
 "%x.model.divPart.spoken; %n.event; | %n.kinesic;
 | %n.pause; | %n.shift; | %n.u;
 | %n.vocal; | %n.writing;">
```

... and, indeed, to

One Document Does it all

Lou Burnard and Sebastian Rahtz

Text Encoding Initiative

<pause>

pause	a pause either between or within utterances.
Class	model.divPart.spoken att.timed att.typed att.asc

Declaration	element pause
	{
	att.global.attributes,
	att.timed.attributes,
	att.typed.attributes,
	att.ascribed.attributes,
	empty
	}

Attributes	Global attributes and those inherited from [att.t
Example	<pause dur="PT42S" type="pregnant"/>

Generation of alternate outputs

1. Relax NG schema fragments are generated by an XSLT transform
2. ... and progressively flattened and simplified by a further set of XSLT transforms
3. DTDs, compact Relax NG, and W3C Schema are all generated using James Clark's trang (but not necessarily from the same inputs)

Vocabularies like MathML and SVG inclusion are managed by simply `<include>`ing the relevant RelaxNG grammars, each in their own namespace.

Customizing the TEI

The TEI has over 20 modules. A working project will:

- Choose the modules they need
- Probably narrow the set of elements within a module
- Probably add local datatype constraints
- Possibly add new elements
- Possibly localize the names of elements

We can do all that in ODD


```
<schema>
<moduleRef name="tei"/>
<moduleRef name="header""/>
<moduleRef name="textstructure"/>
<moduleref name="linking"/>
</schema>
```

From which we can generate...


```
<grammar ns="http://www.tei-c.org/P5/"
 xmlns="http://relaxng.org/ns/structure/1.0"
 datatypeLibrary=
   "http://www.w3.org/2001/XMLSchema-datatypes">
<include href="Schema/tei.rng"/>
<include href="Schema/header.rng"/>
<include href="Schema/textstructure.rng"/>
<include href="Schema/linking.rng"/>
</grammar>
```

More interestingly..


```
<schema>
<moduleRef name="teiheader"/>
<moduleref name="verse"/>
<!-- add a new element -->
<elementSpec ident="soundClip">
<classes memberOf="tei.data"/>
 <attList>
  <attDef ident="location">
  <datatype><rng:ref name="data.pointer"/></datatype>
  <valDesc>A location path</valDesc>
  <desc>supplies the location of the clip</desc>
  </attDef>
 </attList>
 <desc>includes an audio object in a document.</desc>
</elementSpec>
<!-- change an existing element -->
<elementSpec ident="head" mode="change">
<content><rng:text/></content>
</elementSpec>
</schema>
```

Uniformity of description

- modules, elements, attributes, value-lists are treated uniformly
- each has an identifier, a gloss, a description, and one or more equivalents
- each can be added, changed, replaced, deleted within a given context
- for example, membership in the att.type class gives you a generic TYPE, which can be over-riden for specific class members

Overriding a value-list


```
<elementDecl ident="list" module="core">
<classes>
 <memberOf key="att.typed"/>
</classes>
<!--... -->
<attDef ident="type" mode="replace">
<valList>
<valItem ident="ordered">Items are ordered</valItem>
<valItem ident="bulleted">Items are bulleted</valItem>
<valItem ident="frabjous">Items are frabjous</valItem>
</valList>
</attDef>
</elementDecl>
```

... not as easy as it looks (lazy evaluation rules)

Our gesture towards ontological mapping

The `<equiv>` element supplies a URI which identifies an equivalent concept (*not* a name) in some externally-defined ontology, e.g.

- ISO data category registry
- CIDOC conceptual reference model
- Wordnet

Using other vocabularies

- Namespaces help with the obvious cases (e.g. mathML, SVG...)
- But they don’t help where there is overlap (e.g. HEML)
- And they enforce an ‘Us and Them’ mentality
- Can we do better?