Oxford University Computing Services

http://www.oucs.ox.ac.uk/

TEI as an XML production DTD

Lou Burnard, Sebastian Rahtz

TEI meets XML Authoring: the Summary

1. What is a DTD and how do you choose one?
2. The TEI architecture
3. A TEI application for authoring
4. Typesetting TEI documents, with some remarks on XSL Formatting Objects







What is a DTD and how do you choose one?

- The XML concept is dangerously powerful:
  - SGML (and XML) elements are light in semantics
  - one man's <p> is another's <para> (or is it?)
  - the appearance of interchangeability may be worse than its absence
- To get the best out of XML, you need two kinds of DTD:
  - document type **declaration**: elements, attributes, entities, notations (syntactic constraints)
  - document type **definition**: usage and meaning constraints on the foregoing
- Published specifications for SGML DTDs usually combine the two, hence they lack modularity

Some answers

- Namespaces do not provide an answer (though at least they remind us the problem exists)
- Rolling your own DTD
  - ... starting from scratch
  - ... by combining snippets, preferably from an existing conceptual framework (aka **architecture**)
- DTD customization
  - **definitions** should be meaningful within a given user community
  - **declarations** should be appropriate to a given set of applications
- The TEI provides a good candidate architecture

The T E what?

- Originally, a research project within the humanities
  - Sponsored by leading professional associations
  - Funded 1990-1994 by US NEH, EU LE Programme et al
- Major influences
  - digital libraries and text collections
  - language corpora
  - scholarly datasets
- International consortium established June 1999 (see `http://www.tei-c.org`)

Goals of the TEI

- better interchange and integration of scholarly data
- support for all texts, in all languages, from all periods
- guidance for the perplexed: **what** to encode — hence, a user-driven codification of existing best practice
- assistance for the specialist: **how** to encode — hence, a loose framework into which unpredictable extensions can be fitted

These apparently incompatible goals result in a highly flexible, modular, environment for DTD customization.

TEI Deliverables

- A set of recommendations for text encoding, covering both generic text structures and some highly specific areas based on (but not limited by) existing practice
- A very large collection of element **definitions** combined into a very loose document type **declaration**
- A mechanism for creating multiple views (DTDs) of the foregoing
- *One* such view and associated tutorial: TEI Lite (`http://www.hcu.ox.ac.uk/TEI/Lite/`)

for the full picture see `http://www.hcu.ox.ac.uk/TEI/Guidelines/`

How many DTDs do we need?

- one (the Corporate or WKWBFY approach)
- none (the Anarchic or NWEUMP approach)
- as many as it takes (the Mixed Economy or XML approach)

or a single main DTD with many faces (a British DTD)

The Chicago Pizza Model

A useful metaphor for expressing modularity

Now implemented at http://www.hcu.ox.ac.uk/TEI/pizza.html

```
<!ENTITY % base "deepDish | thinCrust | stuffed" >
<!ENTITY % topping "pepperoni | mushrooms |
sausage | anchovies...  " >
<!ELEMENT pizza ( %base;, tomatoSauce,
cheese, (%topping;)*) >
```







To build a view of the TEI dtd, take...

- the core tagsets
- the base of your choice
- the toppings of your choice
- (optionally) a reference to your extensions

```
<!DOCTYPE TEI.2 SYSTEM "tei2.dtd" [
<!ENTITY % tei.prose "INCLUDE" >
<!ENTITY % tei.analysis "INCLUDE" >
<!ENTITY % tei.extensions.ent SYSTEM "myMods.ent" >
<!ENTITY % tei.extensions.dtd SYSTEM "myMods.dtd" >
]>
<tei.2>.....</tei.2>
```

... in your extensions you can...

- rename elements

```
<!ENTITY % n.p para >
```

- undefine elements

```
<!ENTITY % seg IGNORE>
```

- supply additional (or replacement) declarations

```
<!ENTITY % seg IGNORE>
<!ELEMENT %n.seg (#PCDATA)>
```

- supply entirely new elements and embed them in the architecture

```
<!ENTITY % x.phrase 'blort|'>
<!ELEMENT blort (#PCDATA)>
```







```
<!ATTLIST blort %a.global;
farble (foo|bar|baz) "baz">
```

and cook thoroughly

- ‘compile’ the dtd to remove all parameterization
- essential for XML use
- better project management
- (did I mention the Pizza Chef web site? `http://www.hcu.ox.ac.uk/TEI/pizza.html`)
- don’t forget the documentation!







How does this work ?

To understand this, you need to remember the following concepts from SGML for Dummies:

- parameter entities
- marked sections
- effect of multiple declarations
- Main DTD consists of marked sections, each containing declarations for one tagset

```
<![ %TEI.tagset [
<!-* declarations for tagset here *->
] ]>
<![ %TEI.another [
<!-* declarations for another here *->
] ]>
```







- At its start, the main DTD sets defaults for all %TEI.xxx entities

```
<!ENTITY % TEI.tagset "IGNORE">
<!ENTITY % TEI.another "IGNORE" >
```

How does this work? (contd)

- Similarly, each element and attlist declaration in a tagset is enclosed by a marked section

```
<![ %element [
<!ELEMENT %n.element - - (#PCDATA)>
<!ATTLIST %n.element %a.global; >
] ]>
```

- Element names(GIs) are always referenced indirectly; entity declarations for the element names are embedded first

```
<!ENTITY % n.element "element">
```

- Entity declarations for the element definitions precede each marked section

```
<!ENTITY % element "INCLUDE">
```

Element Classes

- Most TEI elements are assigned to one or more
  - **element classes**, identifying their syntactic properties, or
  - **attribute classes**, identifying their attributes
- This provides a (relatively) simple way of
  - documenting and understanding the DTD
  - parameterizing content models
  - facilitating customization
- An alternative way of doing architectural forms

Some TEI classes

- model classes:
  - divn: structural elements like divisions (, <div1>, <div2>...)
  - divtop: elements which can appear at the start of a divn element (<head>, <epigraph>, <byLine>...)
  - chunk: paragraph-like elements (<sp><p><lg>...)
  - phrase: elements which appear within chunks (<hi>, <foreign>, <date> ...)
- attribute classes:
  - global: attributes which are available to every element (n, lang, id, TEIform)
  - linking: attributes for elements which have linking semantics (targType, targOrder, evaluate)

Customization

- Simplest kind of customization involves redefinition of existing elements (removal followed by addition)

```
<!-* in TEI.extensions.ent *->
<!ENTITY % p "IGNORE">
<!-* in TEI.extensions.dtd *->
<!ELEMENT %n.p - - (#PCDATA)>
```

  Note that class membership is unaffected

- A slightly more complex kind involves adding a new element to an existing class

How does this work?(contd)

- Each model class is defined as a parameter entity, containing a reference to an initially null extension class, and a list of members

```
<!ENTITY % x.class "" >
<!ENTITY % m.class "%x.class; name1|name2|name3 ..." >
<!ELEMENT % n.element - - (%m.class;+)>
```

- To add a new member to a class, we redefine the extension class:

```
<!ENTITY % x.class "myChunk|myOther|">
```

Syntactic sugar

- Occam's razor has been liberally wielded in the TEI, resulting in much use of the `<tag type="xyz">` idiom
- In `<xyz TEIform="tag">`, the TEIform attribute can be used to indicate that this is syntactically equivalent to `<tag type="xyz">`







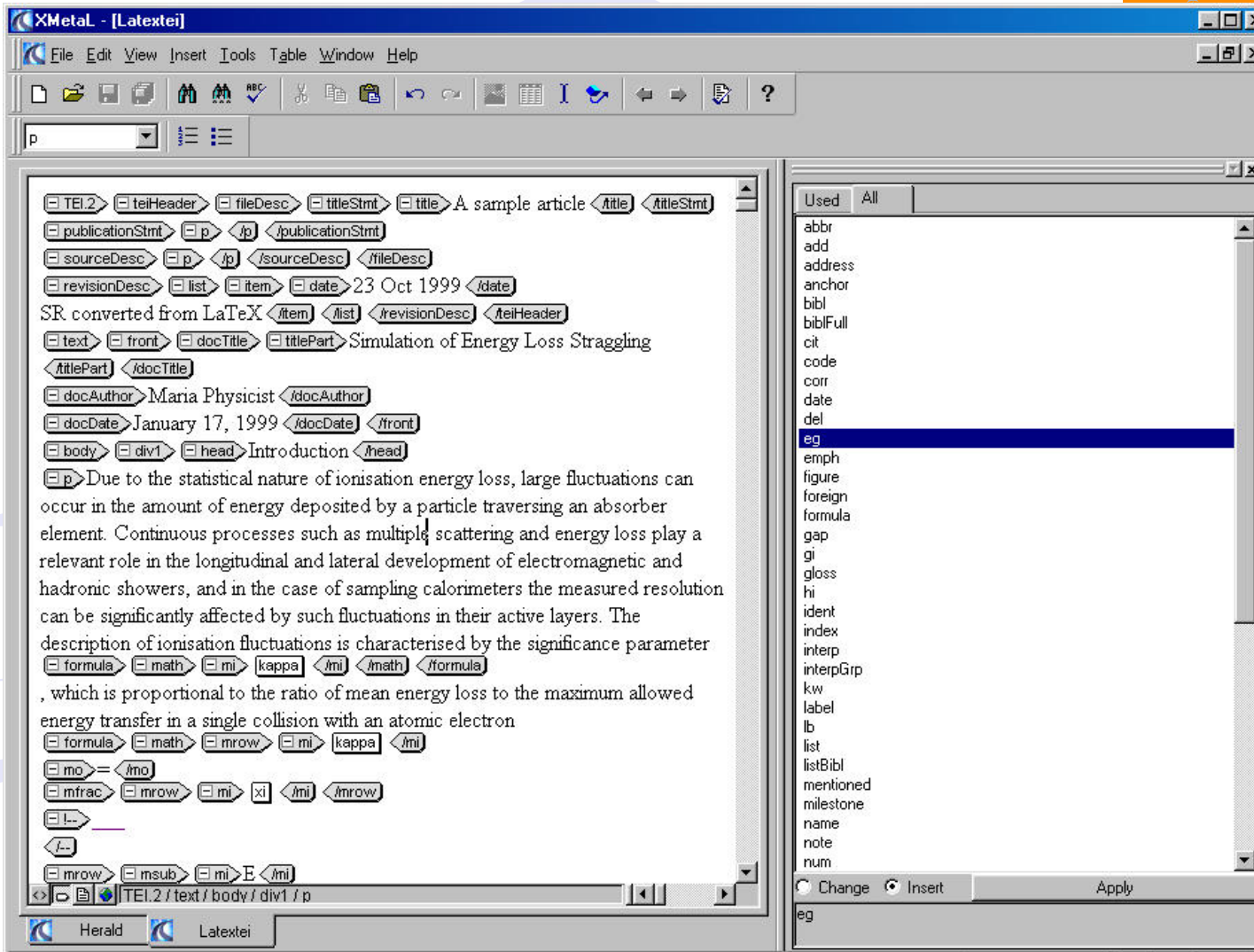
Using the TEI as an authoring DTD

The TEI emphasizes description rather than creation of text. Consequently...

- We end up using `type=""` attributes a great deal, eg

```
<list type='steps'>
<item>Log in to the lecture room network
with your course username and password.</item>
<item>Start Netscape by double clicking on its icon.</item>
</list>
```

- We offer the author too much choice in "what element can I insert here" editors:


XMetaL - [Latextei]
File Edit View Insert Tools Table Window Help
p
TEI.2 teiHeader fileDesc titleStmt title A sample article /title /titleStmt
publicationStmt p /p /publicationStmt
sourceDesc p /p /sourceDesc /fileDesc
revisionDesc list item date 23 Oct 1999 /date
SR converted from LaTeX /item /list /revisionDesc /teiHeader
text front docTitle titlePart Simulation of Energy Loss Straggling
/titlePart /docTitle
docAuthor Maria Physicist /docAuthor
docDate January 17, 1999 /docDate /front
body div1 head Introduction /head
p Due to the statistical nature of ionisation energy loss, large fluctuations can occur in the amount of energy deposited by a particle traversing an absorber element. Continuous processes such as multiple scattering and energy loss play a relevant role in the longitudinal and lateral development of electromagnetic and hadronic showers, and in the case of sampling calorimeters the measured resolution can be significantly affected by such fluctuations in their active layers. The description of ionisation fluctuations is characterised by the significance parameter
formula math mi kappa /mi /math /formula
, which is proportional to the ratio of mean energy loss to the maximum allowed energy transfer in a single collision with an atomic electron
formula math mrow mi kappa /mi
mo = /mo
mfrac mrow mi xi /mi /mrow
!--
/--
mrow msub mi E /mi
TEI.2 / text / body / div1 / p
Herald
Latextei
Used All
abbr
add
address
anchor
bibl
biblFull
cit
code
corr
date
del
eg
emph
figure
foreign
formula
gap
gi
gloss
hi
ident
index
interp
interpGrp
kw
label
lb
list
listBibl
mentioned
milestone
name
note
num
Change Insert Apply
eg

- We lack elements for every possible area

A suitable exercise for the TEI customization features...







Where are extensions needed for authoring?

Tables the TEI's minimalist model sweeps all the complexity into an already over-loaded `rend` attribute

Maths and other scientific notations (TEI assumes you will use an external notation)

Algorithmic graphics (The Death Of The Embedded Graphic)

Front matter for documents other than early printed books, e.g. STM articles

Office documents and other things born digital

Two office documents

```
<!  - a memorandum marked up in TEI ->
<!ENTITY u-shortsight SYSTEM "http://www.ourcompany.com/policy/"
NDATA URL> ..
<text>
<front>
<opener type="from"><name>Ty Coon</name></opener>
<opener type="to"><name>Ev Angelist</name></opener>
<date>Today</date>
</front>
<body><div>
<p>Re your memorandum of <date>July 21st</date>, I
think that the chance of us switching to XML in
this company is minimal.  See <xptr doc="u-shortsight"/>.
</p>
</div></body>
</text>
```


```
<!- a business letter marked up in TEI->
<text><body>
<p><address><!- ...  ->
<salute>Dear Ty</salute>
<p>Do you realize that the word-processor stored
your memo to me marked up in XML?</p>
<signature>Ev</signature>
</body></text>
```

Possible practical answers

We may need to do some or all of:

- Define extensive additional tagsets, possibly containing much syntactic sugar, for new domains
- Suck in external DTDs, like MathML, SVG, and XHTML tables and forms (but we will need to address name clashes and universal namespace support may be a while coming)
- Use all and only those parts of the TEI we need to avoid tag overload for authors
- Add convenience attributes (eg to bypass purist XLink markup for URLs)

The author vs the editor?

Hold on: do we need to use the same DTD for authoring, for archive, for editing, for production? The TEI philosophy allows us:

1. Develop sample documents for a new domain using generic tools like `<div>` and `type` attributes
2. Generate a private *authoring* DTD which uses domain-specific language:

```
<!  - memorandum marked up in TEIMEMO ->
<memo>
<front>
<from_opener>Ty Coon</from_opener>
<to_opener>Ev Angelist</to_opener>
<date>Today</date>
</front>
<body>
<div>
```


```
<p>Re your memorandum of <date>July 21st</date>, I think that
the chance of us switching to XML in this company is mini-
mal.
See <xptr url="http://www.ourcompany.com/policy/"/>.
</p>
</div>
</body>
</text>
```

3. Translate domain documents back to generic TEI for interchange
4. Editing can take place in the domain language or the general one
5. Publication can use a third DTD, the domain language or the general one

We may never publish our internal DTD, or we may propose it for consideration as a new TEI tagset.

Typesetting TEI documents

What are the practical experiences with publishing TEI documents?

- The next generation browsers display our XML natively
- For current browsers, transformation of TEI XML to HTML is a well-understood art
- For print, we might consider
  - Transform to HTML and trust Mozilla's formatter
  - Read into eg 3B2 and write style sheets
  - Convert to proprietary codes using eg Omnimark
  - Process with standardised formatting language (XSL FO)







XSL FO — just say no!

- The XSL FO page model is inherited from DSSSL, and is unproven for production-quality print formatting
- The XSL specification is unfinished and incomplete.
- You cannot easily tweak your formatter's behaviour with this system
- A layout language decoupled from the page makeup cannot deal with some design specifications ("the table caption is centered unless, when typeset, its length exceeds the page width; in this case it is set ragged right")
- Will the big DTP players kill XSL FO to preserve their systems?







XSL FO — reasons to be cheerful

- We have several free systems to experiment with (eg FOP, PassiveTeX), and commercial implementations being demonstrated (eg RenderX)
- We need next-generation layout systems anyway, capable of dealing with full Unicode and all variations on left-right/up-down typesetting
- XSL FO does not threaten page formatters and line breakers — it gives them a reason to survive
- Things can only get better!

Conclusions

- The TEI is stable, rigorous, and well-documented
- The TEI is also flexible, customizable, and extensible *in documented ways*
- The TEI remains SGML, but XML DTDs can be generated trivially
- The architectural approach offers the best compromise for practical work.