Digital Texts with XML and the TEI
Part 2: Taming the TEI Tiger

Lou Burnard

February 2003

Today's topics

- The TEI and its architecture
- The TEI Header
- Working with XML editors
- Working with the Pizza Chef

In today's exercises, you'll create a TEI header with emacs, and also build your very own pizza.

XML : a licence for ill?

XML allows you to make up your own tags, and doesn't require a DTD...

- 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
- But XML is still too good to ignore
  - mainstream software development
  - proliferation of tools
  - the future of the web

DTD : what does it mean?

- 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

- Rolling your own DTD
  - ... starting from scratch
  - ... by combining snippets, preferably from an existing conceptual framework (aka **architecture**)
- customizing someone else's DTD
  - **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

Designing a DTD for the TEI

- How can a single mark-up scheme handle a large variety of requirements ?
  - all texts are alike
  - every text is different
- Learn from the database designers
  - one construct, many views
  - each view a selection from the whole

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.tei-c.org.uk/pizza.html

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

To build a TEI pizza, 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>
```

The core tagsets

- detailed metadata provision: the TEI Header
- tags for a large set of common textual requirements:
  - paragraphs
  - highlighted phrases
  - names, dates, number, abbreviations...
  - editorial tags
  - notes, cross-references, bibliography
  - verse and drama

The base tagsets

- define basic high-level structure of document
- one must be chosen from:
  - prose, verse, or drama
  - transcribed speech
  - dictionaries
  - terminology
- or combine two or more using either of
  - the general base (anything anywhere)
  - the mixed base (homogenous divisions)

TEI additional tagsets

- sets of elements for specialised application areas
- can be mixed and matched ad lib
- currently provided:
  - linking and alignment; analysis; feature structures;
  - certainty; physical transcription; textual criticism,
  - names and dates; graphs and trees; figures and tables;
  - language corpora....
- in preparation...
  - manuscript description

The TEI Header

- What's in it?
- How do you use it?

Exercise 1

- Using emacs to build a TEI document

What do we mean by TEI extensions?

Two files (one containing entity declarations, the other containing element and attribute declarations) can be mixed in to any TEI dtd. They allow you to...

- rename elements

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

- undeclare elements

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

The pizzaChef gives you a list of all the elements available from your chosen tagsets, and generates extension files for you

... you can also ...

- 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">
```

...finally, the pizza is cooked

- The **carthage** program removes
  - parameterization in the DTD
  - unreferenced or inaccessible elements
- It is thus
  - essential for creating an XML dtd
  - useful in project management

What can go wrong?

- extensions must use SGML syntax
- beware of zombie elements
- beware of over zealous pruning
- remember that some TEI rules are not enforced (or enforceable) by the DTD

You have to know what's on the menu before you can choose from it

How does it work?

- Use of parameterised *marked sections* within main DTD
  - declarations making up each tagset are enclosed by an IGNORE marked section
  - declarations for each element are enclosed by an INCLUDE marked section
  - their status can be over-ridden in the DTD subset
- Use of parameterised class system

An example

In the DTD subset we write:

```
 <!ENTITY % TEI.prose "INCLUDE">
<!ENTITY % biblStruct "IGNORE">
```

The prose tagset within the TEI dtd looks like this:

```
<!ENTITY % TEI.prose "IGNORE">
<![ %TEI.prose [

 .... lots of other declarations ...

<!ENTITY % biblStruct "INCLUDE">
<![ %biblStruct [
  <!ELEMENT biblStruct .... >
  <!ATTLIST biblStruct .... >
] ]>
  ... yet more declarations ....

] ]>
```

In SGML (and XML) the DTD subset declaration always wins...

Element Classes

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

Some TEI model classes

- `divn`: structural elements like divisions (`<div>`, `<div>`, `<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>` ...)

TEI 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

Class mobility

- 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|">
```

The TEIFORM attribute

Two main usages...

- protect applications from the effect of element renaming

```
<titolo TEIform="title">...</titolo>
```

- protect applications from the effect of syntactic sugar

```
<tag type="xyz">
```

can be rewritten as

```
<xyz TEIform="tag">
```

A case study: the Lampeter corpus

See http://www.tu-chemnitz.de/phil/english/real/lampeter/lamphome.htm (or look in the Oxford Text Archive)

- Fairly typical requirements for language corpora
  - light presentational tagging
  - structural markup for access
  - demographic information about text production
  - small number of tags to ease data capture and validation
- Implementation
  - tagsets: prose base, and tags from four additional sets
  - some extensions, many exclusions

The Lampeter corpus DTD subset

```
<!DOCTYPE teiCorpus.2 SYSTEM "tei2.dtd"
[<!ENTITY % TEI.prose "INCLUDE">
<!ENTITY % TEI.corpus "INCLUDE">
<!ENTITY % TEI.figures "INCLUDE">
<!ENTITY % TEI.transcr "INCLUDE">
<!ENTITY % TEI.extensions.ent
SYSTEM "lampext.ent">
<!ENTITY % TEI.extensions.dtd
SYSTEM "lampext.dtd">
<!-- more declarations here -->
]>
```

The Lampeter corpus extensions.ent

```
<!ENTITY % analytic 'IGNORE' >
<!ENTITY % biblStruct 'IGNORE' >
<!-- hic desunt multa -->
<!ENTITY % supplied 'IGNORE' >
<!ENTITY % x.phrase
      "it|ro|sc|su|bo|go|">
<!ENTITY % x.biblPart
      "printer|pubFormat|bookSeller|">
<!ENTITY % x.demographic
      "socecstatusPat|biogNote|">
```

The Lampeter corpus extensions.dtd

```
<!ELEMENT it - - (%phrase.seq)>
<!ATTLIST it %a.global; >
<!-- Similar definitions for
     ro sc su bo go
     persName printer pubFormat
     bookSeller biogNote socecstatusPat
-->
```

Exercise 2

- http://www.tei-c.org.uk/pizza.html

Using the TEI for authoring

A DTD for authoring should be

- prescriptive rather than descriptive
- closely tied to current authoring practice
- *very* easy to use

This suggests that we need

- contentfull tagging
- only the tags we need
- all the tags we need

Contentfull tagging

Which is better for the authorer:

```
<list type='steps'>
<item n="1">Log in to the network
with your course username and
password.</item>
<item n="2">Start Netscape by
double-clicking on its icon.</item>
<!-- ... -->
</list>
```

or

```
<stepList>
<step n="1">Log in to the network
with your course username and
password.</item>
<step n="2">Start Netscape by
double-clicking on its icon.</item>
<!-- ... -->
</stepList>
```

?

All and only

Unmodified TEI offers authorers too many choices:

- four different types of bibliographic citation
- three (or four) different tags for proper names
- an indigestably rich choice of text editing tags

At the same time, unmodified TEI lacks

- detailed table model
- detailed tags for mathematical and other formulae
- front matter for modern publications
- tags for multimedia objects

all this can be addressed by TEI customization

Why bother?

- The TEI is a well-known reference point
- Using the TEI enables
  - sharing of data and resources
  - shared modular software development
  - lower learning curve and reduced training costs
- The TEI is stable, rigorous, and well-documented
- The TEI is also flexible, customizable, and extensible *in documented ways*
- The architectural approach offers the best compromise for practical work.