Digital Texts with XML and the TEI

Lou Burnard

February 2003

Questions we will try to answer on this course

1. What is text mark-up for?
2. What is XML?
3. How is the TEI system organized and what is it for?
4. How do I customize the TEI system to create digital texts the way I want them?
5. How do I do cool stuff with my digital texts?

Questions we will (probably) not try to answer on this course

- Who can I get to do all this for me?
- How would I do all this using Word?
- How would I do all this using a database?
- How would I do all this using some other XML scheme?
- What is a digital text for anyway?

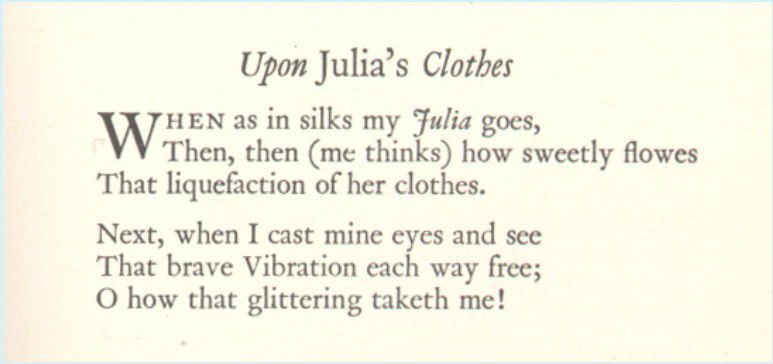
Today's topics

- What do we mean by a digital text?
- What do we mean by markup?
- What is the TEI?

We will try to provide answers to these questions. You will also explore them, both theoretically and practically. And you will see the first of many XML editors...

What's in a text?

Is this:



Upon Julia's *Clothes*

WHEN as in silks my *Julia* goes,
Then, then (me thinks) how sweetly flowes
That liquefaction of her clothes.

Next, when I cast mine eyes and see
That brave Vibration each way free;
O how that glittering taketh me!

the same as this:

Upon Julia's Clothes

When as in silks my Julia goes,
Then, then (me thinks) how sweetly flowes
That liquefaction of her clothes.

Next, when I cast mine eyes, and see
That brave Vibration each way free;
O how that glittering taketh me!

The ontology of text

Where is the text?

- in the shape of letters and their layout?
- in the original from which this copy derives?
- in the ideas it brings forth? in their format, or their intentions?

Texts are abstractions conjured up by readers.

Markup encodes those abstractions.

Encoding of texts

- Texts are more then sequences of encoded glyphs
  - They have *structure* and *content*
  - They also have multiple *readings*
- Encoding, or markup, is a way of making these things explicit
- Only that which is explicit can be reliably processed

Styles of markup

- In the beginning there was *procedural* markup

```
RED INK ON; print balance; RED INK OFF
```

- which being generalised became *descriptive* markup

```
<balance type='overdrawn'>some numbers</balance>
```

- also known as *encoding* or *annotation*

descriptive markup allows for re-use of data

Some more definitions

- Markup makes explicit the distinctions we want to make when processing a string of bytes
- Markup is a way of naming and characterizing the parts of a text in a formalized way
- It's (usually) more useful to markup what things *are* than what they *look like*

What does markup capture?

Compare

```
<head>Upon Julia’s Clothes</head>
<lg><l>Whenas in silks my <hi>Julia&lt/hi> goes,</l>
<l>Then, then (me thinks) how sweetly flowes</l>
<l>That liquefaction of her clothes.</l>
</lg>
```

and

```
<s n="1" role="head">
     <w type="pp">Upon</w>
     <w type="np">Julia</w><w type="pos">’s </w>
     <w type="nn2">Clothes</w>
</s>
<s n="2" role="line">
     <w type="adv">Whenas</w>
     <w type="pp">in</w>
     <w type="nn2">silks</w>
     ...
</s>
```

What's the point of markup?

- To make explicit (to a machine) what is implicit (to a person)
- To add value by supplying multiple annotations
- To facilitate re-use of the same material
  - in different formats
  - in different contexts
  - for different users

First exercise

Imagine you are going to markup several thousand pages like this.

- Which features are you going to markup?
- Why are you choosing to markup this feature?
- How reliably and consistently can you do this?

Now, imagine your budget has been halved. Repeat the exercise!

Some alphabet soup

SGML Standard Generalized Markup Language

HTML Hypertext Markup Language

XML eXtensible Markup Language

DTD Document Type Definition (or Declaration)

CSS Cascading Style Sheet

XSLT eXtensible Stylesheet Language - Transformations

Oh, and then there's also

TEI Text Encoding Initiative

XML: what it is and why you should care

- XML is **structured data** represented as strings of text
- XML looks like HTML, except that:-
  - XML is **extensible**
  - XML must be **well-formed**
  - XML can be **validated**
- XML is application-, platform-, and vendor-independent
- XML empowers the **content provider** and facilitates data integration

An example XML document

```
<?xml version="1.0" encoding="utf-8" ?>
<cookBook>

  <recipe n="1">
   <head>Nail Soup</head>
   <ingredientList>  .... </ingredientList>
   <procedure>  ....   </procedure>
  </recipe>

  <recipe n="2">
  <!-- contents of second recipe here -->
  </recipe>

<!-- hic desunt multa -->

</cookBook>
```

XML terminology

An XML document contains:-

- elements, possibly bearing attributes
- processing instructions
- entity references
- CDATA marked sections
- IGNORE/INCLUDE marked sections

An XML document must be *well-formed* and may be*valid*

XML is an international standard

- XML requires use of ISO 10646
  - a 31 bit character reportoire including most human writing systems
  - encoded as UTF8 or UTF16
- other encodings may be specified at the document level
- language may be specified at the element level using **xml:lang**

The rules of the XML Game

- An XML document represents a (kind of) *tree*
- It has a single *root* and many nodes
- Each node can be
  - a subtree
  - a single *element* (possibly bearing some *attributes*)
  - a string of *character data*
- Each element has a type or *generic identifier*
- Attribute names are predefined for a given element; values can also be constrained

Representing an XML tree

- An XML document is encoded as a linear string of characters
- It begins with a special *processing instruction*
- Element occurrences are marked by *start-* and *end-tags*
- The characters < and & are Magic and must always be "escaped"
- *Comments* are delimited by <!-- and -->
- *CDATA sections* are delimited by <![CDATA[and]]>
- Attribute name/value pairs are supplied on the start-tag and may be given in any order
- Entity references are delimited by & and ;

XML syntax: the small print

What does it mean to be *well-formed*?

1. there is a single root node containing the whole of an XML document
2. each subtree is properly nested within the root node
3. names are always case sensitive
4. start-tags and end-tags are always mandatory (except that a combined start-and-end tag may be used for empty nodes)
5. attribute values are always quoted

Splot the mistake

```
<greeting>Hello world!</greeting>
<greeting>Hello world!</Greeting>

<greeting><grunt>Ho</grunt> world!</greeting>
<grunt>Ho <greeting>world!</greeting></grunt>
<greeting><grunt>Ho world!</greeting></grunt>

<grunt type=loud>Ho</grunt>
<grunt type="loud"></grunt>

<grunt type= "loud">
<grunt type ="loud"/>
```

Defining the rules

A **valid** XML document will reference a *document type declaration* (DTD) :

```
<!DOCTYPE div SYSTEM "verse-div.dtd">
```

```
<!DOCTYPE TEI.2 SYSTEM "tei2.dtd">
```

A DTD specifies:

- name of the root element
- names for all elements used
- names and default values for their attributes
- rules about how elements can nest
- names for re-usable pieces of data (entities)
- and a few other things

n.b. A DTD does *not* specify anything about what elements "mean"

The DTD Subset

- As well as referencing a DTD, an XML document type declaration can include some extras known as the *DTD subset*

```
<!DOCTYPE TEI.2 SYSTEM "tei2.dtd" [
        <!-- additional declarations here -->
]>
```

- Declarations in the subset are processed before those in the DTD
- This gives us the ability to modify a DTD... see later!

An example DTD file

```
<!ELEMENT div (head*, (lg|l|milestone)*, signed*)>
<!ATTLIST div n  CDATA #IMPLIED
              type (verse|prose|drama) "prose">
<!ELEMENT lg (head?, (l|milestone)+)  >
<!ATTLIST lg n  CDATA #IMPLIED>
<!ELEMENT head (#PCDATA) >
<!ATTLIST head type  CDATA #IMPLIED>
<!ELEMENT l  (#PCDATA|emph|q|milestone)* >
<!ATTLIST milestone n CDATA #IMPLIED
                  unit CDATA #REQUIRED >
<!ELEMENT milestone EMPTY >
<!ELEMENT q (#PCDATA|milestone)*>
<!ATTLIST q complete (Y|N) "Y" >
<!ELEMENT signed (#PCDATA)>
<!ENTITY mdash "&#x2014;">
```

This might be invoked using a DOCTYPE statement like the following

```
<!DOCTYPE div SYSTEM "filename.dtd" []>
```

DTDs or schemas?

- DTDs are written in a special and not very expressive DTD language
- Although currently widespread, they are likely to be replaced for practical use by one or other of the new *schema languages*
  - W3C schema
  - Relax NG
- ... which are more expressive, offer better validation facilities for applications, and are expressed in XML

Defining an element

An element declaration takes the form

```
<!ELEMENT name contentModel >
```

name is the name of the element

contentModel defines valid content for the element

The *content* of an element can be:

- #PCDATA
- EMPTY
- other elements
- *mixed* content combines PCDATA and other elements

Content models

Within a content model:

- *sequence* is indicated by comma
- *alternation* is indicated by |
- *grouping* is indicated by parentheses

Occurrence indicators:

[nothing]	once	?	optionally once
+	one or more times	*	zero or more times

If #PCDATA appears in a content model...

- it can only appear once
- it must appear **first**
- if in an alternation, only the * occurrence indicator is allowed

For example...

```
<!ELEMENT a (b+) >
<!ELEMENT b EMPTY>
<!ELEMENT c (#PCDATA)>
<!ELEMENT a (b,c) >
<!ELEMENT a (b|c)* >
<!ELEMENT a (#PCDATA|b|c)* >
<!ELEMENT a (b,  (c|d)*) >
<!ELEMENT a (b?, (c|d)+) >
<!ELEMENT a (b?, (c+|d+)) >
```

Defining an attribute list

An attribute list declaration takes the form

```
<!ATTLIST name attributelist >
```

name is the name of the element bearing these attributes

attributeList is a list of attribute specifications, each containing

- an attribute name
- a declared value
- a default value

For example:

```
<!ATTLIST recipe serves CDATA #REQUIRED
                 id     ID    #IMPLIED
                 tested (yes|no|maybe) "maybe">
```

Defining an attribute list (2)

The range of possibilities is actually rather limited:

declared value can be

- an explicit list e.g. (fish|fowl|herring)
- CDATA
- ID, IDREF, or IDREFS

default value can be

- an explicit value e.g. "fish"
- #IMPLIED
- #REQUIRED
- FIXED

Entities

An *entity* is a named sequence of characters, predefined for convenience.
Typical uses include:

- to represent characters which cannot reliably be typed in
- as a short cut for boiler plate text
- containers for external (non-XML) data such as graphics
- as a means of abbreviating parts of a DTD (parameter entities)

A special form of entity name is available for most characters, based on its position in the ISO 10646 standard.

Entities: some examples

```
<!ENTITY mdash "&#x2014;">
<!ENTITY tei "Text Encoding Initiative">
<!ENTITY fig1 SYSTEM "fig1.png" NDATA png>
<!ENTITY % foodTypes
      "(veg|prot|fat|sugar|flavour|unspec)">
```

A parameter entity is one way of changing the range of values permitted for attribute values.

```
<!ATTLIST food type %foodTypes; #IMPLIED>
```

If a DTD contains two or more definitions for the same entity, then the first one found wins. This means a declaration in the DTD subset can over-ride one in the DTD:

```
<!DOCTYPE cookBook SYSTEM "cookbook.dtd" [
<!ENTITY % foodTypes "(good|bad|indifferent)">
]>
```

Second exercise

Use Xmetal to markup The Fly poem.

- The script is in your handouts
- This exercise uses the same toy DTD we described above
- How useful do you think this DTD is?

We have a site licence for Xmetal, which is a commercial product. Next week we will use a different, open source, editor.

DTD : what does it really 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 (if you can find them) for XML DTDs usually combine the two, hence they lack modularity

Some typical scenarios

1. Make up your own DTD
   - ... starting from scratch
   - ... by combining components from one or more pre-existing conceptual frameworks (aka **architecture** or **namespace**)
2. Customize a pre-existing DTD
   - **definitions** should be meaningful within a given user community
   - **declarations** should be appropriate to a given set of applications

The TEI is a good candidate for the second approach

The T E what?

- Originally, a research project within the humanities
  - Sponsored by three 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.tei-c.org/TEI/Lite/)

for the full picture see
http://www.tei-c.org/TEI/Guidelines/

Legacy of the TEI

- a way of looking at what 'text' *really* is
- a codification of current scholarly practice
- (crucially) a set of shared assumptions and priorities about the digital agenda:
  - focus on content and function (rather than presentation)
  - identify generic solutions (rather than application-specific ones)