TEI Simple: ideas and progress so far

Magdalena Turska

September 2014

TEI

- every project is different and we delight in flexibility and customization
- projects are not that different and we aim at standardization & interoperability

‘(...) documents worth encoding in TEI are very different from customer letters. But not that different, and eight out of ten probably will benefit from staying within the confines of a well thought-out standard schema and its surrounding processing rules. And even the two that don’t may benefit from staying within that standard schema as far as possible. ’ - M. Mueller

TEI Simple

- Joint project between TEI Consortium, Mellon Foundation, Northwestern University, University of Nebraska-Lincoln, and the University of Oxford
- TEI Simple aims to define a new highly-constrained and prescriptive subset of the Text Encoding Initiative (TEI) Guidelines suited to the representation of early modern and modern books, and a formally-defined set of processing rules which permit modern web applications to easily present and analyze the encoded texts
- Project investigators are Martin Mueller (NW), Brian Pytlik Zillig (UNL), Sebastian Rahtz (Ox).
- Project team includes Magdalena Turska (DiXiT project, Oxford), James Cummings (Oxford), Lou Burnard (Oxford)
- Project runs from September 2014 to July 2015

Objectives

1. The highly constrained and prescriptive element subset of TEI Simple
2. The processing model (Simple Processing Model: SPM)
3. Formal mapping of the TEI elements used by Simple to the CIDOC CRM
4. TEI-Performance Indicators
5. Integration of TEI Simple into the TEI infrastructure

The CIDOC Conceptual Reference Model (CRM) provides definitions and a formal structure for describing the implicit and explicit concepts and relationships used in cultural heritage documentation.

The Simple schema: guidelines

- ‘Simple’ does not have to mean ‘Small’
- The schema is based on analysis of existing usage from corpora (over 50k printed works)
  - EEBO TCP
  - Oxford Text Archive TEI P5 files (includes ECCO)
  - Deutsches Textarchiv
  - Documenting the American South

  Expecting to also use French corpora
- Our biggest enemy is ambiguity for the encoder
- The target is encoding of the <text>; the <teiHeader> and any <sourceDoc> or <facsimile> must simply conform to tei_all

Results

We isolated 104 elements which are needed in the body of a text

We divide them into groups by function, mainly for documentation purposes

We start analyzing attribute usage

@rend and @type proposal

- We will *not* prescribe *@type*, but instead publish a separate suggested taxonomy for possible interpretative use
- We will preclude use of *@rend* and *@style*
- We will produce a closed list of values for *@rendition* using a pseudo-protocol of "simple:"

```
<p rendition="simple:bold">This is quite bold, but this
<hi rendition="simple:sup">is
   superscript</hi>
</p>
```

Element groups (1)

editorial	abbr add choice corr del desc expan gap handShift orig reg sic space supplied unclear
plays	actor castGroup castItem castList role roleDesc sp speaker spGrp stage
name	name rs
wrapper	argument byline closer dateline epigraph opener postscript salute signed trailer
header	teiHeader
linguistic	c pc s w
titlepage	docAuthor docDate docEdition docImprint docTitle imprimatur titlePage titlePart

Element groups (2)

titlepage	docAuthor docDate docEdition docImprint docTitle imprimatur titlePage titlePart
(general)	author ab address addrLine addSpan anchor back bibl body caesura cb cell cit code date div emph figDesc figure floatingText foreign formula front fw g gloss graphic group head hi item l label lb lg list listBibl measure milestone name note num p pb ptr q quote ref row seg soCalled table TEI term text time title

Prerequisites

- choices about rendering preserved in ODD file
- support multiple rendering output
- support multiple uses of markup
  - appear in rendering
  - contribute to an index
  - subscribe to a facet
  - appear on a map
- cf <equiv> to map to RDF
- cf <constraint> for providing rules about validity

ODD extension

<process> instruction available for <elementSpec>
will define a way of processing this element

multiple processing instructions may occur to define expected behaviour in various contexts or output formats

<process> element

<process> instruction available for elementSpec will define a way of processing this element

multiple processing instructions may occur to define expected behaviour in various contexts or output formats

process attributes

xpath/constraint XPath expression defining a context in which this processing instruction is applicable

name/action name of the function from TEI Simple function library to be applied; input for the function supplied as function parameter

mode output mode for which this processing instruction is applicable

style/class css class or simple:class name of formatting instruction to be applied to the output

Defaults

- * if no <process> for any given mode, emit textual content
- * if no @xpath, means any use of this element
- * <process> rules are additive, not alternates
- * @style defaults to element name (equates to HTML @class or Word style)
- * @mode defaults to 'render' (????)
- * appearance overridden by @rendition and @style
- * <rendition> override remote CSS

New ODD example

```
<elementSpec ident="choice">
 <process xpath="not(ancestor::front) and corr and sic"
  name="makeInline(reg)" mode="render"/>
 <process xpath="corr and sic"
  name="makeMarginalNote(corr)" mode="render"/>
 <process xpath="ancestor::front and (corr and sic)"
  name="makeLinkedMarginalNote(corr,sic)" mode="render"/>
 <process xpath="corr and sic"
  name="makeInline(corr)" mode="textextract"/>
</elementSpec>
```

```
<elementSpec ident="speaker">
 <process name="makeInline(.)"
  mode="render"/>
</elementSpec>
```

```
<elementSpec indent="name">
 <process name="makeInline(.)"/>
 <process name="makeMarginalNote(.)"/>
</elementSpec>
```

Multiple outputs

```
<elementSpec ident="app">
 <process xpath="not(ancestor::app) and (lem)"
  name="makeMarginalNote(.)" mode="render" class="note"/>
 <process name="makeInline"
  content="lem" mode="render"/>
 <process name="makeInline"
  content="lem" mode="textextract"/>
</elementSpec>
```

Lb and fw

```
<elementSpec ident="lb">
 <process name="makeNewline()"
  mode="diplomatic"/>
<!-- no specific process instruction for default "render"
mode means to output textual content (in this case: no
output)-->
</elementSpec>
```

```
<elementSpec ident="fw">
 <process name="omit()" mode="render"/>
 <process name="makeMarginalNote(.)"
  mode="diplomatic"/>
<!-- no specific process instruction for default "render"
mode means to output textual content (in this case: no
output)-->
</elementSpec>
```

titlePage

```
<elementSpec ident="titlePage">
 <process name="makeBlock(.)"
  mode="render"/>
 <process name="oimt()"
  mode="textextract"/>
</elementSpec>
```

A preliminary processing categorisation

Catego	Meaning	Example
1	metadata header	fileDesc
4	structural division	div
5	uncategorized block level object	quotation
6	semantic block level object	person
7	uncategorized inline object	hi
8	semantic inline object	persName
9	list	list
10	list item	item
11	table	table
12	cell	cell
13	row	row
14	out of line note	note
15	figure	figure
16	pointer	ptr
17	Janus element (alternate children)	choice
18	modern commentary element	desc

How does that work?

1. Processing categories define a format-independent way of expressing required output.
2. Function library that can handle these processing scenarios is an essential part of TEI Simple environment.
3. The categories are mapped to a presentation format, using HTML and CSS concepts where possible.