Creating High-Quality Print from TEI Documents

Generating LaTeX with the TEI Processing Model

Beni Ruef[1] Wolfgang Meier[2]

[1]Law Sources Foundation of the Swiss Lawyers Society

[2]eXist Solutions GmbH

TEI Conference 2019 in Graz

Outline

Who/What is the Swiss Law Sources Foundation?

research institution, founded in 1898 by the Swiss Lawyers Society

- publishes critical editions of Swiss law sources from the the early Middle Ages until 1798 in its “Collection of Swiss Law Sources” (aka SSRQ: **S**ammlung **S**chweizerischer **R**echts**q**uellen)
- law sources: sources of law like town charters, constitutions, rights of use, catalogues of goods, court decisions etc.
- 120 (logical) volumes with a total of 82’000 pages finished
- 16 projects ($\equiv$ volumes) in progress (whereof 10 TEI-based), projects can last 10 years…

Characteristics of the Collection

- different text types, diachronic, and doubly multilingual, both source texts (Latin, French, Italian, German and Romansh) and editorial content (German, French and Italian)
  in other words: very heterogeneous
- mirrors the technical development within the last 120 years:
  - until ca. 1995: lead type
  - from ca. 1995 until 2010: *FrameMaker*
  - from 2010 until 2018: *InDesign*
  - since 2011: LaTeX (desperate because of *InDesign* and deciding to do everything ourselves…)
  - TEI since 2010 (retrodigitalization) / 2012 (digitally born)

What’s so special about the SSRQ’s TEI Portal?

- entirely based on the TEI Processing Model (and the TEI Publisher Libraries, respectively)
- complex critical apparatus with highly nested TEI elements, e.g. `<app><lem/><rdg><unclear/></rdg></app>`
- rich semantic annotations (`<date>`, `<measure>`, `<persName>`, `<placeName>` etc.)
- complete schema and validation down to attribute values documented in the ODD
- multilingual: labels and attribute values translated to German, French and English

Why/how to produce a print version in the TEI age?

- yes, a print version is still needed!
  - as an immutable publication suitable for reference
  - as long-time storage *(don't laugh)*
- requirements
  - well-defined, clear structure and clean layout
  - supporting typographic complexity (indices, hyphenation, line numbers etc.) and quality (ligatures, kerning etc.)

Choices

XSL-FO

- fails short on formatting a complex critical apparatus
- typesetting quality does not live up to high demands unless using a commercial engine

LaTeX

- has a long tradition of formatting scholarly editions, providing packages like reledmac
- LaTeX code can preserve most of the TEI semantics: readable for humans, easier to debug
- SSRQ has LaTeX expertise

PM instead of XSLT: why would you?

- single ODD expresses web and print output in **descriptive**, **standardised** way
- document everything in place using TEI (**literate programming**)
- sustainability, interoperability
  - limited, standardised syntax: can still be understood and processed in 20 years
  - TEI source + ODD = all you need to (re-)generate a meaningful representation

Some issues though ...

- LaTeX is not a low-level formatting language
- packages tend to be highly customizable
- no general formulation for a particular feature: there's more than one way!

Challenges

- static mapping of PM behaviours to LaTeX commands/environments tends to be too limited
- CSS as language for `<outputRendition>` fails short

Thus...

We need some kind of **templating** on top of behaviours!

Extension Take One: `<pb:template>`

Output a persName to LaTeX macro:

```
<elementSpec ident="persName" mode="add">
    <model behaviour="inline" output="latex">
        <pb:template>\persname{[[content]]}</pb:template>
    </model>
</elementSpec>
```

- output remains human-readable
- semantics are preserved

<pb:template> Element

```
<model behaviour="inline" output="latex">
    <pb:template>\persname{[[content]]}</pb:template>
</model>
```

- may occur once within a model
- may contain text (e.g. LaTeX) or XML (HTML) content
- placeholders in [[..]] to inject other parameters
- template expanded first and result passed to behaviour replacing parameter `content`

Example

Additional parameter passed in:

```
<elementSpec ident="persName" mode="add">
    <model output="latex" behaviour="inline">
        <param name="ref" value="@ref"/>
        <pb:template>
        \persname{[[ref]]}{[[content]]}
        </pb:template>
    </model>
</elementSpec>
```

Extension Take Two

An even simpler proposal, extending `<param>`:

```
<elementSpec ident="persName" mode="add">
    <model output="latex" behaviour="inline">
        <param name="ref" value="@ref"/>
        <param name="content" type="template">
        \persname{[[ref]]}{[[content]]}
        </param>
    </model>
</elementSpec>
```

- allow template for any param
- @type=”template” to distinguish and allow future extensions

Key Benefits

- enables arbitrary complex output to be generated (text or XML)
- does not change the semantics of `<model>`
- very **fast implementation**: directly translated into XQuery functions – zero overhead
- other output formats possible, e.g. TUSTEP ...

Using Templates to Produce TEI

- TEI PM is not limited to TEI: general purpose tool to process any XML
- Example: transform docx into TEI using an ODD
- Same features as TEI XSL stylesheet but just 170 lines of ODD!

Take away

- templating allowed us to generate LaTeX code fully compatible with earlier volumes
- existing LaTeX styles were reused
- a single ODD describes transformations to HTML and LaTeX!
- custom behaviours avoid redundancy and writing XQuery code

Links

- Law Sources Foundation of the Swiss Lawyers Society
- SSRQ's TEI Portal
- The TEI Processing Model in the Guidelines
- TEI Publisher