Universität Zürich[UZH]

Technical Report

Stand-off Markup in TEI Documents: Improving the Structure of the Text+Berg Corpus

Author: Mathias Müller
mathias.mueller@uzh.ch

Institute of Computational Linguistics
University of Zurich, Switzerland

Current version: 02 —— October 3, 2016

Contents

1 Introduction

This document serves two purposes. First, it is an introduction to stand-off markup in the context of the Text Encoding Initiative (TEI) Guidelines for encoding language resources. Second, it presents several scenarios showing how stand-off markup could actually be introduced into an existing code base, the Text+Berg corpus. I have been very careful to make those scenarios as tangible and explicit as possible, in order to allow informed decisions on the adoption of stand-off markup.

Even though this document was written in the course of a project that focuses on one specific corpus, the Text+Berg multilingual resources of the Institute of Computational Linguistics, University of Zurich, virtually everything applies to TEI corpora in general. The reading will require prior knowledge of at least XML and TEI P5.

2 What exactly is stand-off markup?

When language resources are stored in our digital age, and especially if their creators have linguistic uses in mind for them, they are not stored merely as text. For many applications of electronic text in computational linguistcs and related fields, text is only useful if information is added on top of the actual text content. Additional information can either mean meta data (for example, the time and place a certain corpus has come into being) or any description, interpretation or analysis of – even imposing structure on – the data. It is wise to uphold a strict separation between the actual data and any additional information, much in the same way as web development tells us to always separate content from presentation. And that is where markup finally comes into play.

Markup is a technique, usually with a finite set of key words or instructions to make the result machine-readable, that allows annotating the source text with additional information while making sure that the two dimensions are never confused. The only type of markup language that is of interest here is XML markup, and most of the readers should already be familiar with it. This is not an introduction to the XML standard, please inform yourself elsewhere and only read on as soon as you are comfortable with the topic. Returning to the properties of XML markup, the following XML example illustrates how content is easily separated from annotation:

```
<s authored="Noam Chomsky">Colorless green ideas sleep furiously.</s>
```

Listing 1: `source.xml`

If needed, the source text can be extracted from this XML snippet with ease. In a simple case, this setup, i.e. the source text and all annotations kept in the same file, might already be sufficient. In real-world projects, researchers are often confronted with constraints that make it impossible to organize their language resource in this way. For instance, they might be required to leave all the source content intact (other reasons are discussed later in this chapter). Then, it would be advisable to think about *stand-off markup*.

Stand-off markup describes markup that is, for some reason, removed from the precise location it is meant to ”mark up”. Developing our example above, a stand-off version could look like:

```
<s xml:id="s123">Colorless green ideas sleep furiously.</s>
<author>
   <name>Noam Chomsky</name>
   <occurrence reference="#s123"/>
```

```
</author>
```

Listing 2: A stand-off version of `source.xml` (only an XML fragment)

where the <s> and <author> elements are separable items, but still reside in the same physical file on storage. But there is nothing stopping us from storing stand-off annotations in a separate file:

```
<s xml:id="s123">Colorless green ideas sleep furiously.</s>

<author>
   <name>Noam Chomsky</name>
   <occurrence reference="source.xml#s123"/>
</author>
```

Listing 3: Stand-off markup in separate files

The most extreme form of stand-off markup, aptly named "extreme stand-off markup", posits that the source text should not contain any markup whatsoever, to guarantee that the annotations do not interfere with the source:

```
Colorless green ideas sleep furiously.
```

Listing 4: `source.txt`

```
<s xml:id="s123">
   <xi:include href="source.xml"/>
</s>
```

Listing 5: `segmentation.xml`

```
<author>
   <name>Noam Chomsky</name>
   <occurrence reference="segmentation.xml#s123"/>
</author>
```

Listing 6: `authors.xml`

Adding many more levels is left as an exercise for the reader, but it should be easy to see the kind of frameworks stand-off annotations lead to. Usually, annotations layers are organized hierarchically, in the sense that, in our example, segmentation.xml is the first stand-off document that references the source text. All subsequent stages do not directly reference the source document anymore. If very well planned, the extreme stand-off markup approach can be powerful, but there is a catch. With current XML technologies, it is definitely impossible to implement a solution that automatically internalizes data from all annotation layers. Given the gigantic size of today's language corpora, we absolutely rely on automatic means of processing the texts

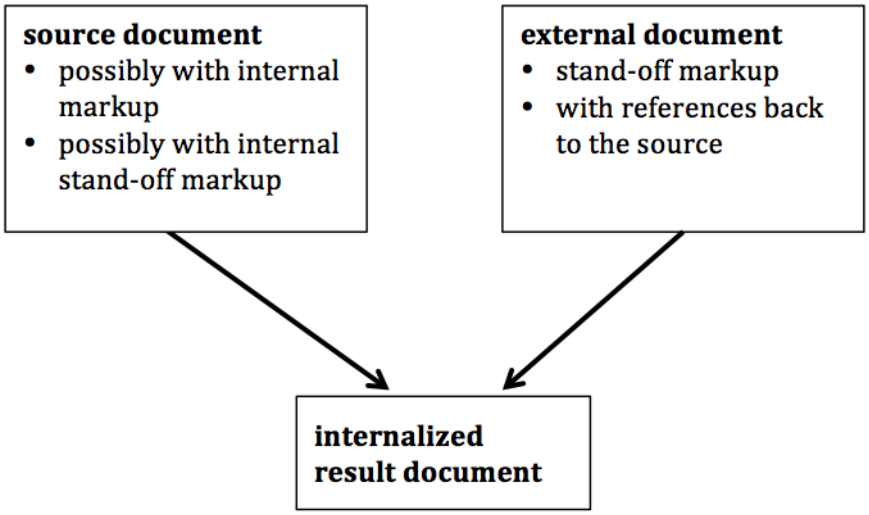

Figure 1: General architecture of stand-off markup, either internal to the source document or stored externally. The process of merging the content back together is called *internalization*.

and assembling components. The precise reason for this impossibility is a (slightly technical) shortcoming of the XInclude specification: extreme stand-off would require XIncluding parts of the source text with character offset pointers (presumably using the string-range() function of the xpointer() scheme), which is currently impossible because an XInclude element does not allow `parse="text"` and the presence of the **`xpointer`** attribute at the same time. Put another way, XInclude only addresses specific parts of documents if they are well-formed XML documents. I highly recommend rereading this section after consulting the chapter on XInclude and XPointer.

Even if extreme stand-off is out of the question, information can be kept in several files, but it should always be possible to 1) reconstruct the relationships between multiple documents and 2) assemble all pieces back together to form a whole. The first requirement is usually realized with links, references or pointers that are truly indispensable in such a framework, whereas the techniques to assemble stand-off markup are less straightforward and only possible with custom tools, regrettably only for use in a specific project. As mentioned already, the concept of stand-off markup does not limit the number of files involved, which comes in handy for the development of corpora with several layers of annotation. Typically, text is annotated on the levels of parts of speech, lemmata, polarity, syntax, named entities, times and places and so on. Figure 1 summarizes all that has been said so far on stand-off document structures.

3 Stand-off architecture in TEI documents

At this point, we can make the leap to a specific vocabulary of XML markup: TEI P5 – and to how stand-off markup is envisioned in the current release of the TEI guidelines. A large part of chapter 16 ("Linking, Segmentation and Alignment") is devoted to stand-off architecture, although TEI only half-heartedly embraces the concept. Much of the planned work on TEI stand-off has fallen by the wayside, together with the W3C efforts on the technologies used for TEI stand-off that have not exactly been fruitful until today. Still, chapter 16 outlines the reasons for using stand-off markup and how to realize stand-off in practice. Before we delve into the rationale behind stand-off markup, it should be noted that the following is still greatly hindering actual implementations: There is no agreed upon container for stand-off markup in the TEI vocabulary (though many variants have been put forward, including <**standoff**>, <**stf**>, <**stdf**>, <**stand-off**> and <**stfGrp**>) and there is no consensus about where stand-off markup should be put in a TEI document. There, the most prominent suggestions are inside <**teiHeader**>, <**back**> of the text or as the <**text**> of a separate document.

3.1 When to use stand-off markup in TEI documents

Stand-off annotations are not the norm within TEI, and are definitely more complicated conceptually, compared to TEI documents that only have internal markup. Therefore, the adoption of stand-off markup should be a well-considered decision. The TEI Guidelines and the TEI Wiki cite the following reasons for using stand-off markup, and I have added my own thoughts:

- The source document is read-only, for example because it is supplied by a third party that bars any modifications.
- The source document should not be modified for other reasons, e.g. because of a firm belief that markup should not interfere with source text.
- It should be possible to publish the annotations without the source text, e.g. due to legal issues.
- Introducing stand-off markup can help to avoid redundancy.
- It is known that the majority of the user base only needs access to a certain part of the annotations, and do not want to be troubled with the stand-off markup.

If none of the above is true for your project, you should probably reconsider your decision to use stand-off markup. After clarifying use cases for stand-off architecture, the next subsection will discuss how exactly it can be realized with the means available to us in TEI.

3.2 How to technically realize stand-off markup in TEI

So far we have only been concerned with explaining the concept of stand-off markup without showing any actual TEI XML. Be warned that some of the ideas presented in this section are purely theoretical considerations – that have not been followed up with an implementation.

3.2.1 Using stable identifiers and TEI vocabulary

Remember that one of the most basic tenants of stand-off markup is that the relationships between the source text and the stand-off annotations must be traceable at all times. Referring, linking or pointing to things becomes a lot easier if the target content has an *ID* – a stable identifier that is unique throughout the whole document. This is in line with the XML standard itself, which defines an attribute **xml:id** that resides in the predefined namespace **xml:**. Any decent XML processor requires that all values of **xml:id** in a document must be unique, otherwise it will reject the document as being malformed. Given that all targets have an ID, the TEI provides a number of elements and attributes to establish relationships between stand-off markup and the source fragment they annotate. Let us recall the example from previous sections, a TEI document fragment with an ID:

```
<s xml:id="s123">Colorless green ideas sleep furiously.</s>
```

Listing 7: `source.xml`

Now, there are several simple methods to express a relationship, each with slightly different semantics. There is an element <**ptr**> that simply "points to another location":

```
<person>
   <persName>Noam Chomsky</persName>
   <ptr target="#s123">
   <ptr target="source.xml#s123"/>
   <ptr target="http://www.example.com/source.xml#s123">
</person>
```

Listing 8: `stand-off.xml` with TEI pointers

The example above illustrates three different kinds of pointers (in this order): A pointer to another location in the same document, a pointer to a location in another document and finally a

pointer to a location in a document from a remote server. Note that pointers have no inherent semantics, apart from pointing to things – it is not defined what kind of relationship they establish or what purpose the pointing should serve. Thus, the intended use of pointers should be detailed in the documentation of your own TEI document format, or additional attributes should be used to clarify the intended purpose.

Another method, closely related to pointers, is the <**ref**> element, which references another location. Again, the intended meaning of the reference is a matter of interpretation, but <**ref**> is commonly used for cross-references in texts, along the lines of "see figure 2 for illustration". If this element were to be employed for stand-off markup, this is what the result could look like:

```
<person>
   <persName>Noam Chomsky</persName>
   <p>Noam Chomsky has coined the sentence
    <ref target="source.xml#s123">
    Colorless green ideas sleep furiously.
    </ref>.
   </p>
</person>
```

Listing 9: `stand-off.xml` with TEI references (indented to improve formatting)

The <**ptr**> and <**ref**> elements also have in common their directionality. They establish a relationship that is essentially one-way; it is always clear which location is doing the pointing and which one serves as the target. To encode a bidirectional (or "non-directional") relationship, the TEI offers the <**link**> element:

```
<person xml:id="p234">
   <persName>Noam Chomsky</persName>
</person>
<link target="source.xml#s123 #p234"/>
```

Listing 10: `stand-off.xml` with TEI links establishing a non-directional relationship

Besides the elements mentioned so far, there is a whole array of attributes useful for pointing and linking, mostly within **att.global.linking** and **att.pointing**. Those attributes will not be explained here, except for the **corresp** attribute. The following example makes use of this attribute and is more comprehensive than the examples we have seen so far, in that it is a valid TEI document:

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
   <teiHeader>
      <fileDesc>
         <titleStmt>
```

```
            <title></title>
         </titleStmt>
         <publicationStmt>
            <p></p>
         </publicationStmt>
         <sourceDesc>
            <p></p>
         </sourceDesc>
      </fileDesc>
   </teiHeader>
   <text>
      <body>
        <p>
           <s xml:id="s123">Colorless green ideas sleep furiously.</s>
        </p>
    </body>
    <back>
      <div type="persons">
       <listPerson>
        <person xml:id="p234">
          <persName corresp="#s123">Noam Chomsky</persName>
        </person>
       </listPerson>
      </div>
    </back>
  </text>
</TEI>
```

Listing 11: `stand-off.xml` with corresp attribute

3.2.2 Automatic internalization with techniques external to TEI

All elements and attributes presented in this section are convenient methods of expressing relationships between the source content and the stand-off markup, especially because an **`xml:id`** attribute is allowed on any XML element. Yet, none of them solves the problem of internalization, that is, assembling the parts back together. The TEI guidelines suggest to use XInclude and XPointer for that. Both of those W3C technologies will be explained in detail in the next chapter.

4 XInclude and XPointer: Including and addressing XML

4.1 XInclude

XInclude is an XML technology and an official W3C recommendation. It introduces an element <**xi:include**> that has its own namespace and is interpreted by a compliant XML processor not as actual XML content, but as a placeholder for content that should be included from elsewhere, and inserted at the precise location of the <**xi:include**> element. More precisely, XInclusions have replacement semantics – once the XInclude statement is executed, the original <**xi:include**> element disappears from the document. The general syntax of an XInclude statement is:

```
<xi:include xmlns:xi="http://www.w3.org/2001/XInclude"
href="resource.xml" parse="text|xml" xpointer="fragment_identifier">
<xi:fallback>fallback content</xi:fallback>
</xi:include>
```

Listing 12: XInclude syntax

There is an **href** attribute that identifies an external document, the **parse** attribute declares the format of the said document (either text only, or XML) and finally there is an attribute **xpointer** which can hold an XPointer fragment identifier, which is used to identify a portion of the target document. A number of restrictions are placed on the XInclude mechanism:

- if **href** is present, the XInclusion definitely targets an external resource
- if **href** is missing, the XInclusion definitely includes from the same document
- if **href** is absent, **xpointer** must be present
- if `parse="text"`, then **xpointer** cannot be present
- if an XInclusion fails, the <**xi:include**> element is replaced with the content of <**xi:fallback**> if there is such an element

One of the not-so-obvious consequences of this set of restrictions is that XInclude will not work with an extreme stand-off framework, because there would be no way to reference parts of the original, source document that does not have any markup. The use of <**xi:fallback**> is discouraged in TEI, rather, the processing should fail when XInclusions cannot be executed. Before studying an actual example of XInclude used to internalize TEI stand-off markup, it is worth looking at the value of the **xpointer** attribute, a so-called "fragment identifier". XPointer is a W3C recommendation in its own right, and will be explained in the next section.

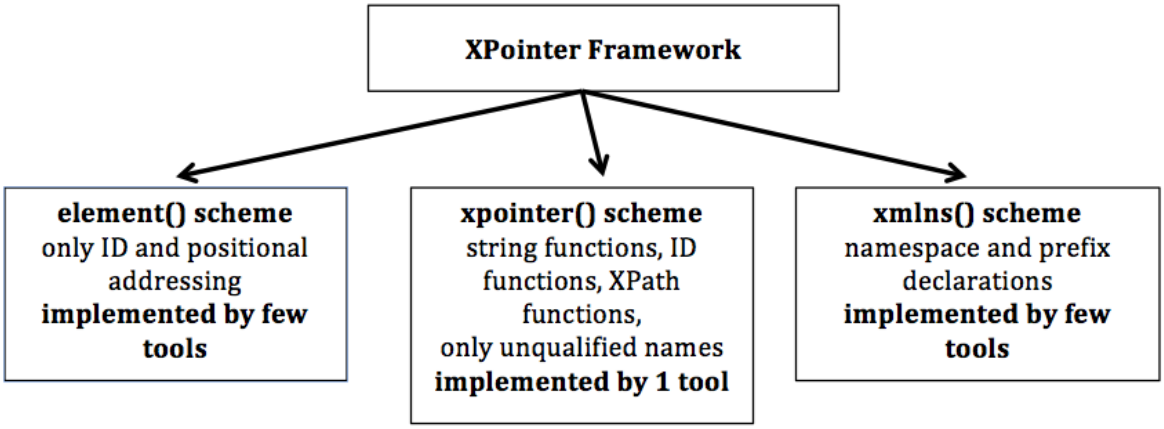

Figure 2: The XPointer framework acts as a modular container for XPointer schemes

4.2 XPointer

XPointer is a language for addressing parts of XML documents, very much like XPath. The W3C defines an overarching framework (the "XPointer Framework") to hold all the subcomponents of XPointer, the so-called *schemes* (do not confuse with schemas). Each scheme has its own capabilities and functions. Since the schemes do not depend on each other, they can be implemented in isolation when implementing all of them would be too cumbersome, considering that very few people use XPointer. Apart from modularization, the second advantage of a framework that acts as a container is that it facilitates the addition of new schemes. To this end, the W3C have set up an XPointer scheme registry, although it is rarely consulted. Figure 2 shows the three predefined schemes:

- **element()** Addressing XML elements that have an ID or via their position in the tree. The W3C rules that an application only qualifies as compliant to the XPointer recommendation if it implements the element() scheme.
- **xpointer()** The most comprehensive scheme, supports string functions, XPath expressions and addressing with IDs, the most prominent function being `string-range()`.
- **xmlns()** Holds namespace declarations by associating namespace prefixes with namespace URIs and making the declared prefixes available to other schemes.

As for the XPointer language itself, the general syntax is:

```
<pointer xpointer="http://www.example.com#xpointer(//item[1])"/>
```

Listing 13: XPointer syntax (the attribute value only)

The first part before # is an URI that identifies an XML resource, either local or on a remote system, and everything that comes after # is called an XPointer fragment identifier. **xpointer** parenthesizing the expression inside should be interpreted as follows. The xpointer() scheme must be used for evaluation, and the expression **//item[1]** is evaluated according to the rules defined in the xpointer() scheme working draft. In this case, the expression is an XPath expression that selects all elements that are called "item", but only if they are the first child element with this name, relative to their parent.

XPointer expressions can point to either locations, ranges or points inside an XML document. As such, they are a clear extension to the XML processing model: In XDM, the current data model for XML, locations could be thought of as positions in the tree, but ranges and points have no representation whatsoever. This makes integration of XPointer with existing XML technologies a challenge.

But even if implementing XPointer is hard, the XPointer framework and the schemes have simply received little attention. This makes problems arise, which can be summarized succinctly: the unstable status of the W3C documents, the TEI guidelines themselves being unclear on the subject, differences between TEI and W3C XPointer schemes, scheme naming confusion and, finally, a severe lack of tool support for XPointer.

After much deliberation, the XPointer framework was finally published as a W3C recommendation, but many of the schemes are working drafts that "may be updated, replaced, or obsoleted by other documents at any time." Naturally, developers are cautious to implement working drafts – even more so because these documents have now been working drafts for more than ten years. The situation is similar in the TEI guidelines, where the use of XInclude and XPointer is discussed in chapter 16. The main obstacle is "that the specification (the TEI Guidelines) doesn't properly address what implementation would mean" (Cayless and Soroka, 2010). The guidelines themselves also acknowledge this fact, for instance by stating that the semantics of internalization are not exactly authoritative:

However, internalization is not clearly defined for all stand-off files, because the structure of the internal and external markup trees may overlap. In particular, when an external markup document selects a range that overlaps partial elements in the source document, it is not clear how the semantics of internalization (inclusion) should work, since partial elements are not

XML objects.

Since internalization is vital to stand-off architecture, it is unfortunate that its definition in the guidelines is still unclear. But the devil is also in the details: in many places, the TEI version of schemes and functions deviate from the one laid out by the W3C. To illustrate this, consider the two incarnations of `string-range()`:

```
string−range(node set, substring, offset, length)
```

Listing 14: Signature of the W3C `string-range()` function

```
string−range(fragment identifier, offset, length)
```

Listing 15: Signature of the TEI `string-range()` scheme

Not only is the "substring" argument missing in the TEI version of `string-range()`, the TEI `string-range()` is also, confusingly, conceived of as an XPointer scheme in its own right, not as a function of the xpointer() scheme, as in the W3C version. A possible explanation for this "unfortunate homonymy" (Bański, Tei Wiki on XPointer) is that the TEI stand-off working group had no hope that the W3C xpointer() scheme would ever be implemented. Redefining the function that retrieves string ranges as a scheme makes it independent of the implementation status of the xpointer() scheme.

4.3 Tool support

Because of the reasons laid out above, and because of sheer lack of interest, there are very few implementations of XInclude and XPointer, and all of them are incomplete.

- Popular XML editors like Oxygen[1] do not XInclude if the target is in the same document because of XML parser streaming, which makes it impossible to include parts of the document while it is still being read. Presumably, all streaming parsers have this limitation and it is perhaps not an outlandish claim that in the near future, almost all XML processing will be streamable.

[1] See `http://www.oxygenxml.com/xml_editor/xinclude_support.html` for Oxygen documentation.

- Saxon[2], the most reliable XSLT processor, very often used for XML processing, does not have its own implementation of XInclude. It depends on the XML parser to XInclude before handing the document tree to Saxon. This is only possible with Xerces. Xerces does not support the xpointer() scheme[3].
- There is only one XSLT 2.0 implementations of XInclude that I am aware of: XIPr, only wholesale includes with href or with an xpointer attribute, but only using the element() scheme.
- Then there is a set of XSLT 2.0 stylesheets (http://rotorz.com/ xml/2009/XPointer/) that are an implementation of the XPointer framework, supporting the xmlns(), xmlns-local(), element(), xpath1(), xpath2() schemes – but not xpointer().
- There is only one tool[4] that implements XInclude and the xpointer() XPointer scheme: xmllint[5]. xmllint supports the xmlns(), element() and xpointer() schemes, including the W3C `string-range()` function. Yet, xmllint is not an XSLT processor and cannot transform XML documents once the XInclusions are done.

To make a long story short, setting up your language resource as stand-off TEI documents with XInclude instructions that automatically assemble content will

- definitely require custom tools if the XInclusions entail selecting parts of strings from other documents and if it should be possible to directly apply XSLT transformations to the XIncluded result.
- most likely require custom tools otherwise. As you know by now, XInclude and XPointer are not exactly mature technologies and you may need to spend a lot of time reading the unwieldy W3C specifications.

5 Examples of stand-off markup with XInclude and XPointer

This section first gives a complete (hypothetical) example of stand-off documents with XInclude and XPointer. Then, I present two actual projects that have used XInclude and XPointer.

[2]See `http://saxon.sourceforge.net/` to download the latest free version (9.6 at the time of writing).

[3]See `https://xerces.apache.org/xerces2-j/faq-xinclude.html` for more details on XPointer with Xerces.

[4]`http://www.w3.org/XML/2002/10/LinkingImplementations.html` lists several others, but all of them are unavailable.

[5]See `http://xmlsoft.org/xmllint.html` for online synopsis and documentation. xmllint is available in most Unix-like environments.

5.1 Proof of concept with XInclude

The following sample documents illustrate how XInclude can be used to establish relationships between documents, and how the XPointer `string-range()` function can be used to address targets from the source document. The example is taken from the TEI guidelines, but adapted to work with xmllint, because the functions presented in the guidelines are not implemented. It is worth reiterating that XIncluding substrings from a text document without XML markup is impossible. Therefore, wrapping the text content in a single XML element (as is done in the source document of this example) is the closest we can get to extreme stand-off markup.

```
<content>
To make a prairie it takes a clover and one bee, One clover,
and a bee, And revery. The revery alone will do, If bees are few.
</content>
```

Listing 16: `source.xml` with minimal markup (only indented to improve formatting)

```
<text xmlns:xi="http://www.w3.org/2001/XInclude">
   <body>
      <head>Poem</head>
      <l>
         <xi:include href="source.xml" parse="xml"
            xpointer="xpointer(string-range(/, 'To', 1, 48))"/>
      </l>
      <l>
         <xi:include href="source.xml" parse="xml"
            xpointer="xpointer(string-range(/, 'To',49, 23))"/>
      </l>
      <l>
         <xi:include href="source.xml" parse="xml"
            xpointer="xpointer(string-range(/, 'To',72, 12))"/>
      </l>
      <l>
         <xi:include href="source.xml" parse="xml"
            xpointer="xpointer(string-range(/, 'To',84,26))"/>
      </l>
      <l>
         <xi:include href="source.xml" parse="xml"
            xpointer="xpointer(string-range(/, 'To',110,17))"/>
      </l>
   </body>
</text>
```

Listing 17: `stand-off.xml` with XInclude processing

The document containing the stand-off markup (`stand-off.xml`) can now be processed automatically by xmllint with the `--xinclude` switch:

```
$ xmllint --xinclude stand-off.xml > internalized.xml
```

Listing 18: Unix command to perform XInclusion

The output file will then look like:

```
<text xmlns:xi="http://www.w3.org/2001/XInclude">
   <body>
      <head>1755</head>
      <l>
         To make a prairie it takes a clover and one bee,
      </l>
      <l>
          One clover, and a bee,
      </l>
      <l>
          And revery.
      </l>
      <l>
          The revery alone will do,
      </l>
      <l>
          If bees are few.
      </l>
   </body>
</text>
```

Listing 19: Result of XInclusion redirected to a file

5.2 TEI projects that have used XInclude and XPointer

Even after searching very carefully, I could not find more than two examples of TEI documents that are described as containing stand-off markup and using XInclude and XPointer. The first example is a representation of a papyrus, created by Hugh Cayless[6]. The document makes use of XPointer expressions as values of the **`corresp`** attribute, but does not use XInclude at all. Also, the string-range() scheme proposed by the TEI guidelines is used, which has no implementation, as you know by now. This means that the correspondence between an entry in the list of persons and the XPointer target cannot be established automatically, only with manual inspection or custom tools.

[6] See `https://gist.github.com/hcayless/2653676` for a Github repository with the code.

```
<TEI>
<!--Header omitted-->
  <text>
    <back>
      <div type="persons">
        <listPerson>
          <person xml:id="tm_person_195364">
            <persName corresp="#string-range(//lb[@n='1'],3,6)"
ref="http://www.trismegistos.org/name/3756">Κράτης</persName>
          </person>
          <person xml:id="tm_person_195365">
            <persName corresp="#string-range(//lb[@n='2'],0,9)"
ref="http://www.trismegistos.org/name/2811">Διονύσιος</persName>
          </person>
          <person xml:id="tm_person_195370">
            <persName corresp="#string-range(//lb[@n='4'],0,4)
                               #string-range(//lb[@n='17'],11,4)"
ref="http://www.trismegistos.org/name/8865">Ἀτᾶς</persName>
          </person>
        </listPerson>
      </div>
    </back>
  </text>
</TEI>
```

Listing 20: A papyrus represented as a TEI document with XPointer references

However, the example above does not use XIclude at all. To my knowledge, the only project that holistically embraces the concept of stand-off markup, and also uses XInclude and XPointer where applicable, is the national corpus of Polish (Bański and Przepiórkowski, 2009). It is described as a multi-layered corpus, the layers corresponding to one of the labels in figure 3.

Still, the authors point out that the shortcomings of XInclude as a language, contradicting specifications and the undecidedness of the TEI guidelines regarding stand-off annotations greatly hindered the development of the corpus. Unfortunately, the corpus is only available through a search interface, and the source is not available for inspection.

Regarding the fate of XInclude and XPointer, it is fair to say that there will be no breakthroughs in the near future. It would be very unwise to make your corpus design depend on schemes or functions that are not implemented yet, because it is not likely that they will ever be implemented. There are other methods to organize stand-off annotations, although they require more effort and a priori knowledge about corpus structure.

Now that we have discussed the notion of stand-off markup in general, surveyed all means available in TEI to reference, link to or point to content and studied XInclude and XPointer

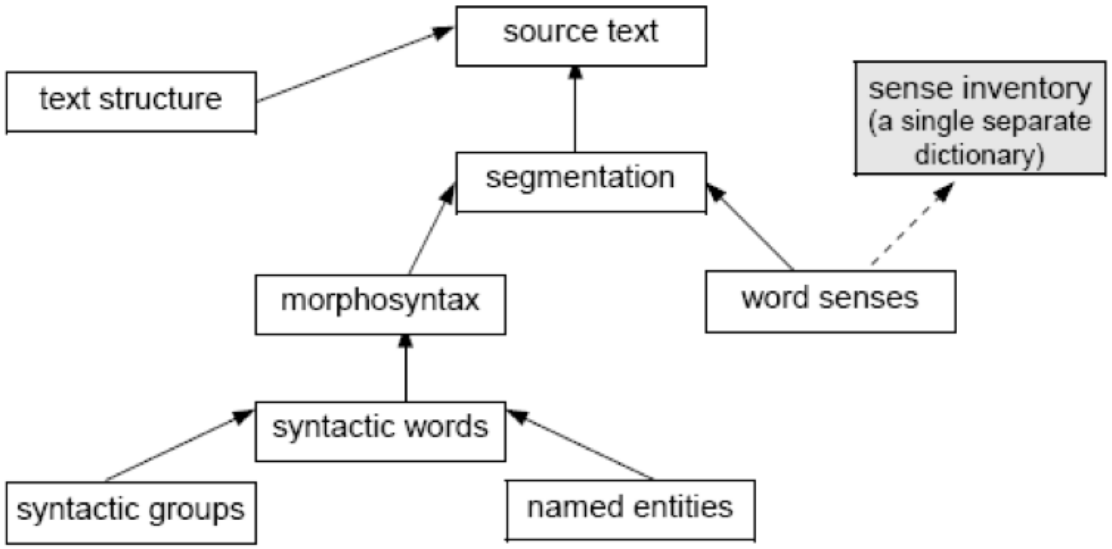

Figure 3: The stand-off architecture of the national corpus of Polish. Each annotation level references the source document or more fundamental stages of annotation.

in detail, we are now well prepared to make informed decisions about the Text+Berg corpus structure.

6 Recommended scenarios for Text+Berg

There are several ways to go about improving the corpus structure of Text+Berg, depending on how much work should be put into it, but I regard the following as the absolute minimum that must be done:

- IDs must conform to the XML specification (e.g. cannot contain colons or start with numbers) and be consistent throughout and across documents.
- The main yearbook documents holding the articles must be stored in a standardized format like TEI.
- There must be a schema for all parts of the corpus (the main documents, the NER documents, the sentence alignment and TimeML files) to ensure validity. The schemata should be written in either DTD, XML Schema or RelaxNG.

Regarding the stand-off nature of the NER, TimeML and alignment files, I would advise against overusing XInclude and XPointer. The problems associated with those technologies are explained in detail in earlier chapters. If XInclude is used at all, it should only be used for wholesale inclusions of other documents, which is supported by a good many XML applications. If it is guaranteed that IDs are consistent and well-formed, and can always be resolved to a specific document and target, putting the content back together is not rocket science. Put another way, the advantage of automatic internalization with XInclude is clearly eclipsed by the disadvantage of lacking tool support.

6.1 Scenarios

The scenarios are listed in ascending order with respect to complexity and effort required. Also, they build upon each other, i.e. B entails A, C entails A and B and so on.

(A) TEI for the main documents
Only turn main book documents into TEI documents. Only make IDs consistent across documents, mostly adding document names to the references that already exist. Write schemata for at least the main articles.

(B) TEI for all documents except TimeML
Make all documents TEI documents, with the exception of TimeML files, since TimeML is already a full-flegded standard. The alignment files would contain <**linkGrp**> and <**link**> elements (similar to XCES), the NER files would contain <**persName**> and vocabulary for geographical entities.

(C) XInclude content from the main document into NER, alignment and TimeML files
XInclude the content from the source document into the NER and alignment documents, and make the references intra-document IDREF values that are only valid after the content has been XIncluded. However, documents would not be valid TEI if no XInclusion can take place.

(D) Almost extreme stand-off markup
Source text only with minimal segmentation (paragraphs), or even all text inside one single root element, then additional layers of annotation that are stand-off, each Xincluding content with Xpointer string ranges or similar. A lot of effort required, restructuring work would perhaps not outweigh the benefits.

6.2 Closing remarks and current state of affairs

My personal recommendation would be to implement scenarios A and B because they bring practical benefits without complicating corpus access for non-specialist users. As of release 152 in October 2016, scenario A has already been implemented and most of the work required for scenario B is done. Specifically, I have already written

- an XSLT 2.0 stylesheet that transforms the main documents to TEI. `T+B2TEI.xsl`, current version: 6. Requires Saxon 9.* installed on your system. A compliant (but slightly outdated: 9.1) version of Saxon is installed on our servers.

- a documentation detailing how elements and attributes in the main article files are transformed: `mapping_main.pdf`. This document only describes the mapping for scenario A. The mapping for all other file formats (alignment files, NER files, TimeML files) is described in `mapping_auxiliaries.pdf`.

- transformations for the NER, alignment and TimeML files. The stylesheets are called `ner_fix.xsl`, `alignments_fix.xsl` and `timex_fixxsl` respectively. The NER and alignment files are turned into valid TEI documents, in the TimeML files only the IDs are fixed.

  Caution: The NER and TimeML transformations will only produce the correct result if they are passed the ID of the yearbook they reference as a command line parameter.

  Use the transformations for those auxiliary files at your own risk. As of release 152, we do not guarantee that the output will be useful.

- a complete XML Schema for the *original* Text+Berg format that is guaranteed to validate all documents in the current release (151): `TB_schema_release_152.xsd`

- a TEI ODD document for the main Text+Berg TEI documents on the basis of which any kind of schema can be generated. Current version: `TEI_schema_T+B.odd`. Ideally, the ODD document is transformed to RelaxNG, the current version of the generated RNG schema is `TEI_schema_T+B.rng`.

—— July 2015, Mathias Müller

7 References and resources

7.1 Specifications and standards

Normative W3C documents

XInclude recommendation: `http://www.w3.org/TR/xinclude/`

XPointer Framework: `http://www.w3.org/TR/2002/PR-xptr-framework-20021113/`

xpointer() scheme: `http://www.w3.org/TR/xptr-xpointer/`

xmlns() scheme: `http://www.w3.org/TR/2002/PR-xptr-xmlns-20021113/`

element() scheme: `http://www.w3.org/TR/2002/PR-xptr-element-20021113/`

XLink recommendation: `http://www.w3.org/TR/xlink11/`

Markup standards for encoding language resources

Relevant chapter of the TEI P5 Guidelines:
`http://www.tei-c.org/release/doc/tei-p5-doc/en/html/SA.html`

XCES, includes stand-off variants:
`http://www.xces.org/schema/#standoff`

PAULA:
`https://www.sfb632.uni-potsdam.de/images/doc/PAULA_P1.1.2013.1.21a.pdf`

LAF, an ISO standard:
`http://www.iso.org/iso/catalogue_detail.htm?csnumber=37326`

XSTANDOFF:
`http://www.lrec-conf.org/proceedings/lrec2014/pdf/308_Paper.pdf`

TimeML specification:
`http://www.timeml.org/site/publications/timeMLdocs/timeml_1.2.1.html`

7.2 Community efforts

List discussions

Most recent TEI-L thread on the future of XPointer schemes and current implementations:
`http://tei-l.970651.n3.nabble.com/String-range-pointing-td4027078.html#a4027090`

Discussing the name and place of a stand-off element in TEI:
`http://sourceforge.net/p/tei/feature-requests/378/`

Workgroup and Wiki

Summary of the council-appointed TEI workgroup on stand-off markup, mostly unimplemented, theo-

retical considerations:
`http://www.tei-c.org/Activities/Workgroups/SO/sow06.xml`
Stand-off use cases in the TEI Wiki, sadly with only one example:
`http://wiki.tei-c.org/index.php/Stand-off_use_cases`
TEI Wiki article on XPointer in the TEI context:
`http://wiki.tei-c.org/index.php/XPointer`
General discussion of personography with TEI:
`http://danielgriff.in/2014/personography-or-thinking-about-people-with-tei/`

Code samples and proposals
Example of personographic annotations with stand-off:
`https://gist.github.com/hcayless/2653676`
Tentative ODD proposal for stand-off annotations 2014:
`http://sourceforge.net/p/lingsig/code/HEAD/tree/branches/standoff/proposal.odd.xml`
Tentative ODD proposal for stand-off annotations 2015:
`https://github.com/laurentromary/stdfSpec/blob/master/Specification`
Attempted implementation of TEI XPointer schemes with XSLT 2.0 functions (proof of concept only):
`https://github.com/hcayless/tei-string-range`

7.3 Published literature

Proposing the structure of stand-off TEI, specifically an <**stf**>, <**standoff**> and <**stfGrp**> element:
Pose, J., Lopez, P. and Romary, L (2014). A Generic Formalism for Encoding Stand-off Annotations in TEI. HAL. URL: `https://hal.inria.fr/hal-01061548/document`.

Identifying stand-off annotations as a weak spot of the TEI P5 Guidelines, confirming that tool support is virtually inexistent, and that stand-off recommendations are not instructive:
Bański, P. (2010). Why TEI stand-off annotation doesn't quite work: and why you might want to use it nevertheless. In: Proceedings of Balisage: The Markup Conference 2010, Montréal, Canada, August 3 - 6, 2010. Balisage Series on Markup Technologies, vol. 5 (2010). URL: `http://www.balisage.net/Proceedings/vol5/html/Banski01/BalisageVol5-Banski01.html`

Attempt to implement the TEI custom XPointer string-range() scheme with XSLT 2.0 functions:
Cayless, H. A., and Soroka, A. (2010). On Implementing string-range() for TEI. In: Proceedings of Balisage: The Markup Conference 2010, Montréal, Canada, August 3 - 6, 2010. Balisage Series on Markup

Technologies, vol. 5 (2010). URL: http://www.balisage.net/Proceedings/vol5/html/Cayless01/BalisageVol5-Cayless01.html

An actual use case where stand-off annotations were used in a large TEI project:
Bański, P. and Przepiórkowski, A. (2009). Stand-off TEI Annotation: the Case of the National Corpus of Polish. In: Proceedings of the Third Linguistic Annotation Workshop, ACL-IJCNLP 2009, pages 64–67, Suntec, Singapore, 6-7 August 2009. URL: http://www.aclweb.org/anthology/W09-3011

A general discussion about the use of markup languages to encode cultural heritage texts:
Schmidt, D. (2010). The inadequacy of embedded markup for cultural heritage texts. Literary and Linguistic Computing (2010) 25 (3), pages 337–356. First published online April 16, 2010. URL: http://www.hki.uni-koeln.de/sites/all/files/courses/3227/Schmidt-2010.pdf

Oft-cited paper on structuring texts with multiple layers of annotation in a stand-off fashion:
Dipper, S. (2005). Xml-based stand-off representation and exploitation of multi-level linguistic annotation. In Berliner XML Tage 2005, Humboldt-Universität zu Berlin, 12. bis 14. September 2005, pages 39–50. URL: http://www.linguistics.ruhr-uni-bochum.de/~dipper/papers/xmltage05.pdf

On the impossibility of Xincluding stretches of text (poster presentation by Piotr Bański):
http://bansp.users.sourceforge.net/pdf/Banski-Balisage2010-poster.pdf

Encoding biblical texts with TEI P5 stand-off means:
Cummings, J. (2009). Converting Saint Paul: A new TEI P5 edition of The Conversion of Saint Paul using stand-off methodology. Literary and Linguistic Computing (2009) 24 (3). URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/308_Paper.pdf

Rethinking Pointers in TEI:
Cayless, H. A. (2013). Rebooting TEI Pointers. Journal of the Text Encoding Initiative [Online], Issue 6, December 2013, Online since 22 January 2014. URL: http://jtei.revues.org/907

7.4 Tools

Saxon, the most reliable and free XSLT processor (Dr. Michael Kay):
http://www.saxonica.com/welcome/welcome.xml

XPointer Framework for XSLT 2.0 (Lea Hayes):

`http://rotorz.com/xml/2009/XPointer/`

XIPr XInclude Stylesheets (Dr. Eric Wilde):
`https://github.com/dret/XIPr`

Oxygen, a powerful XML Editor (Syncro Soft SRL):
`http://www.oxygenxml.com/`

Altova XML Spy, an XML Editor:
`http://www.altova.com/xmlspy.html`

xmllint (using the Gnome libxml2, Daniel Veillard):
`http://www.xmlsoft.org/`

Xerces, a popular XML parser with some XInclude support:
`http://xerces.apache.org/`

7.5 Color coding scheme

maroon	element names
red	attribute names, namespace prefixes
blue	attribute values, namespace URIs
black	text content
green	comments
purple	TEI classes