Standoff properties as an alternative to XML for digital historical editions

Desmond Schmidt

eResearch, ITEE, University of Queensland, Australia

desmond.allan.schmidt@gmail.com

Abstract

In the past few years digital humanists have recognised that the development of tools for the collaborative creation of shareable, reusable and archivable transcriptions of historical documents is being held back by the complexity and lack of interoperability of customisable XML encoding standards. To fix this problem transcriptions must be split into their constituent parts: a plain text and individual sets of markup. The plain text is easily processable by text-analysis software or search engines, and may be divided into layers representing different states of each document. Each markup set may also represent a separate layer of information to enrich an underlying plain text, and may be freely combined with others. Unlike in XML, textual properties in markup sets can freely overlap. Plain text versions and their corresponding markup can also be reliably and efficiently converted into HTML, the language of the Web. Unlike previous attempts at separating markup from text, the proposed data representation is designed to be fully editable, and development of a suitable Web-based editor is at an advanced stage. To solve the problems we now face will require a radical rethink of how we mark up texts, and the adoption of globally interoperable standards to which these internal representations can be easily converted, such as HTML, plain text and RDFa.

Standoff properties as an alternative to XML for digital historical editions

Introduction

Most digital humanists are familiar with markup, or the use of named tags to indicate properties that text should possess in addition to its basic character data. The dominant form of markup in digital humanities is XML, the extensible markup language (W3C 2008), originally proposed in 1998 as a replacement for SGML (standard generalised markup language), created at IBM for technical documentation (Goldfarb 1990). SGML was itself derived from GML, the generalised markup language still in use, which was one of the first markup languages to enforce the distinction between form and content. GML also formed the basis of HTML, the now ubiquitous language of the Web (Schmidt 2010).

All these forms of markup are embedded in the plain text. They are carried along with it whenever the text is edited or moved. This a great convenience, but it comes at a price that has already been explained at length elsewhere (Schmidt 2010, 2012, 2014). In summary, there are three main problems:

1. *Overlap.* Encoding in a context-free markup language like XML forces the encoder to impose on the text a strict hierarchical structure that often runs counter to the perceived structure of the text. This leads to the familiar 'overlapping hierarchies' problem (Renear et al, 1993) where the content has usually to be fudged, copied or made more complex than would be desired, in order to get around this restriction.
2. *Non-interoperability.* The encoder must embed, along with the tags, his or her particular interpretation of the markup scheme being used, and its application to the text under consideration, within the goals of the encoding project as a whole (Renear 2000, Durusau 2006). This leads to incompatibilities between marked-up texts even when the tags are standardised, as has recently been verified on a large scale in the TextGrid project (Aschenbrenner 2015).
3. *Poor longevity and reusability.* Customising the markup to facilitate particular tasks (for example, text to image linking) or to realise the objectives of a particular editorial project, builds into the transcription dependencies on the software that processes it, and vice versa (Aschenbrenner 2015, 205). When that software becomes obsolete or is lost, the transcriptions must be re-interpreted and the software re-implemented for them to be fully reusable.

These last two drawbacks of SGML/XML markup are often ignored, under the misapprehension that the interoperability and durability of XML itself will automatically be conferred on the data encoded in it. A recent survey by Dee (2014) revealed that encoders of texts in Text Encoding Initiative XML (TEI-XML 2015) believe that following this standard will yield texts that are reusable, interoperable and durable. However, a number of TEI-experts have recently distanced themselves from this popular belief. Interoperability is 'impossible' (Bauman 2011) or 'hard' (Holmes 2015), TEI encoded data 'offer no advantage over plain-text, html, or epub texts' (Mueller 2011), 'Interoperability is essentially at odds with the development of TEI's expressive and interpretative potential' (Gailey 2012). 'A lack of interoperability between digital resources for humanists, whether big or small, plagues our research community' (Brown 2015). 'Sharing and reuse of research data is hard to achieve … Even when projects use shared standards for data, metadata and interfaces, interoperability is not guaranteed' (Aschenbrenner 2015 – author's translation[1]).

Interoperability is probably the most serious of these problems, because without it you cannot build shared software. My tool won’t work with your texts or vice versa, and research funds will be wasted through reinventing the wheel for each project, instead of building on the efforts of others.

Longevity and reuse are similarly threatened: ‘We don’t have a good model in use today for a sustainable XML edition with which we could develop a shared conception of digital editing. There are a lot of silo projects that have been developed for specific sets of documents … but often do not have sufficient infrastructure to ensure longevity’ (Hajo 2010); ‘… once the funding ceases, most of these sources will remain accessible only as long as the supporting hardware and software can be kept in working order.’ (Rosenthaller *et al.* 2015). ‘That all sounds very good, but it doesn’t change one bit the fact that TextGrid, probably in the next few years, will disappear. And it is good if it does.’ (Jannidis 2015, p.33 – author’s translation[2]).

In these quotes the dependency between software and the transcriptions they engage with is not always made explicit. But for there to be no connection, transcriptions in XML would have to be purely descriptive and abstract. A long-standing theory states that the longevities of software and XML transcriptions are unrelated: ‘Provided that the text of a digital edition is encoded following an international standard and equipped by a suitable set of metadata, then it should be relatively easy to preserve and, if necessary, to migrate; the same cannot be said for the interface.’ (Pierazzo 2015, p.187). This is, more or less, what was said 20 years ago: ‘For scholarly editions, such longevity is essential. And that is why I say that the development and use of software-independent markup schemes like SGML and the TEI Guidelines will prove more important, in the long run, to the success of electronic scholarly editions, ... than any single piece of software can.’ (Sperberg-McQueen, 1994). But is that really true? How can you have ‘software-independent markup’?

Software that processes XML relies on the existence of certain elements in the transcription, and each element was also presumably added under the expectation that it would be correctly processed by software. But the huge variability in the selection of elements and attributes in customised TEI-XML is too great for any one program to handle (Haaf *et al.* 2015). Already ten years ago Hillesund (2005) debunked the XML model of ‘one input – many outputs’, arguing that, whatever technology is used, ‘most texts are made for a special use in a specific context for a limited group of readers.’ Alan Galey likewise argues that the separation of form and content in XML runs counter to the humanistic tendency to see the two as distinguishable but inseparable (Galey, 2010, 112). And for confirmation one only has to look at the many presentational features often added to ‘logical’ markup to make it work for specific uses. For example, in the TEI <pb> (page-break) element, <pb n=”2” facs=”./images/0002.png”/>, the file-path and file type link the logical page to an explicit computing environment. Likewise the new <zone> and <surface> elements specify absolute points and dimensions to enclose texts, whose successful presentation will depend on the use of specific fonts, type-sizes, and image resolutions (TEI 2015, Ch.11). And the common practice of linking elements to one another, across the natural structure of XML, presupposes that there is a special program that can do something with that information. Dependencies like these effectively bind the transcriptions to the lifetime of the software that drives them. And once that software fails, the problem reduces to interoperability again: how to access someone else’s data when the software they wrote for it no longer works.

Since interoperability seems to be the key problem that governs the shareability, reusability and longevity of transcription data, the recent spike in discussions about interoperability seems to be building into something of a consensus: that, contrary to theory and popular belief, TEI encoded transcriptions of historical sources are actually *hard* to reuse, share or archive.

In contrast to TEI-XML, plain text, devoid of embedded markup tags, is highly interoperable, durable and reusable, since it may be loaded and fully utilised in thousands of already existing applications. Even without any markup, plain text is extraordinarily rich: all one needs is a suitable text-analysis program to exploit it. The objection that plain text still contains a degree of interpretation (Pierazzo 2015, 109) is valid but moot, since this does not stop it from being interoperable. So the goal of this paper is to describe a practical and working system for separating markup from text to preserve its natural interoperability and durability, without incurring the serious drawbacks of embedded markup languages like XML.

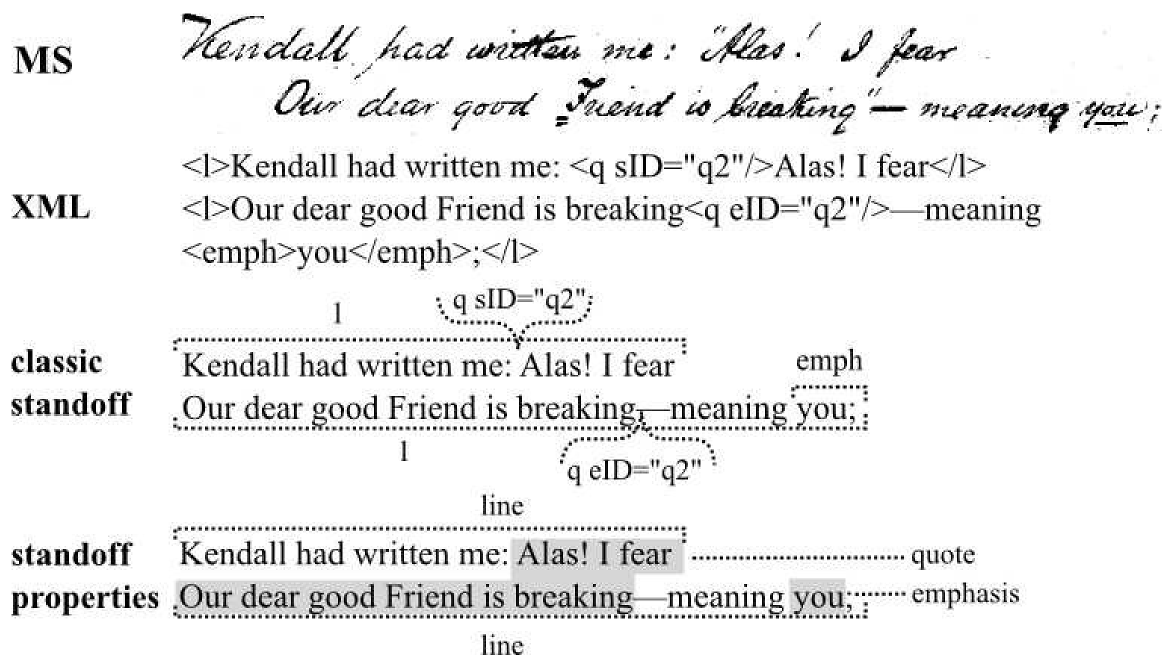
In pursuit of this goal, Section 1 below distinguishes the two main methods for separating markup from text: standoff markup and standoff properties. Sections 2 and 3 then describe the two components of standoff properties: ‘plain text’ and markup. Since markup in this solution allows textual properties to overlap freely (unlike elements in XML) Section 4 describes how this overlap can be resolved when converting sets of markup and their corresponding plain text reliably into HTML. Section 5 then describes the current implementation of standoff properties, and Section 6 summarises the main advantages of this approach.

1. Forms of ‘standoff’

An important distinction must be drawn between the two main forms of non-embedded markup. The first type arose from a long tradition in linguistics of removing originally embedded markup from texts so that several, otherwise conflicting, markup analyses may be combined with it (Souter 1993, Ide *et al.*, 2000, Ide 2012). These are called ‘standoff markup’ systems. In this method the locations of the markup tags are recorded, so they can be later recombined with the text. Although the text may be used independently, a hierarchical or tree structure is still being imposed upon it. And although several different analyses of the text can co-exist independently, they cannot be combined.

Another drawback with standoff markup is the fact that linguists usually have no need to modify the underlying text, and so they do not bother to develop solutions that will allow this. Literary scholars, however, usually require that markup and text be modifiable. Standoff markup systems are therefore seen, perhaps unfairly, as impractical for the encoding of digital editions for scholarly or other uses.

But there is another kind of non-embedded markup, which for want of a better name may be called ‘standoff properties’. Unlike the other two forms, this kind of markup does not have a syntax, and does not form a markup language. Instead, standoff properties are simple names for spans of underlying text that may freely overlap. Fig. 1 shows the differences between the three forms of markup when applied to a fragment of poetry by Charles Harpur (an Australian 19th century poet):



```
<l>Kendall had written me: <q sID="q2"/>Alas! I fear</l>
<l>Our dear good Friend is breaking<q eID="q2"/>—meaning
<emph>you</emph>;</l>
```

Fig. 1: Embedded, classic standoff and standoff properties compared

In the example text the quotation ‘Alas! I fear Our good dear friend is breaking’ cuts across the line-break before ‘Our’. Although there is a <q> tag for a quote in TEI-XML, it cannot be used here because of the overlap between <q> and <l>. Instead, milestone tags are used to try to link the start and end-quotes together (DeRose 2004). But this requires custom software to work, and any normal XML tool would fail to see the intervening text as a quotation. The standoff properties representation is the only one that records the overlap in an elegant and processable form.

In various guises the standoff properties solution has been proposed many times (Nelson 1987, p.147, Thaller 1996, Buzzetti 2000, Smith *et al.* 2006, Piez 2010) but the implementation described here was originally developed as part of the HRIT (humanities resources infrastructure and tools) project at Loyola University, Chicago in 2010 (Shillingsburg and Hayward 2010). This was then developed further in the AustESE project at the University of Queensland 2012-2013 (Osborne *et al.* 2013).

Standoff properties are not a drop-in replacement for XML. Indeed, such a replacement, if it mimicked all the uses to which XML is currently put in the digital humanities, would only be a copy of XML using different technologies, and would not circumvent any of its shortcomings. As described earlier (Schmidt 2014) the uses of TEI-XML fall naturally into four classes:

1. Metadata about the document as a whole (the TEI header)
2. Variation in the underlying text-stream, tags like <app>, <rdg>, <lem>, <sic>, <corr>, <orig>, <reg>, <choice>, <subst>, <add>, <del> etc.
3. Annotation as in <note>, <interp>, <interpGroup>, or the ‘ana’ attribute (Singer 2013)
4. Properties of the text such as <stage>, <hi>, <p> etc.,

The standoff properties solution described here aims to replace class 4 markup only. The other classes are implemented using different technologies. 1. can be handled by external, not embedded metadata; 2. is handled by the NMERGE program (Schmidt 2015b) in the overall Ecdosis editorial system; 3. can be handled by external annotations standards like Open Annotation (Sanderson *et al.* 2013), and tools supporting it. The common practice of linking elements and textual fragments together in TEI-XML is not required, since overlap is eliminated by the combined effect of the NMERGE tool and standoff properties. Hyperlinks can be implemented in standoff properties as described below, or as annotations.

So standoff properties form part of an overall solution to the problem of encoding historical documents for publication in digital form on the Web or elsewhere. Since standoff properties require textual data to be divided into two forms: plain text and markup, these two categories will now be examined in more depth.

2. Plain text

A good plain text should be as reusable and durable as possible. The representation of as wide a range of character codes as technically feasible in plain text is easily achieved by the adoption of Unicode (2015). And Unicode is stable. Based ultimately on the ASCII standard (1963), ISO/IEC 8859–1 and other earlier encodings, Unicode is now used on almost every personal computer and mobile device. Changing which characters are represented by which numeric codes would now be so painful as not to be worth the effort. It is obvious that, for the foreseeable future, text will continue to be recorded in Unicode.

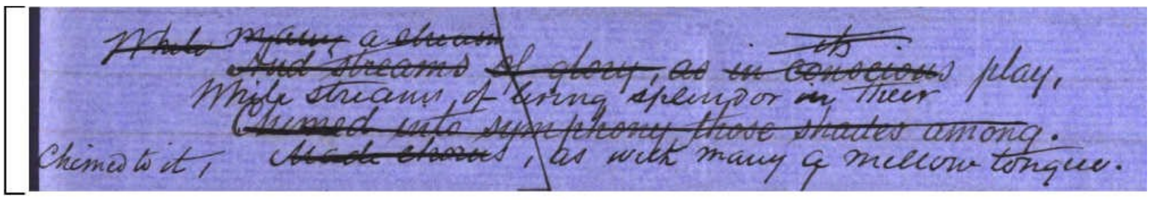
It is also clear that, to be reusable and readable, plain text must be virtually complete and coherent. Hence internal variations in the form of additions, deletions or other divergences cannot be permitted. This may seem an impossible constraint to allow, since virtually any manuscript sources, even ancient papyri and inscriptions, contain erasures and insertions, substitutions and transpositions, and even printed texts contain errors that must be corrected.

2.1 Layering

The solution, however, is quite simple. All variations, however they arise, can be assigned to separate layers, each layer forming one coherent reading of the text, even if it may be partially incomplete.

Such a separation may be disputed, since it is common practice to record internal variations using inline markup, as in TEI-XML. However, as pointed out earlier (Schmidt 2010) the biggest cause of overlap in the encoding of historical documents is textual variation. Complexity and imprecision of XML markup increase dramatically whenever changes to the underlying text exceed moderate levels. At some point of complexity, editing the TEI-XML transcription is just as likely to damage it as to improve it, especially if it is being edited collaboratively. Also, removal of markup from a text containing internal variation at a later date, or for indexing by search engines, merges the internal variants together, resulting in incoherent texts that were never written by the author, and which cannot be reliably searched. Separation of the text into coherent layers is thus necessary to allow the development of accurate and easy to use interfaces to edit historical documents. A complex example will illustrate the power of this method:

MS



~~While~~ ~~Many~~ a ~~stream~~
~~And streams~~ ~~of glory~~, ~~as~~ ~~in~~ ~~conscious~~ ~~its~~ play,
While streams of living splendor ~~in~~ their
~~Chimed into symphony those shades among.~~
Chimed to it, ~~Made chorus~~, as with many a mellow tongue.

TEI documentary

```
<l><del>And streams</del><add><del>Many</del> a <del>stream</del></add>
<del><add>While</add></del> <del>of glory,</del> <del>as</del> <del>in</del>
<del>conscious</del><del><add>its</add></del><add>While streams of living splendor in
their</add> play,</l><l><del>Chimed into symphony those shades among. </del>
<add><subst><del>Made chorus,</del><add>Chimed to it,</add></subst> as with many
a mellow tongue.</add></l>
```

TEI parallel segmentation

```
<l><app><rdg><del>And streams</del> as in <del>conscious</del> play,</rdg>
<rdg><del>Many</del> a stream of glory as in <add>its</add> play,</rdg>
<rdg><del><add>While</add></del> a <del>stream of glory in its</del> play,</rdg>
<rdg><add>While streams of living splendor in their</add> play,</rdg></app></l>
<l><app><rdg><del>Chimed into symphony those shades among.</del></rdg>
<rdg><add><del>Made chorus,</del> as with many a mellow tongue.</add></rdg>
<rdg><add>Chimed to it,</add> as with many a mellow tongue.</rdg></app></l>
```

Layers

And streams of glory, as in conscious play,
Chimed into symphony those shades among.

Many a stream of glory as in its play,
Made chorus, as with many a mellow tongue.

While a stream of glory in its play,
Made chorus, as with many a mellow tongue.

While streams of living splendor in their play,
Chimed to it, as with many a mellow tongue.

Fig. 2: TEI-XML and layering compared

There are different styles of TEI markup. The documentary method is shown immediately under the facsimile in Fig. 2. It records the deletions and additions as scribal acts, without attempting to create coherent states. Another common TEI method is parallel segmentation, used here to create four states over two lines. Both representations are close to the limit of human comprehension, and yet this is only a moderately complex example, and the XML markup has been simplified. By contrast, the layered representation is entirely readable and clear, and yet it records *exactly the same information*. The status of added and deleted text can be inferred from the absence or addition of text by comparing the layers with one another. And, if desired, the exact position of the changes on the page, often recorded in the TEI-XML, could either be added via a layer of standoff properties, inferred from the facsimile, or indicated by text-to-image links.

For the second XML representation at least, it is possible to write a program that can parse a TEI-XML file and tease apart the variants. An example is the VERSIONING MACHINE (Schreibman *et al.* 2010), which requires variants to be marked up in this way. Another example is the IMPORTER tool in Ecdosis (Schmidt 2015c), which contains a ‘splitter’, originally designed for processing the Charles Harpur texts. However, since TEI-XML markup may vary, depending on research needs and interpretation by the encoder, there are as many ways to mark up variants as there are to mark up other textual features. Hence such a program will always be a customised product.

This method resembles the way that layers of correction were marked up in the Hypernietzsche markup language (Saller 2003), and also the layering in Gabler’s Ulysses (1984, x). But it might be challenged by the fact that in modern manuscripts at least, a correction on a particular level cannot be assumed to belong to the same state of the text as another change elsewhere at the same level.

Although that is true, it is also moot. If no connection can be made between changes in one part of a document and changes in another part, due to similarity of handwriting and distance apart, putting them in the same layer is simply an expediency that does no harm. Although interpretation was exercised in deciding to which layer each change belongs, this is also true in the XML case, since by nesting or grouping corrections TEI-XML also numbers the sequence of changes implicitly. Layers can thus be viewed as simply a clearer way to store the same (interpreted) data. If a diplomatic rendition of the text is desired it can be composed just as easily from the layers as it can from the embedded markup. Although layering is not a perfect solution, it is a very useful one, and represents a great simplification over the embedded markup form.

Another objection to layering may be that it involves considerable duplication of text. Individual layers are complete and may only differ, in extreme cases, from other layers by a single letter. However, this problem can be eliminated by the use of an MVD (multi-version document) representation. The NMERGE program (Schmidt 2015b) takes a set of versions of a work, whether layers or transcriptions of separate documents, and merges them into a variant graph, as shown in Fig. 3.

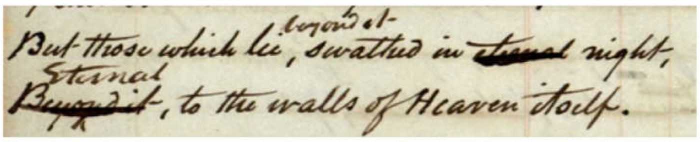
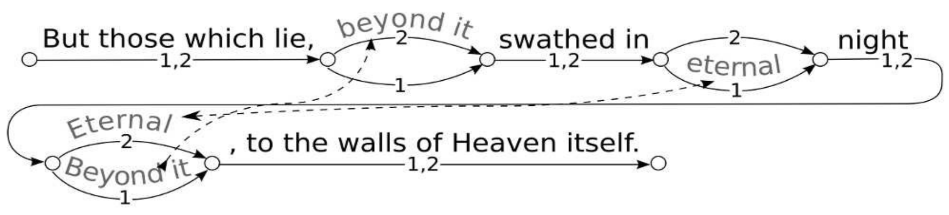
MS	
Layers	1: But those which lie, swathed in eternal night, Beyond it, to the walls of Heaven itself. 2: But those which lie, beyond it, swathed in night, Eternal, to the walls of Heaven itself.
Variant Graph	
MVD	(1,2: But those which lie),(1:),(2:1:),(1,2:, swathed in),(1:2:eternal),(2:),(1,2:night,),(1:1:Beyond it),(2:2:),(1,2:,to the walls of Heaven itself.)

Fig. 3: MS, layers, variant graph and MVD of the same text

Here the words ‘beyond it’ and ‘eternal’ are both transposed by successive deletion and insertion. The transpositions are represented by dashed arrows and by grey text in the diagram. The first time the words occur they are given an identifier: 1 for ‘beyond it’ and ‘2’ for ‘eternal’. The second time they occur in the transposed position the numbers are used to point back to the earlier instance, without repeating the text. Since parts of the text shared by other versions or layers are already merged into a single copy, this removes all duplication of text between versions[3]. The scribal status of ‘eternal’ as deleted and ‘beyond it’ as inserted, if they need to be recorded, are properties of the text and belong to the markup, not to the underlying textual structure. The MVD is shown here in a plain text format, but it is in fact machine-generated.

A further objection may be that the possible number of combinations of changes in a document is extremely large, and could not be represented by any practical number of layers. However, closer examination of manuscript sources indicates that there are in fact in each local instance of variation only a small number of possible coherent states. Layers are provided as a mechanism for recording internal variation, not as a definitive answer to all its possible forms, and exactly how many layers are chosen to represent the text is, like markup itself, a matter of interpretation. For simplicity it may suffice to record only the final state of the text or the states before and after correction; others will prefer to record all states they can discover.

When saving documents for external uses, for example, when exporting to another installation, full layers are still a useful format. They allow the encoding of texts as coherent versions using globally interoperable standards like HTML, and the extra storage requirements are today insignificant. But as an internal representation – for example for computing the differences between versions – the MVD or variant graph is more useful.

2.2 Paragraphs, lines and hyphenation

One of the most obvious characteristics peculiar to historical documents on paper, vellum, or other physical media is that they contain line-breaks, and sometimes words split over lines or even pages. Joining up split words and eliminating line-breaks may seem necessary for most digital displays, which require reflowing of the text, but throwing away this vital information about the source is usually a mistake. Without recorded line-breaks a transcription cannot be displayed meaningfully next to a facsimile of the page-image it was derived from, and the transcription loses an important fine-grained referencing system.

However, recording line-breaks means recording hyphens. Particularly in English and French hyphens are ambiguous at line-end: they may designate either a single split word, or a hyphenated compound word divided exactly at the hyphen. In TEI-XML other markup may also intervene between line-breaks, making the joining up problematic. But all this information can be derived by a carefully designed display program that takes all of these concerns into account. There is thus no need to mark hyphens with any special markup in the plain text, since the ambiguity was already present in the source document.

Paragraphs can be designated, as in Markdown, using two blank lines, or if desired, via indents at the start of a paragraph. Excess whitespace is insignificant, since it can be easily eliminated in processing, but is useful if it helps make the text readable.

2.3 Pages

Page-divisions are not essential to a plain text representation. However, if page-numbers have been added on good authority, for example, by the author, such divisions can be represented by the number of the page on a separate line with no markup of any kind – no surrounding brackets etc. This is because in standoff properties all textual properties must be represented by external markup. But if the author wrote ‘213’ in the top right hand corner of a page then that is part of the text. It may even be layered, if corrected from/to another value. Note that, in the TEI-XML scheme page-numbers are never versionable, since their content must appear in an attribute. There might be several page-numbers on a page of mixed authority. In the Harpur manuscripts we identified five types of page-number: authorial, library page-numbers, printed page numbers, the facsimile image-number and numbers assigned by the editorial project. Only one number should generally be inserted into each layer of the plain text transcription to avoid confusion, and it is up to the editors of a particular project to decide which one, if any, is to be used. Additional non-authoritative page-numbers, such as those supplied by a library or by the editor, can be represented via markup, annotations or metadata.

2.4 Residual markup in ‘plain’ text

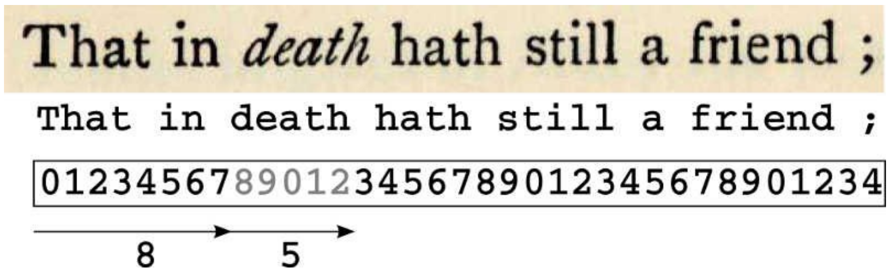
Some may object that this form of ‘plain text’ is not plain: it contains residual markup in the form of line-breaks, paragraph breaks, and perhaps indents and page-numbers (Sperberg-McQueen 1991, 35). Although this is true, the whole point of a good plain text is that it is reusable for other purposes, including reading and text-analysis. The residual markup features *do no harm*, and in fact represent useful features of the text that can be overridden, if desired, by applying explicit markup to them. For example, soft-hyphens followed by new-lines can be hidden, page numbers can be displayed outside of the text, extra white-space can be compressed or deleted etc.

2.5 Summary

In summary the design for plain text calls for line-breaks to be recorded with hyphens, paragraphs with double line-breaks or indents, and individual page-numbers inserted as needed on separate lines. Textual variation is to be represented through layering, that is the copying of entire coherent versions representing a particular state of the text at each local point. Versions are assumed to be combinable in a multi-version document or exportable to standard formats such as HTML, through combination with standoff properties.

3. Markup

Since, in the standoff properties approach, markup is stored separately from the text, there has to be some way to make it point back to the correct location in the text where it applies. To do this, a *range* is defined as a pair of numbers that record the start-location of a property and its length. Each range also has a name, for example, the range *'italics', 8, 5* might designate the 'italics' property at character position 8 of length 5, as in Fig. 4:



That in *death* hath still a friend ;
That in death hath still a friend ;
01234567890123456789012345678901234
8
5

Fig. 4: Range of property 'italics'

A set of such ranges can record all the properties that can be assigned to a text. Ranges should be sorted first by their increasing start-position and, if they are equal, by their decreasing length. In that way, longer properties will enclose smaller ones even when their start-positions coincide. Since there was no standard format for standoff properties already defined, the HRIT research group defined a format called STIL, or STandard Interval Language, expressed in JSON[4]. It is true that annotation systems might be reused as a standard way to record standoff properties, as used, for example in Earmark (Di Iorio *et al.*, 2011). However, annotation is based on the RDF standard, which uses a complex subject-predicate-object structure that is too heavyweight for the light markup needs of historical text encoding.

Since the ranges don't belong to any hierarchy, and are independent of one another, the overlapping hierarchies problem simply disappears. One set of standoff properties may thus be freely combined with another, provided that the ranges are re-sorted. This is very useful. For example, when two versions of a work are compared the results can be expressed as a set of differences in STIL form. This markup set is then added to the text's original markup when displayed, producing the left and right halves of a side by side display[5]. Another use is when displaying search results[6]. Most modern browsers do not allow discontinuous selections, but the result of a search is usually several words that need highlighting per document, and are only rarely contiguous. The selected words can be highlighted in a separate markup set, then combined with the main one to display the hits. Other potential uses include a separate markup layer for text-to-image linking (Schmidt 2015a), and further applications will doubtless emerge in future. This allows complex markup requirements to be broken down into simpler sets, which can then be recombined as needed.

3.1 Other features of STIL

Because STIL has to be able to represent markup extracted from legacy XML documents, and because it aims to be converted into formats with attributes like HTML, it must also support attributes on ranges. However, we have found that attributes are rarely needed in practice. The reason seems to be that attributes are often defined in TEI-XML to represent logical organisations of more specific properties: for example the <hi> element. <hi rend=”italics”> means no more nor less than the property ‘italics’. In other cases attributes add extra properties that could be expressed more naturally via annotations, such as ‘<person sex=”F” age=”adult”>’, or combinations of simple properties such as <stage rend=”italics”>. This is just two properties: ‘stage’ and ‘italics’. The remaining uses of attributes on ranges appear to be only to help implement some function in the software, such as providing an id or a target, rather than specifying an actual property of the text. An example is an attribute like resp=”mary” (responsibility for this feature is Mary), which can probably be represented by other programmatic means, such as a version control system, or through separating markup into layers, then attributing one layer to Mary. In other words, it is not necessary to record everything the way XML does.

Other features of STIL include the ability to have properties with no length, or to suppress content. This is used when importing TEI-XML where the header has to be suppressed. A full specification of STIL is contained in the Ecdosis site (Schmidt 2015d).

3.2 Generating STIL documents

Most STIL documents generated so far have originated from XML sources. The STRIPPER program (Schmidt 2015g), which is used by the Ecdosis IMPORTER service (Schmidt 2015c), is used to extract STIL properties from simplified XML files that have already undergone a splitting process, that is, they contain no markup specifying internal versions, and are already separated into layers.

As a result there is no internal overlap of properties within a single set. Cases like overlap between lines and speeches in a play, or line-breaks in the middle of quotations, are already resolved by splitting properties where overlap would naturally occur. Ideally one would prefer to represent overlapping properties through actual overlap. However, this imperfection is insignificant for several reasons. First, since sets of standoff properties must be rendered into HTML in most cases, overlapping properties would have to be split anyway. Second, the overlap could be created by simply combining adjacent properties that have the same names. Third, this form of overlap within a layer is not common. Most cases of overlap are between separate markup sets, and this is already catered for by the ability to merge STIL markup sets as described above. Overlap between layers is handled by NMERGE.

3.3 Layering of STIL

When importing from XML, layers have to be created to represent the different internal states of the text. Likewise, when transcriptions of separate versions are merged into one MVD, all XML markup has to be removed and stored as standoff properties in STIL format. Hence for each layer or version of plain text there will be a corresponding layer or version of STIL markup. Already in the HRIT project, both the text and the standoff properties sets were merged into separate MVDs, which were dubbed CorTex and CorCode respectively[7]. The CorTex often also included other separate versions, themselves divided into layers, and merged into a single document. In this way an entire work, or section of a work, could be represented with just two files, a CorTex and a CorCode.

Each layer or version could potentially have multiple CorCodes, since the same text could be marked up in several ways. But the default CorCode was the one originally stripped from the XML, and in future would be the version that described the text's basic formatting structure. By treating the CorCode also as a text, variations in markup alone (but not in the text) may be displayed as alterations. For example, variation between a Roman and an italic font. The FORMATTER (Schmidt 2015e) program in Fig. 6 converts one layer or version from a CorTex and a set of CorCodes into a single HTML document at a time.

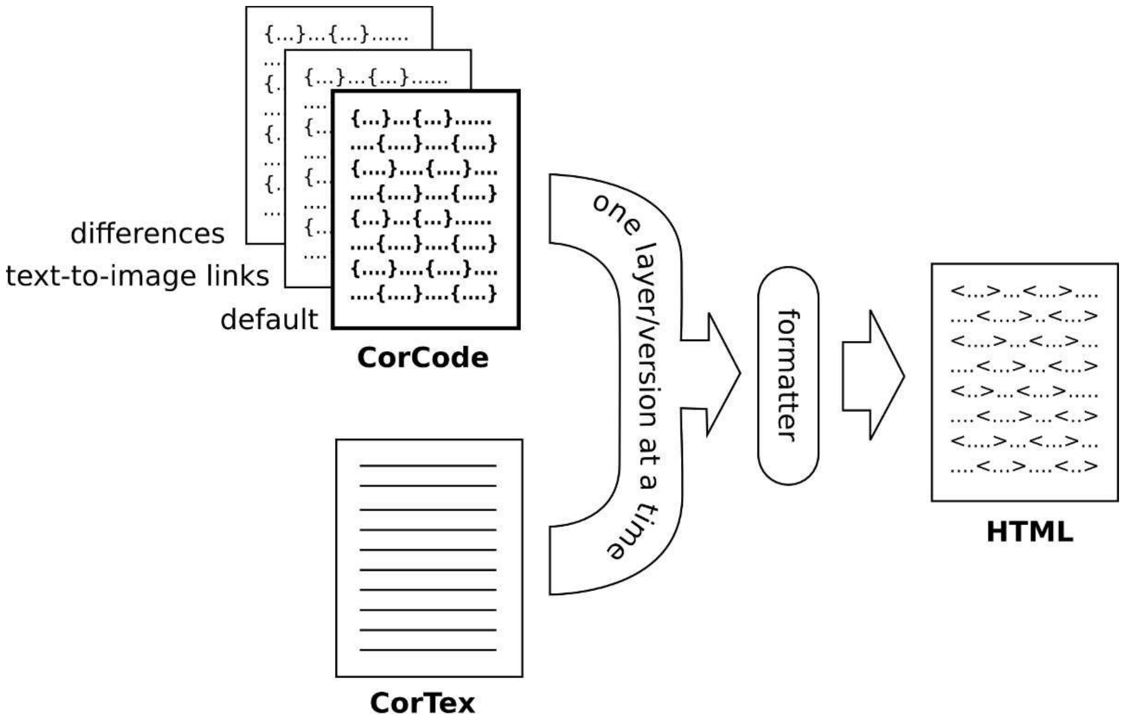
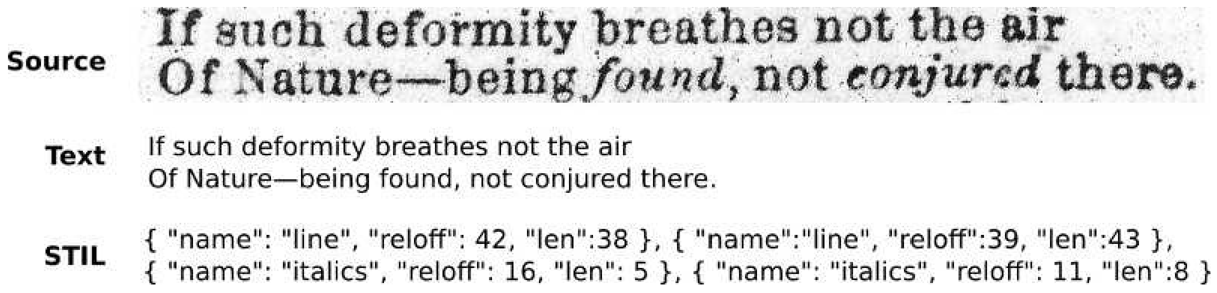


Fig. 5: Relation between CorTex and CorCode

3.4 Updating the CorCode/CorTex

The STIL markup format is designed to be updateable. Exactly how this can be done has been described elsewhere (Schmidt 2016). In a nutshell, though, the character positions in STIL are relative, not absolute. Once sorted, the position records not the distance of a property from the start of the file, but its distance to the start of the preceding property. In this way any curtailment, extension or deletion of properties can be handled by adjusting neighbouring properties, without affecting the rest of the file. This resembles the way that we edit a plain text file: the characters are like ranges of length 1 whose relative offset to the preceding character is also 1. Insertions and deletions in the file do not affect the relative offsets of other characters. The only real difference in the case of STIL, as shown in Fig. 6, is that the relative positions ('reloff') and lengths ('len') of ranges may take values other than 1. When a document is saved, all markup sets relating to it are automatically updated to the new base text.



Source

If such deformity breathes not the air
Of Nature—being *found*, not *conjured* there.

Text

If such deformity breathes not the air
Of Nature—being found, not conjured there.

STIL

```
{ "name": "line", "reloff": 42, "len":38 }, { "name":"line", "reloff":39, "len":43 },
{ "name": "italics", "reloff": 16, "len": 5 }, { "name": "italics", "reloff": 11, "len":8 }
```

Fig. 6: Relative offsetting in STIL markup

3.5 Summary

In summary, standoff properties is a technique for separating markup from text and representing it as a set of simple properties attached to ranges of text. The position of each range is specified by its distance from the start of the preceding range. Sets of standoff properties can be freely combined and properties may overlap. The next section describes how standoff properties and plain text can be reliably combined to produce HTML.

4. Generating HTML

In TEI-XML, HTML is generated via an XSLT stylesheet, but the conversion is only one-way and involves significant information-loss, since the XML sources contain four separate classes of information as described in Section 1 above. Metadata will be lost, and annotations, which are hard to represent directly in HTML, will also probably be discarded. Internal versions will be converted to formats even though they represent different states of the text. However, it is usually argued in favour of XML, that many different outputs can be produced from the same input. In practice, however, almost everyone just wants to produce HTML (Burghart and Rehbein 2012). Even eBooks are made from HTML. The ability to produce PDF for print publications can be achieved in other ways, once HTML has been obtained, for example by printing to PDF.

The advantage of standoff properties lies elsewhere, in their flexibility. Complex markup sets can be divided into separate sets, which can subsequently be combined to produce special effects, for example: the generation of differences between compared texts, search hits or text-to-image links. New markup sets can be added without disturbing the others. However, once two markup sets are combined, a potential for overlap between properties exists. How can this be reliably resolved? After all, HTML is a hierarchical markup language that has a syntax, even if standoff properties do not. The problem boils down to two sub-problems. First, how can the names of standoff properties be linked to appropriate HTML tags, and second, how can the overlap in standoff properties be resolved so that valid HTML will always result? The next two sections examine these problems separately.

4.1 Linking standoff properties to HTML

CSS (cascading stylesheets) is a globally used stylesheet standard developed by the the W3C, which specifies a set of rules for applying formatting properties to HTML (W3C 2015). It is implemented in every commercial Web-browser, to varying degrees, and is used by up to 45 billion pages on the Web (De Kunder 2015). Each CSS rule consists of a selector and some properties. Selectors specify to which elements a set of formatting instructions should be applied. Selectors may take many forms, but one of the most basic types is: *tag.class,* where *class* is the name of a markup property, and *tag* is the desired HTML tag.

To map a standoff property called ‘italics’ to the ‘span’ tag in HTML all that is needed is to write a rule, where the selector is ‘span.italics’ and the formatting instructions appear to the right:

```
span.italics {font-style: italics}
```

If there is a property called ‘italics’ in one of the markup sets, then this rule will be used to connect it to HTML span elements that will be written out as <span class=”italics”>. As a side-effect all the formats attributed to the ‘italics’ class in CSS will also be applied to the corresponding HTML elements. For most HTML pages CSS files will already exist. Either a new one can be added with the required definitions, or new rules can be added to an existing CSS file. But only rules matching the selector pattern *tag.class*, will be used. Also, markup properties without a matching CSS definition will be ignored.

If a standoff property has an attribute, this can be transferred to the HTML by using a custom CSS property-name starting with '-aese-' (as an abbreviation of AustESE). Such names are ignored by the browser because they are used to define custom formats. So for example a CSS rule like:

```
span.merged { -aese-mergeid: id }
```

specifies that the STIL property called 'merged' should be converted into a HTML <span> element, with an id-attribute equal to the value of the mergeid-attribute from the original property. This can be used, for example, in a side-by-side compare view so that text common to both sides will align across the screen as the user scrolls.

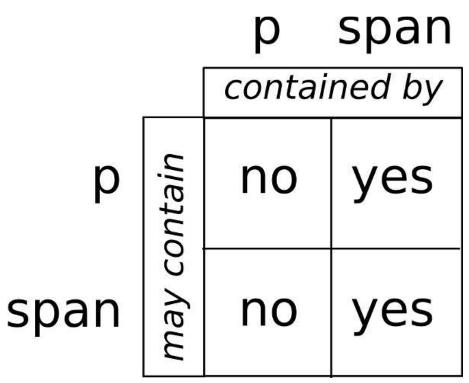
In this way standoff properties can be completely associated with their corresponding HTML elements.

4.2 Resolving overlap

The specification of STIL states that properties can overlap *arbitrarily*, although they are sorted on their start-positions in the text. HTML, on the other hand, is a context-free language that obeys strict nesting, and it seems counter-intuitive that a set of overlapping properties could be coerced into a strict hierarchy, as they must, if a STIL markup set and its underlying plain text is to be converted into HTML. However, the problem is not as hard as it first seems. One important property of STIL mentioned above is that separate markup sets can be merged, so the problem boils down to the resolution of overlap between properties in *one* markup set.

4.2.1 Hierarchy of properties

Section 4.1 above established that properties may be associated with particular HTML element-names. There are 107 element-types in HTML, and for each of these, according to the HTML5 grammar, there is a set of other elements that may appear inside them, and contrariwise each may appear inside another set of elements. So it would be possible to create a table specifying which HTML elements can be inside which. A small section of that table would look like Fig. 7.



		p	span
		contained by	
p	may contain	no	yes
span		no	yes

Fig. 7: Section of HTML containment table

A paragraph (p) may *contain* a span, but it may not appear *inside* one, nor inside itself. But a span can appear inside itself. The complete table would be a rectangle 107 by 107 to completely specify which elements may appear inside which other elements. So, by linking each property in a markup set with a corresponding HTML element, it is possible to work out how those properties are *supposed* to nest.

However, the rules of HTML alone are not enough. For example, it is probably best to convert a line of poetry to a HTML span, but then what is to be done with spans of italics or quotations? Are they inside lines or not? However, by examining the properties in the markup set it can be determined statistically which properties most often nest inside which other properties, and this majority vote can then be used to resolve doubtful cases.

4.2.2 Building a HTML tree

A context-free language like HTML can be represented as a tree. In HTML each *element* represents a piece of text or a collection of other elements. For example, a list encloses a series of list items, which are other elements, whereas a span encloses some actual text. A set of standoff properties, on the other hand, has *ranges* instead of elements. Ranges are element-like in that they may enclose text or other ranges, but ranges are not organised into a tree.

A HTML tree can be created from a set of standoff properties, by starting with an empty document tree, and then converting one standoff property to an element at a time, and placing it in the tree. After each conversion the tree represents a syntactically correct HTML document. So, once all the properties have been converted to HTML elements the entire tree must be valid.

At each conversion step the new property will likely overlap with some element already in the tree. There are five possibilities:

1. The property doesn't overlap with any element.
2. The property is completely enclosed by the element
3. The property completely encloses the element
4. The property overlaps on the left of the element
5. The property overlaps on the right of the element

In each case it can be determined from the augmented containment table described above whether the property is supposed to enclose the element or vice versa. In case 1 the property is converted to an element and added to the root of the tree. In case 2 if the element is supposed to contain the property, then the property is made its child and added to the tree. And in case 3 if the property is supposed to enclose the element then their positions are swapped, the old element becoming the child of the former property. In cases 4 and 5, and in the complementary containment situations for cases 2 and 3, either the element or the property must be split at the point(s) where they overlap.

The table is used to decide whether to split the property or the element. Split elements remain in the tree, but leftover segments of properties that overlap the element to the left or right are temporarily discarded and only reconsidered when all other properties have been placed in the tree. If they still can't be placed then these properties are discarded (but not the text they contain). This can only happen if the stylesheet designer assigned properties to the wrong HTML elements. For example, if properties associated with a <span> element were required to appear inside a list (<ul>) element, which is not allowed by HTML syntax. However, the tree is guaranteed to be valid HTML, and as long as reasonable care is exercised, all properties will be transferred to it.

5. Implementation

The FORMATTER program, outlined above, was written first in 2010 as part of the HRIT project as a commandline tool. In the Ecdosis editing system it currently takes the form of a Java service (2015e). Standoff properties are used for all texts in the Charles Harpur Critical Archive, and also for all editions being constructed on Ecdosis.

Many of the STIL files are still, unavoidably for now, derived from original XML sources. This could, however, be overcome by the development of a suitable STIL-based editor, with transcriptions being prepared in a plain-text format from the start. This desideratum has led to the development of a prototype MML (minimal markup language) editor, which will be described in a separate paper (Schmidt 2015f). In other cases it has been possible to import from other types of texts, for example the Cambridge Conrad Edition markup, which is not XML, or from plain text. In each case an import program had to be written to convert the source files into STIL and plain text. The IMPORTER service has several examples of import filters and a template to make this task easier (Schmidt 2015c).

Computing the differences between different versions of markup has not proven to be sufficiently precise in practice. As a result, none of the versions compared on the Charles Harpur site, or any of the experimental editions on Ecdosis yet use this facility. Differences are therefore computed on the basis of the text only. The problem seems to be the STIL format itself, which does not create sharp enough borders between markup changes. This will, however, be a focus of future improvements.

STIL is not proposed as a serious standard, nor does it need to be. A suitable export format would appear to be HTML, subdivided into versions and layers. The properties of the text are currently expressed via class-attributes in HTML, but this could be easily changed to an RDFa syntax to enable linked data. The separation into layers and versions enables the lossless conversion of all information directly relevant to the text in HTML form. The other classes of textual inforrmation, namely annotation and metadata, can be represented using other technologies.

FORMATTER has proven very useful in the projects it has been applied to so far, in formatting works as HTML documents online, in the side-by-side comparison tool on Ecdosis/Harpur, and also for hit-highlighting during search. We welcome other research groups that may be interested in using this tool for their projects.

6. Conclusions

In summary, the use of standoff properties constitutes a technically sound and efficient solution for rendering the content of digital historical editions in a Web environment. This solution ensures interoperability and longevity of the plain text, and also of the semantic information via HTML. Reusability is enabled by the separation of markup from text, and through the subdivision of markup into sets. The overlap problem is eliminated by the combination of multi-version documents and layering to represent versions, both internal and external, and by the use of standoff properties. The use of plain text versions, subdivided into layers, has the added benefit of recording in a simple way a great deal of information about textual variation that would otherwise require complex embedded markup.

Other advantages include the dramatic simplification of markup that this approach entails: only actual properties of text need be specified; all other former uses of markup may be handled using different techniques: such as external metadata and annotation, and external versioning of the text. The ability to combine markup sets is a key advantage of this divide-and-conquer approach. As texts become more complex in our Web 2.0, linked data world, that strategy is going to become increasingly important.

Digital humanities has now reached the point where the serious shortcomings of XML-based markup for historical documents have been widely recognised. And yet, technicians in the field still seem content to ignore the problems, and to carry on using it. However, the recent failure of the generously-funded TextGrid project to achieve any significant interoperability among hundreds of TEI-XML encoded texts (Aschenbrenner 2015) bodes ill for the building of future collaboration among digital encoding projects, unless a better form of markup can be devised. Standoff properties may seem like a radical solution, but it is one that works. Furthermore, it achieves many of the goals of markup that digital humanists crave. Is it still enough to say, as many do: ‘We know XML has its problems, but we must continue to use it because there is no alternative.’ Now there is one.

5

Notes

[1] ‘Praktisch sind der Austausch und die Nachnutzbarkeit von Forschungsdaten allerdings oft schwierig zu bewerkstelligen, … Selbst dort, wo Projekte gemeinsame Standards für Daten, Metadaten und Schnittstellen nutzen, ist Interoperabilität nicht gesichert.’ (p.202)

[2] ‘Das klingt alles sehr gut, aber es wird meines Erachtens nichts an der Tatsache ändern, dass TextGrid in einigen, wahrscheinlich wenigen Jahren verschwunden sein wird. Und das ist gut so.’ (p.33)

[3] The example has been simplified by ignoring the initial capitals of ‘Beyond’ and ‘Eternal’.

[4] This name was suggested by George Thiruvathukal at Loyola.

[5] See the compare tool on Charles Harpur The Tower of the Dream: http://charles-harpur.org/harpur/compare?docid=english/harpur/h642.

[6] See the search tool on Charles Harpur: http://charles-harpur.org/harpur/search.

[7] These terms were coined by Peter Shillingsburg.

References

W3C. (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition), Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., and Yergeau, F. (eds), W3C. http://www.w3.org/TR/REC-xml/ (accessed 7 December 2015).

W3C. (2015). Cascading Style Sheets. http://www.w3.org/Style/CSS/Overview.en.html (accessed 7 December 2015).

Aschenbrenner, A. (2015). Share It – Kollaboratives Arbeiten in TextGrid. In Neuroth, H., Rapp, A., Söring, S. (eds). TextGrid: Von der Community – für die Community Eine Virtuelle Forschungsumgebung für die Geisteswissenschaften. Glückstadt:Werner Hülsbusch, pp. 201–210.

Bauman, S. (2011). Interchange vs. Interoperability, Proceedings of Balisage: The Markup Conference August 2–5, 2011. Balisage Series on Markup Technologies, vol. 7. https://balisage.net/Proceedings/vol7/html/Bauman01/BalisageVol7-Bauman01.html (accessed 7 December 2015).

Brown, S. (2015). An entity-based approach to interoperability in the Canadian Writing Research Collaboratory, DH2015 Conference Abstracts. http://dh2015.org/abstracts/ (accessed 7 December 2015).

Burghart, M. and Rehbein, M. (2012). The Present and Future of the TEI Community for Manuscript Encoding, Journal of the Text Encoding Initiative, 2. http://jtei.revues.org/372 (accessed 7 December 2015).

Buzzetti, D. (2000). Ambiguità diacritica e markup. Note sull'edizione critica digitale. Soluzioni informatiche e telematiche per la filologia, Pavia, 30–31 March 2000. http://studiumanistici.unipv.it/dipslamm/pubtel/Atti2000/dino_buzzetti.htm (accessed 7 December 2015).

Dee, S. (2014). Learning the TEI in a Digital Environment, Journal of the Text Encoding Initiative 7. http://jtei.revues.org/968 (accessed 7 December 2015).

De Kunder, M. (2015). The size of the World Wide Web (The Internet). http://www.worldwidewebsize.com/ (accessed 7 December 2015).

DeRose, S. (2004). Markup Overlap: A Review and a Horse, Proceedings of Extreme Markup Languages. http://conferences.idealliance.org/extreme/html/2004/DeRose01/EML2004DeRose01.html (accessed 7 December 2015).

Di Iorio, A., Peroni, S., and Vitali, F., (2011). 'A Semantic Web Approach to Everyday Overlapping Markup', *Journal of the American Society for Information Science and Technology*, 62.9: 1696–716.

Durusau, P.. (2006). Why and How to Document Your Markup Choices. In Burnard, L., O'Brien O'Keeffe, K. and Unsworth, J. (eds), Electronic Scholarly Editing, New York: MLA, pp. 299–309.

Gabler, H., Steppe, W. and Melchior, C. (eds.), (1984). Ulysses: a critical and synoptic edition/James Joyce. New York and London: Garland.

Gailey, A. (2012). Cold War Legacies in Digital Editing, Textual Cultures, 7.1: pp. 5–17.

Galey, A. (2010). The Human Presence in Digital Artefacts. In McCarty, W. (ed), Text and Genre in Reconstruction, Cambridge: Open Book Publishers.

Goldfarb, C. F. (1990). The SGML Handbook. New York: Oxford University Press.

Haaf, S. , Geyken, A. and Wiegand, F. (2015). The DTA "Base Format": A TEI Subset for the Compilation of a Large Reference Corpus of Printed Text from Multiple Sources, Journal of the Text Encoding Initiative 8. http://jtei.revues.org/1114 (accessed 7 December 2015).

Hajo, C. M. (2010). The sustainability of the scholarly edition in a digital world, Proceedings of the International Symposium on XML for the Long Haul: Issues in the Long-term Preservation of XML. Balisage Series on Markup Technologies, vol. 6. doi:10.4242/BalisageVol6.Hajo01 (accessed 7 December 2015).

Hillesund, T. (2005). Digital Text Cycles: From Medieval Manuscripts to Modern Markup. Journal of Digital Information 6:1. https://journals.tdl.org/jodi/index.php/jodi/article/view/62/65 (accessed 7 December 2015).

Holmes, M. (2015). Whatever Happened to Interchange? DH2015 Conference Abstracts. http://dh2015.org/abstracts/ (accessed 7 December 2015).

Ide, N., Bonhomme, P. and Romary, L. (2000). XCES: An XML-based standard for linguistic corpora, Proceedings of the Second Annual Conference on Language Resources and Evaluation, Athens May 2000, pp. 825–30.

Ide, N. (2012). Language resource management – Linguistic annotation framework (LAF), ISO. http://www.cs.vassar.edu/~ide/papers/ISO+24612-2012.pdf (accessed 7 December 2015).

Jannidis, F. (2015). Die Zukunft Textgrids. In Neuroth, H., Rapp, A., Söring, S. (eds). TextGrid: Von der Community – für die Community Eine Virtuelle Forschungsumgebung für die Geisteswissenschaften. Glückstadt:Werner Hülsbusch, pp. 33–35.

Mueller, M. (2011). To Members of the TEI-C Board and Council, From Martin Mueller, chair, TEI-C Board. http://ariadne.northwestern.edu/mmueller/teiletter.pdf (accessed 7 December 2015).

Nelson, T., H. (1987) Dream Machines (from Literary Machines 87.1), Microsoft Press.

Osborne, R, Gerber, A and Hunter, J. (2013). eResearch Tools to Support the Collaborative Authoring and Management of Electronic Scholarly Editions, Digital Humanities Lincoln Nebbraska 16–19 July, 2013. http://dh2013.unl.edu/abstracts/ab-258.html (accessed 7 December 2015).

Pierazzo, E. 2015. Digital Scholarly Editing: Theories, Models and Methods, Farnham UK and Burlington USA: Ashgate. http://hal.univ-grenoble-alpes.fr/hal-01182162 (accessed 7 December 2015).

Piez, W. (2010). Towards Hermeneutic Markup: An architectural outline, Conference Abstracts Digital Humanities Conference 2010, Kings College London, July, 2010. http://dh2010.cch.kcl.ac.uk/academic-programme/abstracts/papers/html/ab-743.html (accessed 7 December 2015).

Renear, A., Mylonas, E. and Durand, D. (1993). Refining our Notion of What Text Really Is: The Problem of Overlapping Hierarchies, presented at the Annual joint meeting of the Association for Computers and the Humanities and the Association for Literary and Linguistic Computing, Christ Church, Oxford University, April 1992. http://cds.library.brown.edu/resources/stg/monographs/ohco.html. (accessed 7 December 2015).

Renear, A. (2000). The Descriptive/Procedural Distinction Is Flawed, Markup Languages: Theory & Practice 2:4: pp. 411–420.

Rosenthaller, L., Fornaro, P. and Clivaz, C. (2015). DASCH: Data and Service Center for the Humanities, DSH Advance Access 2015. http://dsh.oxfordjournals.org/content/30/suppl_1/i43 (accessed 7 December 2015).

Saller, H. (2003). HNML HyperNietzsche Markup Language, Jahrbuch für Computerphilologie – online 5. http://computerphilologie.digital-humanities.de/jg03/saller.html (accessed 7 December 2015).

Sanderson, R, Ciccarese, P. and Van de Sompel, H., (2013). Open Annotation Data Model. http://www.openannotation.org/spec/core/ (accessed 7 December 2015).

Schmidt, D. (2010). The inadequacy of embedded markup for cultural heritage texts, Literary and Linguistic Computing 25.3: pp. 337–356.

Schmidt, D. (2012). The role of markup in the digital humanities, Historical Social Research 37.3: 125–146. http://www.ssoar.info/ssoar/handle/document/37836 (accessed 7 December 2015).

Schmidt, D. (2014). Towards an Interoperable Digital Scholarly Edition, Journal of the Text Encoding Initiative, 7. http://jtei.revues.org/979 (accessed 7 December 2015).

Schmidt, D. (2015a). TILT 2: Text to Image Linking Tool, Digital Humanities 2015 Conference Abstracts. http://dh2015.org/abstracts/ (accessed 7 December 2015).

Schmidt, D. (2015b). NMERGENEW: The NMERGE Java library/commandline tool for making multi-version documents. https://github.com/Ecdosis/NMergeNew (accessed 7 December 2015).

Schmidt, D. (2015c) IMPORTER. https://github.com/ecdosis/importer (accessed 7 December 2015).

Schmidt, D. (2015d) STIL standoff properties format. http://ecdosis.net/main/node/15 (accessed 7 December 2015).

Schmidt, D. (2015e) FORMATTER. https://github.com/ecdosis/Formatter (accessed 7 December 2015).

Schmidt, D. (2015f) Minimal Markup Language Editor (prototype). http://charles-harpur.org/main/mml_edit?docid=english/harpur/h642 (accessed 7 December 2015).

Schmidt, D. (2015g) STRIPPER. https://github.com/ecdosis/standoff/tree/master/stripper (accessed 7 December 2015).

Schmidt, D. (2016). Using standoff properties for marking-up historical documents in the humanities *IT – Information Technology* 2016 (forthcoming). http://ecdosis.net/papers/schmidt.d.2016.pdf (accessed 7 December 2015).

Shillingsburg, P. and Hayward, N. (2010). Humanities Research Infrastructure and Tools (HRIT). http://www.luc.edu/ctsdh/researchprojects/hrit-catt (accessed 7 December 2015).

Schreibman, S., Clement, T., Daugherty, S., Hanlon, A., Whalen, R., and Singer, K. (2013). The Versioning Machine 4.0. http://v-machine.org/ (accessed 7 December 2015).

Smith, J., Deshaye, J. and Stoicheff, P. (2006). Callimachus—Avoiding the Pitfalls of XML for Collaborative Text Analysis, Literary and Linguistic Computing, 21.2: pp. 199–218.

Souter, C. (1993). Towards a Standard Format for Parsed Corpora, University of Leeds, School of Computer Studies Research Report Series, Report 93.5. https://www.engineering.leeds.ac.uk/computing/research/publications/reports/1993/1993_05.pdf (accessed 7 December 2015).

Sperberg-McQueen, M. (1991). Text in the Electronic Age: Textual Study and Text Encoding, with Examples from Medieval Texts, Literary and Linguistic Computing, 6.1: 34–46.

Sperberg-McQueen, M. (1994). Textual Criticism and the Text Encoding Initiative, Proceedings of MLA '94, San Diego. http://www.tei-c.org/Vault/XX/mla94.html (accessed 7 December 2015).

TEI (2015) P5: Guidelines for Electronic Text Encoding and Interchange, Version 2.9.1. http://www.tei-c.org/Guidelines/P5/ (accessed 7 December 2015).

Thaller, M. (1996). Text as a Data Type in ALLC/ACH’96 Conference Abstracts (Bergen: University of Bergen, 1996), pp. 252–253.

Unicode (2015). Unicode 8.0.0. http://www.unicode.org/versions/Unicode8.0.0/ (accessed 7 December 2015).