Texts as Hypergraphs: An Intuitive Representation of Interpretations of Text

Elli Bleeker, Ronald Haentjens Dekker and Bram Buitendijk
Texts as Hypergraphs: An Intuitive Representation of Interpretations of Text

Elli Bleeker, Ronald Haentjens Dekker, and Bram Buitendijk

ABSTRACT

Over the past decades, the question of what text really is has been addressed by a large number of conferences, workshops, articles, and blog posts. If there is one thing that, taken together, those contributions illustrate, it is that our understanding of text is—and has been—constantly in flux and open to many interpretations. Still, there is often a gap between how an editor conceptualizes a source text and how this text is encoded and stored on a computer: using TEI XML, editors are compelled to model their text as a single tree (a hierarchy), whether this structure corresponds with their intellectual understanding or not. Textual features that do not fit naturally into the XML data model require additional layers of code, which hinders processing, querying, and interchange. The Text-As-Graph (TAG) data model and the associated syntax TAGML are developed to express and store textual information as a network. To this end, TAG implements a hypergraph model. In the present contribution, we illustrate the benefits of TAG’s hypergraph for the modeling of features like nonlinearity, discontinuity, and overlap. In contrast to a tree model, a hypergraph accommodates these nonhierarchical structures naturally. By making them part of the data
model and the syntax, a TAGML processor can process the features without having to resort to workarounds or schema-aware tools. This lowers the difficulty of working with digital editions and facilitates querying and interchange.¹

INDEX

Keywords: markup, digital editing, genetic criticism, text modeling, overlap, data models for text

1. Introduction

1.1 What text really is

Over the past decades, the question of what text really is has been addressed by a large number of conferences, workshops, articles, and blog posts. If there is one thing that, taken together, those contributions illustrate, it is that our understanding of text is—and has been—open to many interpretations and therefore constantly in flux. The flexible and heterogeneous understanding of text is reflected by the TEI Guidelines. Using a tool like Roma, textual editors can carefully customize modules and elements to arrive at a set of encoding guidelines that corresponds with their interpretation of and research interest in the source text.² Still, the way the textual data is stored on a computer often differs from the editor’s intellectual understanding of text. That is, textual features that do not fit naturally into XML’s tree model can be adequately represented in digital form only with the help of workarounds or additional (vocabulary-specific) coding.³ The more additional coding is needed, the more complicated it will be to encode, process, or query the encoded text.

The present contribution argues that a data model that feels intuitive or natural to the user will in fact improve the user’s interaction with the data. This is also noted by Claus Huitfeldt and Michael Sperberg-McQueen, who describe the value of having a data structure that agrees in many ways with the “ideal structure . . . as it exists in some human mind” since that would significantly ease encoding praxis (Huitfeldt and Sperberg-McQueen 2003, section 2.4). Elsewhere, they write that the mechanisms for handling overlap, like concurrent markup, milestones, fragmentation, virtual elements, or standoff markup, “do not always exhibit the simplicity and clarity which make SGML and XML attractive in simpler cases” (Huitfeldt and Sperberg-McQueen 2006). In the following,
we present the Text-As-Graph (TAG) hypergraph data model and its associated syntax TAGML, and illustrate how text modeled as a hypergraph will correspond more closely to the editor’s “ideal” model. Furthermore, we propose that in addition to facilitating the text-encoding process, using a hypergraph also benefits the processing and querying of the encoded texts.

This contribution builds upon previous publications that introduced the TAG data model (Haentjens Dekker and Birnbaum 2017) and the TAGML syntax (Haentjens Dekker et al. 2018), and examined the modeling of partially ordered text (Bleeker et al. 2018, Bleeker et al. 2020). The specific objectives of the present contribution are to focus on the differences between TAGML and XML when modeling nonhierarchical structures, and to demonstrate the gains in terms of text processing. After giving our definition of text (section 1.2) and briefly describing the research field called textual genetic studies (section 1.3) from which we take our use cases, we move on to review previous work on modeling complex and nonhierarchical text structures (section 2.1). We focus on approaches that do not require the use of workarounds or local solutions. Section 2.2 then briefly outlines the relevant features of the TAG data model and the TAGML syntax. In section 3 we illustrate the difference between modeling nonlinear, discontinuous, and overlapping structures as a tree or as a hypergraph, and the advantages of a hypergraph for processing (section 4). We conclude that, XML’s current prevalence notwithstanding, it does pay to question the data models we use for text encoding. We should emphasize here that it is not our intention to merely criticize the XML data model; rather we want to show the value of questioning the prevailing standards so as to find the most suitable way to model what text really is. After all, our main focus should be on finding the best way to examine, express, query, and publish text. This requires an open, inquisitive way of looking which we hope this contribution will stimulate.

1.2 Definition of Text

Finding a suitable data model for representing text structures cannot be separated from the question of what text really is. Indeed, when researching alternative data models for text, we find that previous initiatives have been partly pragmatic—how can we tackle the limitations of the hierarchical data model implied by SGML or XML?—and partly driven by a philosophical ambition to get closer to our inherent notions and assumptions of text. This makes sense: if we use certain models long enough, they can influence the way we think and argue about text. As Patrick
In view of these philosophical and technical factors, the core of the development of the TAG model is based on detailed definitions of text and document. A document, here, is a physical object: a carrier of written text. Written text is a sequence of characters (e.g., letters, digits, spaces, and punctuation, including symbols and music notation) that is inscribed in a document. From the text, a reader derives information which is organized in a network structure. Finally, we propose that text is partially ordered. This means that it is not always possible to determine the order of all characters in the sequence. Instances of partially ordered text are nonlinear, discontinuous, or overlapping structures.

Take for instance the inline revisions that are often present in historical or literary (draft) documents like the one in figure 1. Here, the revision results in words or characters that may be placed in more than one order, meaning that the character sequence is temporarily nonlinear.

**Figure 1. An example of nonlinear text from the manuscript of Frankenstein.**

![Figure 1. An example of nonlinear text from the manuscript of Frankenstein.](image)


From the perspective of a human reader, the deleted word and the added word in the text fragment of figure 1 represent two variant readings of the character sequence. The deletion and the addition are located at the same position in the sequence and they are mutually exclusive: reading from left to right, we can read either “another task soon devolved” or “another task also devolved.” From an informational perspective, the sequence starts out fully ordered, with the characters neatly following one after another, until the point where the inline variation occurs (right after “task”). At that point, the sequence diverges into two paths or branches. Within each branch, the characters are fully ordered. At the end of the inline revision, the two branches converge and the sequence
becomes again fully ordered. Note that the characters within each branch are at the same location (index) in the character sequence: they represent two mutually exclusive variant readings of the text.

1.3 Use Case: Textual Genetic Research

As we have said, the use cases in this contribution come from the field of textual genetic studies. This type of research is concerned with the way literary works originate and develop over time. Draft manuscripts provide a great source of information, as such documents often reflect the author’s train of thought: words are crossed out, sentences are added, paragraphs transposed, etc. In other words, draft manuscripts represent traces of the writing process, and by extension the creation and development of a literary work. Since this type of information is at the core of textual genetic research, it needs to be expressed in as much detail as possible. And like most digital editors, textual genetic researchers wish to store and represent the results of their research in such a way that it can be explored by others, for instance in digital research environments or digital scholarly editions.

To adequately represent and study a work’s genesis, editors typically (1) express textual variation within one text version, (2) compare textual variation across versions, for instance by collating them, and (3) map the relationships among the various texts and documents related to the genesis of a work. To support either of these activities, a tool needs to be aware of information that is relevant for textual genesis. Ideally, this information is retained throughout the processing of the TEI XML-encoded file. As an example, let us return to the above-mentioned example of nonlinear text in the form of inline authorial revisions in draft manuscripts. Without access to information about the way such information is encoded, a collation tool will simply ignore it, making it more difficult to computationally analyze an author’s revision process. The principle applies to a wide variety of textual features, such as added phrases in which words are crossed out and other words are added, transpositions of parts of a text, several layers of marginalia, etc. As Edward Vanhoutte said in his discussion of the TEI’s potential for text modeling: “the real problems arise when dealing with modern manuscript material” (Vanhoutte 2002).
2. Background

2.1 Related Work

Scholars have been working on topics like text modeling, text encoding, and markup for decades, and they are well aware of the difficulties of representing complex textual characteristics in an effective way. By effective, we mean with few to no additional workarounds or customized technical solutions. As Fabio Vitali has argued, one can theoretically use any data model to express any kind of text, no matter how complex, as long as one is willing to use some workarounds, do some extra coding, and hand over certain tasks to other data formats (Vitali 2016). But the use of hand-overs and extra coding typically hinders the processing and analysis of the encoded text. Furthermore, it impedes human readability and makes it harder to exchange or reuse the encoded text. Full-text search provides a good example of why it is important for a processor to recognize partially ordered text and act on it. Consider a simplified TEI XML encoding of the inline revision discussed above:

Example 1. Example of inline revision.

```xml
<text>
  <!-- some markup and text --> another task <del>soon</del><add>also</add>
  devolved <!-- some more markup and text -->
</text>
```

Since in the XML data model all text and markup are typically ordered, generic XML tools will process the deletion before the addition. This results in a nonsensical sentence: “another task soon also devolved.” Without access to additional information about the nonlinearity implied by the `<del>` and `<add>` elements, a search engine will find neither the phrase “another task soon devolved” nor the phrase “another task also devolved.” It will, however, find the phrase “another task soon also devolved,” even though this phrase never existed in the manuscript. Indeed, as Desmond Schmidt noted, only ten percent of digital editions using inline markup “could find literal expressions that span inline substitutions” (Schmidt 2019, note 3).
Figure 2 presents a concise overview of the existing markup languages that model complex textual features.

### Figure 2. Features supported in markup languages.

<table>
<thead>
<tr>
<th>Markup Language</th>
<th>Explicit Hierarchy</th>
<th>Overlapping Structures</th>
<th>Discontinuity</th>
<th>Nonlinearity</th>
<th>Self-overlap</th>
<th>User-defined Datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXXML</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>EARMARK</td>
<td>yes</td>
<td>yes</td>
<td>yes (expected &quot;elements over noncontiguous ranges&quot;)</td>
<td>no (only multilinearly)</td>
<td>yes</td>
<td>depends (datatyping by means of ontologies is part of the RDF stack)</td>
</tr>
<tr>
<td>TexMECS</td>
<td>depends (supports multiple hierarchies, but it is not always clear to which hierarchy an element belongs)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>LMNL</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>FeAG</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>XCONCUR syntax</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Concurrent XML</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>TAGML</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Previous surveys of the data models for text can be found, among others, in Piez 2008, Huittfeldt et al. 2010, and Vitali 2016. The present overview is based on Vitali’s principle mentioned above: any data model can theoretically express any kind of text feature if it is complemented with workarounds or customized coding, but this is not what we should aim for when investigating the most suitable data model. Accordingly, a green cell labeled “yes” means that the feature is natively supported in the underlying data model. If a feature is supported only with the help of a hack or workaround, or in the application layer, it is taken as a “no” (represented with a red cell in the table). The following subsections focus on three features: nonlinearity, discontinuity, and overlap.

#### 2.1.1 Nonlinearity

The markup language TexMECS is designed as a linear representation for nonlinear objects, modeled as a GODDAG data structure (Huittfeldt and Sperberg-McQueen 2003). In GODDAG, all children of the markup nodes are typically ordered, but TexMECS provides a notation to mark certain markup nodes as unordered. The GODDAG processor ignores the default linear order of these elements’ children, and therefore TexMECS supports the representation of nonlinear
structures. No known working implementation of TexMECS, however, is currently available. At first glance, EARMARK (Extremely Annotated RDF Markup) also seems to support the option to represent nonlinearity: with EARMARK, users can express different linear structures using RDF statements about text fragments, and in this way it is possible to describe multiple text orders (Peroni and Vitali 2009, 4.1; Di Iorio 2009). However, multi-orderedness is not the same as partial orderedness: if a text is partially ordered, it means that (part of the) text has no order. Multi-orderedness always implies a certain order. The EARMARK specification as described in Peroni and Vitali 2009 does not natively support partially ordered text, in the sense that EARMARK users cannot mark the branching of the text stream. It is also important to note that EARMARK is a metamarkup language, which means that users encode their texts not in EARMARK but in an RDF serialization.

2.1.2 Discontinuity

Discontinuity, by which we mean the encoding of a single continuous utterance even though it is interrupted by one or more other structures, is also natively supported in TexMECS. The syntax provides a notation to suspend and resume the discontinuous markup elements (Sperberg-McQueen and Huitfeldt 2008), ensuring that the data model GODDAG considers it one single unit, even though it appears fragmented in the serialization. The same holds for the EARMARK approach: in their 2009 paper, Peroni and Vitali mention that EARMARK is able to handle situations where “non-contiguous ranges are contained by a particular markup item” (2009), which can be understood as a discontinuity situation. Evidently, there are several widely used pointer mechanisms in TEI XML to aggregate elements that belong together but are necessarily separated because of the hierarchical structure of the implied data model. For instance, elements can be linked via the @next and @prev attributes or via the <join> element with @target attributes. Still, these TEI XML mechanisms fall short when held against the criterion that the encoding method needs to be natively supported in the data model and independent of any vocabulary-specific applications.
2.1.3 Overlapping structures

Considering the amount of attention given to “solving” the overlap limitation posed by the single ordered hierarchy of XML, it comes as no surprise that this feature is supported by all alternative encoding approaches, some of which have been designed to address only the overlap constraint. Accordingly, overlapping structures are supported by the aforementioned EARMARK and TexMECS, as well as by the Concurrent XML approach, which also implements a GODDAG structure instead of the single hierarchy tree model of XML (Iacob and Dekhtyar 2005 and Iacob and Dekhtyar 2003). Another extension of XML designed to allow overlapping structures is the Multi-Colored Trees (MCT) approach of Jagadish et al. 2004. A single-ordered tree is just like an XML tree, but each colored tree defines its own local order of the nodes it contains. In the MCT approach, individual nodes can be part of multiple colored trees; as a result, one node can be part of multiple hierarchies. In addition, there is XCONCUR (Hilbert et al. 2005), an XML implementation of SGML’s CONCUR that allows encoders to express concurrent, overlapping markup hierarchies over the same text nodes (hence the name). A non-XML-based data model that allows structures to overlap is the layered markup and annotation language LMNL of Wendell Piez and Jenni Tennison (2002), which permits ranges of markup annotations in a text stream to overlap.

Finally, there exist several standoff approaches to dealing with overlapping structures (not part of the table in figure 2). The Multi-Version Document (MVD) approach, designed by Desmond Schmidt, first separates markup from the text content, and second, breaks down the text content into fragments or ranges (called standoff properties). These fragments are linked to the (set of) witness(es) in which they occur and stored in an MVD. The structure of an MVD is thus similar to that of a variant graph: it is a collection of nodes and edges in which common fragments of text are merged together and only the variant text is made explicit. All text in common between witnesses is recorded only once, and all the differences are stored as separate files. The MVD approach circumvents the challenge of dealing with overlapping hierarchies within one text by separating the multiple layers of revision in a draft manuscript and treating them as individual witnesses. Another recent standoff approach is seen in the Codex project created by Iian Neill, inspired by Schmidt’s work, which combines plain text and standoff properties stored in a Neo4J database.
2.2 Text-As-Graph

2.2.1 Data model

The following paragraphs briefly outline the relevant properties of the TAG hypergraph model and the associated markup language, TAGML. A detailed discussion of the features, properties, and constraints of the TAG data model is not within the scope of this paper, but can be found in Haentjens Dekker et al. (2018) (appendix A). The TAG definition of text (see section 1.2) has informed—and continues to inform—the design of the TAG data model and the TAG markup stack. When discussing data models for text, it is important to keep in mind that a syntax is not necessarily the same as a data model. A data model can theoretically be serialized in multiple ways, but some serializations are more expressive than others. Like XML, TAG is a data model for data models. Accordingly, TAGML is a metamarkup language that can be used to model text as a hypergraph. The TAG data model and markup stack work together to represent and process textual features in a straightforward manner. The goal is to avoid as much as possible the delegation of responsibilities to the schema, ODD, or other vocabulary-specific applications if they can or should be handled by the model.

As we have said, the underlying data model of TAG is a hypergraph. A hypergraph consists of nodes and edges just like any other graph, but with the important difference that some edges in a hypergraph can join together two or more nodes (in contrast to the one-to-one edges of regular graphs). These are called hyperedges. The regular edges in the TAG hypergraph model are directed; the hyperedges are undirected. Nodes in the hypergraph can be connected with either a hyperedge or a regular edge. The TAG hypergraph consists of five types of nodes:

- One Document node. This node serves as the root of the graph. Via a directed edge, the Document node is connected to zero or more Text nodes, Markup nodes, Branching nodes, or Annotation nodes.
- One or more Text nodes. A Text node contains textual content (UTF-8-encoded), and may be connected to one or more Markup nodes with hyperedges. It is connected to other Text nodes or Branching nodes with directed edges.
- Zero or more Markup nodes. A Markup node is connected to one or more Text nodes, and has zero or more Annotation nodes.
• Zero or more Annotation nodes. The Annotation node is connected to one or more Markup nodes or another Annotation node.
• Zero or more Branching nodes. A Branching node is connected to a Text node or another Branching node with a directed edge. It is used to mark the beginning and end of a nonlinear structure.

This variety of edges, hyperedges, and nodes ensures the flexibility of the hypergraph model.

2.2.2 TAG Markup Language (TAGML)

Again, a detailed description of the TAGML syntax is not within the scope of this paper. In the following, we briefly highlight the main features so that readers may better understand the TAGML snippets in section 3. For a full overview of TAGML’s features, we refer to Haentjens Dekker et al. 2018. A TAGML document is a text with embedded markup. Special characters are used to indicate the start and the end of a TAGML markup tag. For every open tag, there should be a corresponding closing tag; see figure 3.

Figure 3. A simple TAGML document with a markup node text and a paragraph node p containing some textual content.

[\text][p]\text{Some text here}\text{p}\text]

Like XML, TAGML is also a metamarkup language, but it models textual information as a graph (a network) instead of a tree. The edges and hyperedges in the hypergraph are created by the parser, ensuring the compactness of the TAGML syntax.
TAGML may resemble existing markup languages like XML, TexMECS, or LMNL, but TAGML is more expressive. For instance, in XML all annotation values are of type string, but TAGML offers data-typing of annotations. These data types are expressed in UTF-8 and interpreted by the TAGML parser as different data types. Encoders can distinguish between integer, string, or Boolean values (figure 4).

**Figure 4. Example of TAGML, featuring different types of annotation value.**

```xml
<poe:poem type='limerick' year=1818 rhymes=true> There was a Young Person of Smyrna [...] </poe:poem>
```

Annotations can also be nested (i.e., annotations on annotations) (figure 5).

**Figure 5. Example of TAGML featuring nested annotations.**

```xml
<poe:poem>
  <poe:tagml>
    <tagml:tagml id='Amsterdam-origin'></tagml:tagml>
  </poe:tagml>
</poe:poem>
```

Another relevant characteristic of TAGML is the layers feature. Layers are used to prevent (self-)overlapping structures within a TAGML file, for instance in the context of overlapping structures. Layers are like identifiers that group together a set of Markup nodes. The markup within each layer is hierarchically ordered. Separate layers are not in any way related to one another, but they can share both Text nodes and Markup nodes. In theory, text encoders can create as many layers as they see fit. This approach ensures that encoders can structure a TAGML document without worrying about overlap. We recognize that, conceptually, layers may be difficult to grasp, but section 3.3 presents some practical examples. See also Haentjens Dekker et al. 2018 and Bleeker et al. 2019.

In short, TAG offers encoders a versatile syntax to model multi-hierarchical, partially ordered textual structures as hypergraphs. With special notations for encoding complex textual features, TAGML is designed to make that modeling process as natural as possible. The markup language has the same compactness as XML and is independent of the user environment. Following the argument of Sperberg-McQueen and Huitfeldt and Peroni and Vitali, we did not consider it ideal to depend on vocabulary-specific search engines and other schema-aware tools to adequately process TEI XML-encoded transcriptions. For that reason, multi-hierarchical and other complex text structures are made part of the data model of TAG. This means that these structures can be
syntactically expressed in TAGML, as opposed to a schema or metadocumentation like an ODD. As a result, the schema no longer has to supplement the data model and can focus instead on schema-specific tasks.¹²

3. Encoding structural features

The sections below present a syntactical representation of structural features in TEI XML and TAGML, together with a visualization of how the encoded information is stored in the underlying data models (respectively a tree and a hypergraph). Instances of nonhierarchical structures can get very complex very fast, but we have kept the examples short and simple with a view to readability. Unless indicated otherwise, all examples in the text come from the authorial holograph of To the Lighthouse by Virginia Woolf, taken from the digital edition Woolf Online.¹³

3.1 Nonlinearity

We define nonlinearity as a characteristic of a character stream with multiple branches, the content of each branch pointing to the same location in the stream. As mentioned above, inline revision offers a good example of nonlinear text. The three figures below show different cases of nonlinearity in the text of a draft manuscript.

3.1.1 Single deletion or addition

In this example (figure 6), the author struck out the words “impossible barriers.” This means that there are two variant readings of the text: one including the deletion (“difference of opinion, impossible barriers, prejudices”) and one excluding it (“difference of opinion, prejudices”).

Figure 6. A single deletion. Fol. 27; SD p. 13.
These readings can be described as two simultaneous branches of text, one branch including the deleted characters and one branch without them. The `<del>` marks the beginning of the forking of the branches. This could be expressed in TEI XML as follows (example 2):

**Example 2. TAGML of a single deletion.**

```
<text> <!-- some text and markup --> difference of opinion, <del>impossible barriers</del> prejudices <!-- some more text and markup --> </text>
```

The TAGML notation looks quite similar (figure 7):

**Figure 7. TAGML of a single deletion.**

```
[text> ... difference of opinion, [?del> impossible barriers <?del>] prejudices... <text>
```

Note, however, that the `del` is marked as optional by preceding it with the affix `?`. The optional deletion implies nonlinearity: there are two branches of the text stream, one with the text marked up by `del` and one without. The subtle yet important difference between TEI XML and TAGML becomes clear when we visualize the encoded fragments (figures 8 and 9):
Figure 8. Visual representation of the XML tree of the single-deletion example.

Figure 9. Visual representation of the TAGML hypergraph of a single deletion.

As in a regular variant graph, the text in the hypergraph below is read from left to right, starting with the Document root node and following the directed edges. Note the branching nodes that mark the start and end of the nonlinear text. The Markup nodes labeled “text” and “del” are connected to the Text nodes by means of undirected hyperedges, visualized as colored circles.
containing the relevant Text nodes. The visualizations illustrate that in the hypergraph data model there are indeed two mutually exclusive branches of the text stream. In the XML data model, there exists but one reading of the text.

3.1.2 Immediate Revision

An immediate revision, described by the Latin term as a revision currente calamo, means that the revision is made during the first stage of writing (and not when the author later returns to the text to make corrections). The fragment in figure 10 shows how Woolf started the sentence with the word “This” and immediately corrected it to “The.” Following general opinion in textual genetic research, we understand an immediate revision as one where there is no other way of reading the text: there is no variant reading that does not include the revision. Accordingly, there is only one path through the text and that path reads “This The idea has grown . . . .” At the same time, we want to retain the information that the first few characters have been deleted.

Figure 10. An immediate revision. Fol. 7; SD p. 3; Names to be used.
It is difficult to capture the nature of a revision made *currente calamo* in TEI XML. It is usually encoded by placing an attribute on the `<del>`, such as an `@rend` with the value "immediate", an `@seq` with the value "0", or an `@instant` with the value "true" (example 3):

Example 3. Example of `<del instant='true'>
<text><del instant="true">This</del> the idea has grown . . . </text>

. Without a schema and an ODD, however, an XML processor would have no way of knowing what the attributes `@instant`, `@seq`, or `@rend` imply. In other words, it would not distinguish a regular deletion from an immediate deletion (figure 11).
In TAGML, it is possible to make this subtle distinction: \([\text{text}][\text{del}]\text{This}</\text{del}]\) The idea has grown . . . .\text{<text}>\). By omitting the affix ? on the \text{del} we indicate that the \text{del} tag is not optional: there is just one path through the text stream. Compare the visualization of the immediate deletion in figure 12 with that of the regular deletion in figure 9. In the hypergraph of the immediate
deletion there is only one branch, whereas in the hypergraph of the regular deletion there are two. This corresponds to the way we interpreted the source manuscript and encoded the text. It is not necessary to add an annotation on the \texttt{del} element to indicate that it is an immediate deletion, so the information is accessible to any TAGML parser.

### 3.1.3 Grouped Revision

A grouped revision is similar to a single deletion and a single addition: again, there are two mutually exclusive ways of reading the text: one reading includes the deleted word(s), and one reading includes the addition. We have already presented an example of a grouped revision in the introduction (figure 1); figure 13 represents another case. Here, the two words “so” and “certainly” are mutually exclusive: whether we choose the original reading “so” or the corrected reading “certainly,” they are at the same location in the text and at the same distance from the start of the sentence. If scholars interpret the deletion and the addition as belonging together semantically, they can group them together using markup. In TEI XML, this can be indicated with the \texttt{<subst>} element, whose purpose is “solely to group its child elements together, the order in which they are presented is not significant” (TEI P5, chapter 11.3.1.5).

**Figure 13. A grouped revision. Fol. 13; SD p. 6.**

The grouped revision example given above (figure 13) can be transcribed as follows in TEI XML:

**Example 4. Transcription of figure 13.**

```xml
<text>
  <!-- some text and markup --> for being <subst>
  <del>so</del>
  <add>certainly</add>
</subst>
  disagreeable <!-- some more text and markup... -->
</text>
```

The \texttt{<subst>} element functions as an indication of a split in the stream of text, which is very similar to the TAGML mechanism to encode the start of branching. We have already illustrated how the affix \texttt{?} in TAGML implies that the markup element is optional, and that using this affix splits the text stream into two branches: one branch with the markup element and any associated text, and
one branch without. To indicate that the text within two branches is semantically related, the divergence of the text stream can be flagged with `<`; the individual branches are separated with a vertical bar `|` and the converging of the branches is indicated with a `|>`. The TAGML notation of the example above would thus be as in figure 14.

Figure 14. A TAGML transcription of a grouped revision.

```
<text> ... for being <del>so</del> <add>certainly</add> disagreeable ... <text>
```
At first sight, there is little difference between the TEI XML and TAGML serializations. On the level of the data model, though, there are significant differences (figures 15 and 16):

Figure 15. Visual representation of the XML tree of a grouped revision.

Figure 16. Visual representation of the TAG hypergraph of a grouped revision.
The XML data model contains no information about the existence of two different paths through the text. When the TEI XML data is parsed by a tool without access to the schema or the ODD file, the only reading is the nonexistent “for being so certainly disagreeable.” The two branches are present on the level of the hypergraph model. Also note that all Text nodes that are directly related on a semantic level are also related in the hypergraph via a direct edge. This means that both “for being so disagreeable” and “for being certainly disagreeable” will be retrieved with a full-text search.

### 3.1.4 Other cases of nonlinearity

In the examples given so far, the branching of the text stream occurs in the written text in the source document. There are also cases in which an editor creates a nonlinear structure that is not in the source document. The TEI XML markup elements `<app>` and `<choice>`, for instance, indicate partially ordered information: they are intended to group together “a number of alternative encodings for the same point in a text” (TEI P5, Chapter 3.4). As with `<subst>`, the order in which the children of a `<choice>` element are placed have conceptually no influence on the meaning. Consider for instance the following case, taken from the TEI Guidelines, chapter 12.1.1:

**Example 5. Example of `<app>`.

```xml
<app>
  <lem wit="#El #Hg">Experience</lem>
  <rdg wit="#La" type="substantive">Experiment</rdg>
  <rdg wit="#Ra2" type="substantive">Eryment</rdg>
</app>
```

Here, the `<rdg>` elements and the `<lem>` element offer alternative readings for the same part of the text, and encoders do not consider the order in which the `<rdg>` elements are placed within the `<app>` to be informational. The same applies to the children of the `<choice>` in the example below, where “the `<sic>` and `<corr>` elements can appear in either order” (TEI Guidelines, chapter 3.4.1):

**Example 6. Example of `<sic>` and `<corr>`.

```xml
<text>. . . marginal comments which indicate that the `<choice>`
  `<corr>`dates</corr>
  `<sic>`date's</sic>
</choice> mentioned in the main body of the text are incorrect.</text>
```
Whereas the partial orderedness of both text and markup are noted in the TEI Guidelines, the children of `<subst>`, `<app>`, and `<choice>` are not stored as partially ordered in the underlying data model of XML. Again, any rules for processing `<subst>`, `<app>`, and `<choice>` and their children need to be expressed in an associated schema, which complicates further processing. Generic XML processors that do not know the schema will assume that their children are fully ordered and produce undesired results.

### 3.2 Discontinuity

Discontinuity happens when a text forms semantically a single continuous utterance, but is interrupted by other elements. The example in figure 17 is taken from a question on the TEI mailing list, and presents an interesting case in which a narrator (Marion) cites a letter she has received. Marion intersperses the citation with her own comments on the text of the letter (“wrote Ada”
and “I had told them so”). Ideally, the citation is encoded as a single expression regardless of the interruptions, so that queries for every utterance of the narrator would return either the full quotation or the quotations that are split up into more parts, depending on the editor’s query.

Figure 17. An example of discontinuity in a running text (Watanna 1916, p.171).

There are several mechanisms to express discontinuous structures in TEI XML. For example, using the @prev and @next attributes, the example of discontinuity (figure 17) would look as follows:

Example 7. Encoding of figure 17.

```xml
<text>
  <s>
    <q xml:id="1" next="#2">Dear Marion: (wrote Ada.)</q>
    <q xml:id="2" prev="#1">We are all very glad..."</q>
  </s>
</text>
```

However, each mechanism requires extensive tagging, a schema that documents the specific properties of the <q> elements, such as their @next and @prev attributes, and documentation such as an ODD file that explains what needs to be done with the <q> elements, their attributes, and the attribute values in order to correctly process the encoded text. So while specialized TEI software would be able to process the two <q> elements as part of one and the same structure, ideally a TEI XML file should be compatible with a wider variety of XML-based tools.
Users can encode discontinuity in TAGML in a more compact way that does not require generating unique values for the @ids of the <q> element. Instead, TAGML users can use the affixes - and + to indicate that a q element is paused and subsequently resumed (figure 18):

**Figure 18. A TAGML transcription of discontinuous text.**

```
[text:<s><q>"Dear Marion: -q (wrote Ada.) +q We are all very glad..."</q></s><text>
```

The visualizations of the respective data models show the difference: in the XML encoding, the sentence contains two separate <q> elements that are not connected on the level of the data model. The TAGML visualization, in contrast, shows that the Text nodes are associated with one and the same q Markup node (figures 19 and 20).

**Figure 19. Visual representation of the XML tree with discontinuous text. On the level of the data model, the quoted text is placed in two separate <q> elements.**
3.3 Overlapping textual structures

One of the most (in)famous examples of textual structures that do not fit naturally in the prevalent XML data model for text is the case of multiple and overlapping structures. As a result, simply mentioning the word overlap at a TEI conference or a Balisage Markup conference is sure to get everyone’s attention. As textual genetic scholars are as interested in the material, documentary aspects as well as in the sequential, textual aspects of a given work, they are familiar with the challenge of modeling both coexisting and overlapping text structures (see, among others, Dillen 2015, 2.3, 2.4, and 5.2; Bleeker 2017, 1.3, 2.2, and 4.2).
Figure 21 and figure 22 present a good example, found on the pages of a 1929 typescript of Sheherazade, or: Wat is liefde zonder verleiding (Sheherazade, or: What is love without seduction) by Raymond Brulez. During the revision of his own typescript, the author decided to cross out two entire paragraphs that also cross document borders. As a consequence, there are two overlapping structures: (1) the deletion of two paragraphs, the second of which (2) runs over document borders.
het keizerlijk huis geleverd worden.
In den namiddag, wanneer de felste hitte voorbij was, kwam door de luistille gangen van het paleis de uitverkorene, klopte bescheiden op de deur van de kamer waar de sultan zijn verwering gecree. Zoals de mannequin in een Parijs modeshuis den naam van het door haar gedragen toilet aan den klant opgeeft: "Afternoon Tea", "Blue Girl", "Un Caprice", of iets dergelijks, zoog de stof in de stem de lichte beweging der actrice die in de rats zit: "Koekoekswassing", "Blauwmoest" of "Voor uw genoegen" - natuurlijk in het Arabisch. Menigvuldig en soms: dat een frisse fantasie gebruikt waren de bemanningen uit het "Keizerlijk College der Weë, westerns, een dienst dat ze aankondigden kwam in de grond toch steeds op hetzelfde noer. Alzoo was Shiráz meertijd ook dit officieel geregeld minneampt ben geraakt.

Insider, wat is liefde die niet voorafgaan is door vermiljoen? Helemaal, Shiráz, afdwinging der Prefeten, krijgt dan welkust voorgeschoteld als een dagelijks kenst, waarvoor hij dan nog geen gebed veroorzaakt heeft. Hoe zoet moet wantrouw zijn dit verlies en een vrouw! De dagen van nerveuze verheugt voor de liefelijkste toestemming.

Shiráz dankt dan de probleemvrijgast in de stoogjes/van Bagdad en op de hellingen der vallei. En dan die herinnering aan verloren zomer, toen hij in
Figure 22. The quarto typescript of Sheherazade (Brulez 1927), p.4.
In TEI XML, the overlap example (section 2.1.3) could be encoded using the `<delSpan>` mechanism:

Example 8. Encoding section 2.1.3.

```xml
<text>
  <div type="page">
    <p>
      <!-- some text -->
      <delSpan spanTo="#1"/>
      <!-- some text -->
    </p>
  </div>
</text>
```

In TAGML, we can combine the mechanism to encode discontinuity with the layer functionality described in section 2.2.2. To deal with the overlap in the Brulez example, we create two layers. We use the affixes - and + on the `del` element to indicate that the deleted text runs over two pages but is part of one and the same deletion.

1. one layer for the document structure containing the pages and the deletions (text > page > del), with the layer identifier “D”;
2. one for the book structure with the paragraphs (text > p), with the layer identifier “B.”

A simplification of the TAGML transcription would look as follows (figure 23):

Figure 23. A simplified example of encoding discontinuous and overlapping structures in TAGML. The layer identifiers D and B indicate the layer to which the Markup nodes belong. The Markup nodes within each layer are hierarchically ordered, but layers can overlap. As a result, overlapping structures can be modeled straightforwardly in TAGML.

The visualizations of the underlying data structures show that it is not possible to encode in XML that the deletion spans multiple paragraphs and multiple `<div>` elements (see figure 24). The visualization of the TAG hypergraph in figure 25 shows that the Text nodes are all contained in one `del` element (indicated with the green hyperedge labeled `del`). This information is available...
at the level of the model and can be parsed and queried without additional information from the application level. In the TAGML document, users can search easily and without any workarounds for deleted sentences, pages, or paragraphs.

**Figure 24.** Visual representation of the XML tree with overlapping text structures. To an XML processor it is not clear that the `<delSpan>` element is related to the `<anchor>` element, nor that the two together indicate a deleted paragraph.

**Figure 25.** Visual representation of the TAGML hypergraph containing overlapping structures example.
By grouping the Markup nodes labeled “page” and “del” in layer “D,” and the Markup node labeled “p” in layer “B,” we can model both discontinuity and overlap. The visualization shows that there is just one Markup node labeled “del” in the hypergraph, and that this Markup node is connected to two Text nodes by means of an undirected hyperedge (visualized in green). These Text nodes are in turn associated via undirected hyperedges (in yellow) to two separate Markup nodes labeled “p” for “paragraph.” Because the p Markup nodes and the del Markup node are grouped in different layers, the fact that they overlap is not a problem. All this information is available at the level of the model and can be parsed and queried without additional information from a schema.

4. Processing

As we have hinted at more than once, the consequences of working with a data model in which nonlinear structures are idiomatically represented become most clear with processing and querying. As mentioned in section 2.1, a generic XML processor takes the text characters in a TEI XML file as fully ordered. This has, first of all, implications for full-text search (only ten percent of editions are able to retrieve literal expressions that include substitutions). By way of example, let us return to the grouped revision (figure 26):

Figure 26. A grouped revision. Fol. 13; SD p. 6.

A generic XML processor would process the word “certainly” directly after the word “so.” As a consequence, the reading “for being so disagreeable” will not exist for an XML processor, nor will the reading “for being certainly disagreeable.” The only reading that would turn up is a nonexistent one. As shown in section 3.1.3, the two distinct readings do coexist in the TAG hypergraph model: the Text nodes “so” and “certainly” are both at the same distance from the root Document node.
This would appear as such in query results. Finally, the direct relationships between the Text nodes are also stored in the hypergraph, by means of a directed edge. A full-text search of the hypergraph would therefore return both readings of the text.

The difference between processing TEI XML and TAGML also becomes clear with discontinuous structures. Let us return to the example given in section 3.2. We can think of at least two scenarios: one in which a user wants to retrieve the fragmented quotes, and one in which a user wants to retrieve all quotes together. The first would not pose a problem for TEI XML, but retrieving the disjointed quotations as one (merged) utterance would only be possible with additional, vocabulary-specific coding. Processing the two \textit{<q>} elements as a single \textit{<q>} requires a set of XSLT instructions that check the values of the \textit{@xml:id} and the \textit{@next} and \textit{@prev} attributes in order to know which \textit{<q>} elements should be stitched together. In TAGML, both scenarios would be equally straightforward. The hypergraph can be queried for the \textit{q} element(s) and their textual content as a whole, or for the \textit{q} elements that have been suspended and resumed.

Processing discontinuous structures can become quite complex. Consider the following fragment (figure 27):

\textbf{Figure 27. A discontinuous deletion. Fol. 9; SD p. 4; Start - Part 1 The Window.}

Let us focus on the deleted phrase “brought . . . so near—only a night & a sail.” Note that the words “within touch” have been inserted into the phrase. Whether they were added later or at the same time is hard to tell. But they are certainly not part of the deleted text. A simplified TAGML transcription of this text fragment would read as in figure 28.

\textbf{Figure 28. A simplified TAGML transcription of a discontinuous deletion.}

\[\text{\textit{Text the wonders to which he had looked forward}} [\textit{del-th-del}] [\textit{del-br-del}] [\textit{del-brought-del}] \text{\textit{within touch}} [\textit{del-so near}] [\textit{del-only a night \& a sail-del-text}]\]

The - and the + affixes on the \textit{del} tags serve to temporarily suspend and then resume the deletion. In the underlying hypergraph model, the deleted passages are associated with one and the same Markup node. Consequently, a simple query for all the Markup nodes labeled “del” suffices to
retrieve the deleted text “brought...so near—only a night & a sail” as one phrase. Similarly, a query for all the del elements that have been suspended and resumed would retrieve the fragmented quote.

Now consider a TEI XML transcription of the same fragment, simplified for readability:

Example 9. TEI transcription of figure 28.

```xml
<text>
<!-- some text and markup --> the wonders to which he had looked forward <del instant="true">th</del> <del instant="true">br</del> <del xml:id="del1">brought</del> within touch <del prev="#del1">so near — only a night & a sail</del><add>with</add> a dazzling, uneasy disquietude, <!-- some text and markup --></text>
```

To process the text of this fragment correctly, one needs to write a rather complicated set of XSLT instructions. At the very least, these instructions need to match the values of the @xml:id and @prev in order to process the first part of the deletion, look for the second part of the deletion, and then concatenate their textual content. At the same time, one has to prevent the second part from being processed twice (first as the second part of the deletion, and the second time together with the regular del elements). After some experimenting and consulting several XSLT specialists, we have come to no less than three different sets of instructions. And considering the ingenuity and technical expertise of the TEI community, we are quite certain there are even more ways. In short, it can be a challenging and time-consuming process to write and tweak vocabulary-specific and schema-aware tools—a daunting task for any TEI XML user who lacks a certain level of technical expertise.

5. Conclusion

The process of text encoding is a constant negotiation with the features of the data model in which the text is expressed. Of course, a data model’s technological limitations can be expanded with workarounds, additional layers of code, or the use of vocabulary-specific tools, but doing so entails several trade-offs. First, depending upon additional files to explain the tagset hinders the (blind) interchange of TEI XML files, not to mention their interoperability (cf. Bauman 2011). Second, not many textual editors can boast the required technological skills—or the funding to engage an IT specialist—to carry out complex coding tasks. As a consequence, the threshold of...
digital editing is raised. What is more, the technical aspects of data models are tightly intertwined with how we conceive of text. It is therefore crucial that we, as the text-encoding community, continue to explore how the limitations of data models influence our editing methods as well as our understanding of texts.

In this contribution, we used the presentation of the TAG data model to offer a higher-level perspective on text modeling. In section 1, we first defined written text as a partially ordered character sequence of letters, digits, spaces, and punctuation, including symbols and music notation. The textual information taken from reading and interpreting a written text can be conceptualized as a network. We illustrated the concept of partially ordered text with examples of nonlinear, discontinuous, and overlapping structures. Section 2.2 explained that while these complex, networked characteristics of text cannot be expressed idiomatically in the existing data models for text, they can be straightforwardly modeled as a hypergraph. In section 3 and section 4, we contrasted TEI XML, as the prevailing data model for text encoding, with TAGML. By visualizing the data models of both TEI XML and TAGML, we illustrated how partially ordered information is stored directly in the hypergraph model, ensuring that TAGML-encoded transcriptions can be queried by any generic TAGML processor.

The scope of the paper was necessarily limited in that it provided only simplified examples of multi-hierarchical content structures, while cultural heritage texts often present much more complicated cases, such as additions within additions, or open variants. Future developments will include a TAGML schema and ontology and further improvements of the TAG query language. In terms of usability, an editor that provides an autocomplete feature is also no luxury, nor is a workflow that includes version management. Finally, we do recognize that TAGML’s setup of a plain-text transcription with several layers of markup (i.e., annotations) pointing to the text nodes does correspond to the concept of a standoff approach. So far, development has focused on inline markup, but future work will explore the potential of standoff markup for TAG. Current work concentrates on further development of validation and autocompletion in the TAGML parser. While the TAG data model itself is still under active development, we believe that our work and findings so far may be of use to the broader text-encoding community, as it will help to broaden the discussion about text modeling.
Naturally, we are aware of the ubiquity of XML for text encoding and the broad functionalities of related X-technologies for modeling and publishing text. We are also aware that designing a new markup language involves a number of nontechnical challenges, such as training and teaching, unfamiliarity, and the (un)willingness of users to adopt new ways of editing. Nevertheless, we see much value in maintaining openness and curiosity toward alternative syntaxes for text encoding and what those may mean for solving long-standing challenges of text representation. Accordingly, we did not set out merely to find fault with the XML data model, but rather to use the TAG model as an occasion to examine some fundamental assumptions about text.

In that respect, it is worth emphasizing that TAG can already be implemented in existing (TEI XML-based) editorial workflows. When exported to TEI XML, overlapping structures in the TAGML document are automatically rendered as milestones using Trojan Horse markup (see Bleeker et al. 2020). Of course, the down-conversion from a hypergraph to a tree model inevitably implies data loss. A TAGML-to-XML export therefore requires a user to reflect on how to render complex textual features in TEI XML. In other words: what XML workarounds need to be implemented to deal with overlapping or nonlinear structures? In view of our argument for more awareness of data models for text, we do not consider this pause for reflection a major disadvantage.

BIBLIOGRAPHY


Brulez, Raymond. 1927. Sheherazade of Literatuur als Losprijs, typescript quarto. AMVC Letterenhuis B917/H2bis.


NOTES

1 The authors express their sincere gratitude to the reviewers who provided extensive and insightful feedback.

2 Dirk Van Hulle and Peter Shillingsburg use the term Orientations to Text to refer to the various research perspectives a scholar can adopt when editing text. They distinguish between material, causal (agents), temporal, genetics (inventive), performance, and aesthetic/commercial orientations (Van Hulle and Shillingsburg).
Overlapping hierarchical structures form a well-known example. In this article we will also discuss nonlinear and discontinuous structures.

The interplay between the ontological and the pragmatic sides of text modeling has been previously noted by Allen Renear et al. (1993).

Examples of digital research environments and editions specifically aimed at supporting and presenting textual genetic research include the Beckett Digital Manuscript Project (created by Mark Nixon, Vincent Neyt, and Dirk van Hulle, ongoing), the prototype of an opening from a notebook of Marcel Proust (created by Julie André, Elena Pierazzo, and Raffaele Viglianti, 2012), the genetic edition of Johann Wolfgang Goethe’s Faust (created by Anne Bohnenkamp et al.; see Brüning et al. 2013), the electronic edition of Willem Elsschot’s Achter de Schermen (edited by Peter de Bruijn et al., 2007), and the Fluid Text edition of Herman Melville’s Typee (edited by John Bryant, see). All links were last accessed on February 25, 2020 in a Mozilla Firefox browser.

To be clear, including textual features in the computational analysis is not only beneficial to textual genetic research. We focus on this research field mainly because of its focus on the wonderfully intricate inscriptions in draft manuscripts.

Only XML attributes and XML attribute values are unordered.

Recognizing the challenge of expressing literary texts as RDF statements, Barabucci et al. developed the FRETTA approach, which is designed “to express EARMARK annotations in an embedded syntax such as XML.” It is unclear, however, whether this approach has been further developed or implemented.

In software development, a stack is a group of programs that work together toward a common goal. A markup stack typically consists of a syntax, a query language, and a schema, and is closely related to the data model.

TAGML can be edited in any editor, but the open source text editor Sublime has a TAGML syntax highlighting package, and the reference implementation Alexandria can be used to parse and validate TAGML documents and store them as a TAG hypergraph.

We reason that the primary function of a schema is (or should be) to explicitly define properties of objects—such as order, cardinality, and data type—as well as their relationships.
13 The authors are grateful to and acknowledge the Society of Authors as the literary representative of the Estate of Virginia Woolf. The Woolf material may not be used for commercial purposes. Please consult the literary representative before reusing her work.

14 Posted by Joey Takeda on March 23, 2019, under the header of Another q question.

15 It is not within the scope of the present article to give a comprehensive overview of the alternative approaches to modeling overlapping structures. Those interested in this discussion will find the work of Allen H. Renear et al. a useful starting point. We can also recommend taking a look at the MLCD Overlap Corpus, which stems from the project Markup Languages for Complex Documents (MLCD), or searching for the topic overlap within the proceedings of Balisage, the markup conference.

16 The authors are grateful to Peter Boot, Vincent Neyt, and Frederike Neuber for sharing their expertise and invaluable insights.

17 Readers are referred to the TAGML documentation and encouraged to experiment with TAG and TAGML in order to explore TAG’s modeling potential for various text-encoding challenges.

18 This view has been communicated before, among others by Elena Pierazzo in her capacity as chair of the TEI board (cited in Ciotti 2018).

19 The reference implementation Alexandria offers an export function that includes a TEI XML format.

AUTHORS

ELLI BLEEKER

Elli Bleeker works as a postdoctoral researcher in the Research and Development Team of the Royal Netherlands Academy of Arts and Sciences. She specializes in digital scholarly editing and computational philology, with a focus on modern manuscripts, genetic criticism, and semiautomated collation. As a Research Fellow in the Marie Sklodowska-Curie–funded network DiXiT (2013–2017), she received advanced training in manuscript studies, text modeling, and XML technologies. She also enjoys spending time in archives looking for forgotten writers from the twentieth century.
RONALD HAENTJENS DEKKER

Ronald Haentjens Dekker is a software architect and lead engineer of the Research and Development Team at the Royal Netherlands Academy of Arts and Sciences. As a software architect, he is responsible for translating research questions into technology or algorithms and explaining to researchers and management how specific technologies will influence their research. He has worked on transcription and annotation software, collation software, and repository software, and he is the lead developer of the CollateX collation tool. He also conducts workshops to teach researchers how to use scripting languages in combination with digital editions to enhance their research.

BRAM BUITENDIJK

Bram Buitendijk is a software developer in the Research and Development team of the Royal Netherlands Academy of Arts and Sciences. He has worked on transcription and annotation software, collation software, and repository software.