Between Flexibility and Universality: Combining TAGML and XML to Enhance the Modeling of Cultural Heritage Text

Elli Bleeker[a], Bram Buitendijk[a] and Ronald Haentjens Dekker[a]

[a]*R&D group, Humanities Cluster, Royal Netherlands Academy of Arts and Sciences*

Abstract

This short paper first presents a conceptual workflow of a digital scholarly editor, and then illustrates how the smaller components of the workflow can be supported and advanced by technology. The focus of the paper is on the need to encode a historical text from multiple, co-existing research perspectives. Step by step, we show how this need translates to a computational pipeline, and how this pipeline can be implemented. The case study constitutes the transformation of a TAGML document containing multiple concurrent hierarchies into an XML document with one single leading hierarchy. We argue that this data transformation requires input from the editor who is thus actively involved in the process of text modeling.

Keywords

text modeling, text encoding, TEI XML, overlapping hierarchies, Multi-Colored Trees, editorial workflows, digital scholarly editing, TAGML, computational pipelines

1. Introduction[1]

How can we effectively model the workflow of a scholarly editor so that we can develop computational technology to support and advance this research process? This has been the overarching research question that informs the work of the R&D group of the Humanities Cluster (part of the Royal Netherlands Academy of Arts and Sciences). Considering the vastness of the question, the fact that our research is ongoing, and the limited number of pages allowed for a short paper, this contribution focuses on two smaller aspects of this question. First, how can we model different research perspectives on the same cultural heritage text? And secondly, how can we ensure that the resulting documents can be processed by generic text analysis tools? Our research takes place in the context of the Text As Graph (TAG) model, under development at the R&D group since 2017, which is set up to address several long-existing challenges for the digital editing of cultural heritage texts [8, 10, 23].

Text encoding can be considered an intellectual activity as scholars are compelled to translate their interpretation of the text into a computer-readable format. This includes selecting a data model, a markup syntax, and a specific vocabulary to encode cultural heritage documents such as literary or historical texts. The choices made here directly influence the subsequent

CHR 2020: Workshop on Computational Humanities Research, November 18–20, 2020, Amsterdam, The Netherlands

elli.bleeker@di.huc.knaw.nl (E. Bleeker)
0000-0003-2039-7300 (E. Bleeker); 0000-0002-3755-5929 (B. Buitendijk); 0000-0001-6737-7986 (R.H. Dekker)

[1]The authors are grateful for the comments of the reviewers that have helped them improve their argument and clarify their paper.

processing, querying, analysing, or repurposing of the encoded document. Ideally, then, scholarly editors base their choice for a data model and an encoding vocabulary on their research question(s), the goal(s) of the encoding project, and/or the properties of the source material. In reality, the majority of editing projects opt for XML as a data model [5]. XML is after all the *de facto* standard for text encoding and accordingly omnipresent: the Text Encoding Initiative (TEI) Guidelines [25] are currently based on XML; the family of X-languages offers a wide support in navigating, querying, and transforming XML documents; and many text editing tools take XML as input format. However, the single-rooted, fully ordered tree structure of XML offers only limited support for the modeling of cultural heritage texts (see section 2).

The questions we address in this paper – how can we enable scholarly editors to model different research perspectives on text and make the result widely available – also involve a form of knowledge dissemination among the scholarly editing community: we want the user to understand how their abstract, conceptual idea of the text corresponds with the way the computer understands that text, how the textual data transforms during the editing process, and how they may influence these transformations. To this end, we conceptualized the workflow of a scholarly editor as a step-by-step process that can be subdivided into smaller tasks or "modular components". This conceptual workflow can be quite easily translated into a computational pipeline in which the output from one step forms the input for the next. As a result, we could develop technology to address the individual components, making conscious choices about which step(s) to automate and which step(s) needed to remain a manual act because they require user input.

After briefly describing our conceptual model of the editorial workflow in section 3, we discuss how we approached the translation from workflow to data model to implementation. After introducing the data model of TAG (section 4.1), we highlight two features of the implementation: the representation of multiple perspectives on a text (section 4.2) and the export from one data format to another (section 4.3). In both cases we will indicate how these features correspond with the steps in the conceptual editorial workflow. Section 5 dives deeper into the export feature by providing the high-level description of the code flow of the export, and section 6 illustrates the transformation process with a small input and output sample.[2]

2. Related work

Probably the most exemplified limitation of XML for text modeling is the fact that the tree structure does not inherently provide for overlapping, discontinuous, or non-linear structures [11, 27, 22, 8]. These types of structures are nevertheless common in cultural heritage texts: the various research perspectives from which a text can be encoded often constitute different hierarchies that (partly) overlap [6]. Since modeling a text from different research perspectives remains a widely acknowledged objective of digital scholarly editing, several alternatives to XML for text encoding have been developed ranging from stand-off approaches to entirely new markup languages (see [11]; see also [8] for a more recent overview of these alternatives).

When using XML, scholarly editors are compelled to use local, project-specific approaches or to do significant additional coding work in order to model overlapping structures in XML.

[2]It should be noted that the implementation of the pipeline is currently on prototype-level. Still, we consider our findings thus far to be relevant for a productive discussion on conceptual models of scholarly activities, and to what extent these can be delegated to software.

As a consequence, the result is in a non-generic file format or uses proprietary software. The result will thus significantly hinder any subsequent querying and interchange of the encoded documents [19, 3]. The approach of the digital edition of Johann Wolfgang Goethe's *Faust* (2016) [4] is a good example of a local solution to modeling a text from multiple, overlapping perspectives. The entire text of *Faust* has been transcribed in TEI/XML several times. Each transcription represents a different research perspective and has a different hierarchical structure. The transcriptions are subsequently synchronized using the collation algorithm of CollateX [9], and the result is stored in a graph database similar to an MCT data structure [17]. Users can switch between perspectives in the edition's graphical user interface.

Another relevant example is presented by the *Annotated Turki Manuscripts from the Jarring Collection Online* (ATMO) project [21]. This project combines regular embedded TEI/XML markup with Trojan Horse markup [11, 21]. Trojan Horse markup elements are a specific type of milestone elements or "markers" that have a namespace definition `th:` to differentiate them from regular milestone elements. Two related markers are linked by means of matching start and end attributes, so the regular XML `<s>The sun is yellow</s>` becomes `<th:s sID="foo"/>The sun is yellow<th:s eID="foo"/>` in Trojan Horse markup. It is quite a well-known strategy to express (partly) overlapping hierarchies in XML, and various XSLT strategies exist to convert Trojan Horse to regular XML content elements and vice versa [2]. In the ATMO project, transcriptions contain three hierarchical structures, and neither structure is permanently dominant: users are able to switch between dominant structures via an XSLT template converting the Trojan Horse milestones to regular XML content elements. A third approach to representing multiple textual perspectives making use of XML and related X-technologies is *Concurrent XML* [12, 7], which represents multiple concurrent hierarchical structures in XML by dividing text into "atomic text nodes" (usually based on word division). Each node is reachable via an XPath Expression locating its place in one or more hierarchies.

Finally, there exist several stand-off systems, such as the *Just In Time Markup* system (1999-2005) in which users create "on demand user-customized versions of electronic editions of historical documents" [1], or the stand-off properties system of Desmond Schmidt and others [20]. The downsides of stand-off systems are that they often require a specialized, non-generic editing environment, and that they require a stable base text (while in practice it often is unstable and subject to changes). What is more, they usually produce quite illegible transcriptions (see [21]). So even though it may not be the perfect choice for text modeling from a computational perspective, XML's substantial role for text encoding as well as the significant amount of XML-based text processing and analysis tools makes it an undeniable community standard. Alternative text encoding approaches are therefore more likely to succeed if they take XML into account, instead of offering a completely new encoding model.

3. Conceptual model

In order to best accommodate the work of digital scholarly editors, we first mapped their research activities (see figure 1).[3] The model has three "levels": the upper level representing the intellectual activity of the editor; the middle level showing the editor's action(s) associated to this thinking; the third and lowest level indicating the product(s) of the actions. Starting at the upper left corner, for example, we can see that the research perspective of the editor influences

[3]This workflow is inspired by [26, 18, 13] among others.

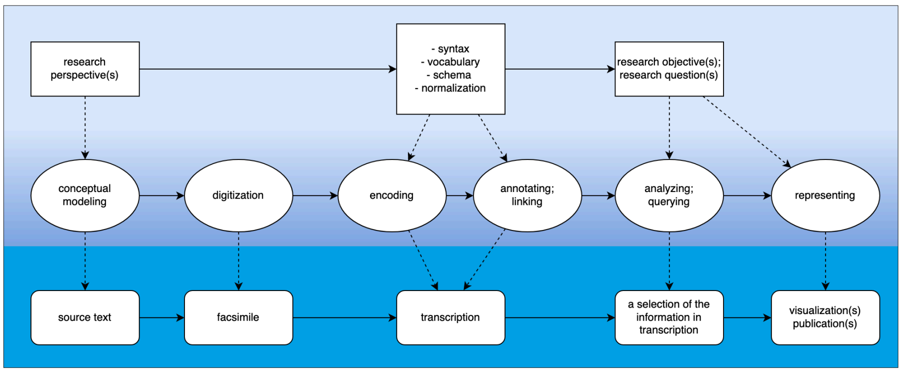

Figure 1: Visualization of the workflow of a digital scholarly editor, illustrating how the intellectual activities of the editor (upper row) affect their actions (middle row) and the output of these actions (lower row).

the conceptual modeling of the source text. When encoding, annotating, and linking the text, the editor has to make decisions about syntax, vocabulary, schemata, etc. The community's standards are also at play here, as they are with the digitization of the source text. The actions of analyzing and/or querying are again influences by the project's research objective. It will produce a selection of the information contained in the transcription, which can subsequently be represented in different ways: via a published edition of the text, a visualization, a data set, etc.

What we intend to demonstrate with this visualization, is how an editor's research perspective(s) on the source text translates to a choice for a certain data model, a syntax, a markup vocabulary, a normalization policy, etc. In turn, these choices influence the way the text can be analyzed, queried or represented. The visualization also shows the importance of an editor who knows what they want to do with the text in terms of querying, visualising, or exporting: knowing this at the outset helps them decide what would be the most suitable data format, markup strategy, and tools. The TAG data model addresses primarily the steps of *encoding*, *annotating*, *linking*, *analyzing*, and *querying* in the workflow. In the remaining of the paper, we will focus on the *encoding* step, showing how TAG allows for the encoding of multiple, co-existing research perspectives on the source text, and on how these encoded documents can be exported to XML for analysis or publication.

4. Implementation

4.1. Data model

In order to computationally support the *encoding* step in the workflow, TAG makes use of a variant of a General Ordered Directed Descent Acyclic Graph (GODDAG) [22]. The data model is based on the Multi-Colored Trees model (MCT), which permits nodes to be shared by multiple hierarchies that are distinguishable by color [15]. The TAG data model distinguishes Text nodes, Markup nodes, and Annotation nodes. In contrast to the mono-hierarchical, fully ordered tree that is implied by XML, the fact that TAG is based on a graph data model means that it can offer much more flexibility in modeling documents from different perspectives, including overlapping hierarchies and non-linear or discontinuous structures [10, 3]. As a result, we find that TAG is able to surpass the limitations of XML for text modeling.

4.2. Alexandria: from research perspectives to views

TAG's reference implementation is the text-repository system *Alexandria*. In contrast to many local or technologically complex approaches to modeling different perspectives on text (see section 2), the code of *Alexandria* is open source and designed to be implemented in any editorial workflow.[4] Users can directly encode texts in the TAG markup language TAGML [24] and upload the TAGML documents in the *Alexandria* repository. This is the functionality that corresponds to the *encoding* step in the editorial workflow. Via *Alexandria*'s command-line interface, users can subsequently query the TAGML documents (the *querying* step), or export them to other formats like PNG, SVG, DOT, and XML for analysis or publication (cf. the *analyzing* and *representing* steps).

The TAGML syntax is similar to that of embedded markup systems like XML or TexMECS [14], with a markup element consisting of an open and a close tag:

```
[s>This is an [del>easy<del] example of [add>the<add] TAGML syntax.<s]
```

Additionally, TAGML markup elements can be grouped together in sets or "layers". A layer of markup elements may be used to express a perspective on text. Each TAGML document can contain one or more layers of markup elements, and markup elements may be shared between layers. A TAGML document could for example consist of a transcription of a poem's text plus three layers of markup: one layer of markup that describes the physical aspects of the poem; one layer of markup that describes its poetic structure; and one layer of markup that describes the poem's linguistic features. Markup elements can be shared by two or more layers. In this way, it becomes easy for the editor to group together the markup elements that represent their research perspective on the text. Note the relationship between the act of encoding and the editor's research perspective as visualized in the workflow above.

Markup elements are associated with a layer by means of a layer suffix on the open tag:

```
[s|+T,+D>This is an [del|D>easy<del] example of [add|T,D>the<add]
TAGML syntax.<s]
```

In the example above, the `s` element is associated with the layer T and D; the `del` element is associated to the layer D; and the `add` element is again part of both the T and the D layer. Each layer in the TAGML is represented by a color in the MCT data model. At the time of writing, the markup elements within each layer are hierarchically ordered. This means that each color in the MCT forms one single-rooted tree. Using the layer functionality, TAGML markup elements can overlap. Figure 2 shows a visualization of the following TAGML sentence:

```
[s|+T,+D>This is an [del|D>easy [add|T>example<del] illustration<add]
of the TAGML syntax.<s]
```

A TAGML document in the *Alexandria* repository can contain many (layers of) markup elements, but it is safe to assume that editors do not always want to see *all* markup. Especially documents containing many annotations can quickly become near-illegible for human readers. So, in addition to grouping together markup elements in layers, *Alexandria* also allows users to generate a new document that contains only a selection of information from the master

[4]Interested users are invited to explore the use of the tool for their own textual material. See https://huygensing.github.io/alexandria/, last accessed September 14, 2020.

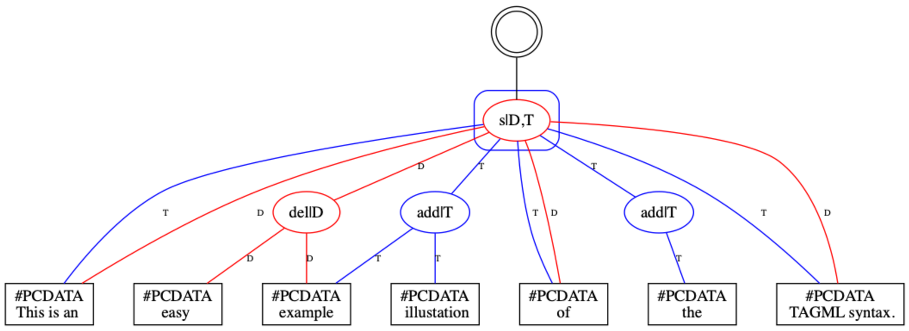

Figure 2: A visualization of the MCT of the example sentence. The two layers D and T are represented in blue and red respectively. Note that the colors show that the markup node of the **s** element is shared by both the T and the D layer; the markup node of the **del** element is only part of the D layer and the **add** element is only part of the T layer.

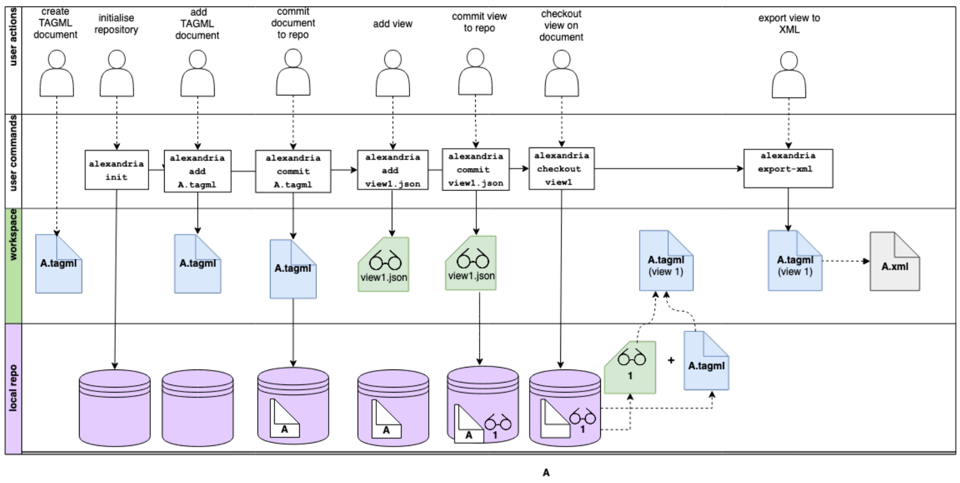

Figure 3: A visualization of a typical *Alexandria* workflow, read from left to right. The first row shows user actions, the second row the related *Alexandria* commands. The third row shows what happens in the user's work environment and the fourth and final row visualizes the text repository of *Alexandria*. In this particular workflow visualization, the user checks out view **1** on TAGML document **A** which will change the contents of the TAGML document **A.tagml** to only those markup tags and text that is defined in the view. This document is subsequently exported to XML as **A.xml**

TAGML document. The user can indicate which information they want to retain and which information can be ignored in a separate file, called a "view definition".

In *Alexandria*, views work as a filter mechanism: users indicate in a view what markup and/or layers they want to see. *Alexandria* works similar to Git: the user can use the view to "checkout" a selection of the information in the TAGML document, while the master document remains intact in the repository (see figure 3 for a visualization of this workflow). The example above showed a set of markup elements grouped together in a layer with the ID `D`. A view definition is expressed in JSON [16] and, in this example, will look as follows: `{"includeLayers":["D"]}`. The editor can also filter on specific markup elements by adding an `includeMarkup` line to the view definition: `{"includeLayers":["D"]} {"includeMarkup:["del"]"}`. Checking out a TAGML document with this view definition generates a new TAGML document containing only the selected markup elements and the selected layer(s). In this case, the checkout generates a TAGML file that contains all markup elements with the layer ID `D` plus all `del` elements. The edited document can subsequently be exported as XML, SVG, or PNG. The export files are not intended to be edited, but can be used for visualizing, analyzing, or publication purposes. This corresponds respectively with the *analyzing*, *querying*, and *representing* steps.

4.3. Exporting to XML and Trojan Horse markup

In the following paragraphs, we outline how we handle the graph-to-tree conversion of the MCT-to-XML export. From a technical perspective, transforming a graph into a tree is rather straightforward. However, since the transformation of a graph with multiple concurrent hierarchies into a mono-hierarchical tree inevitably leads to information loss, it presents some philological questions. What information in the TAGML document is relevant, what markup elements should form the leading XML hierarchy, and what information can be left out and/or does not need to be part of the leading hierarchy? Since the answers to these questions are based on the editor's research perspectives and on what they want to do with the exported XML document (e.g., analyze or visualize), they may vary from time to time.

Again, as figure 1 shows, the editor's conceptual idea of the text translates to their actions which in turn affect the output. In order to ensure that the XML output aligns with the editor's objectives, they can provide their preferences via *Alexandria*'s view feature. By indicating which hierarchy (i.e., which layer of markup elements) should be leading, the editor can coordinate the conversion of a document with several overlapping hierarchies to one single hierarchy. TAGML elements that belong to the leading hierarchy are transformed into XML content elements; TAGML elements that are not part of the leading hierarchy are transformed into Trojan Horse markup elements.

5. Computational pipeline

This section describes the export function from a higher-level perspective. The flow chart in figure 4 starts with a TAGML document and a view document that are uploaded in the *Alexandria* repository. The steps that follow are listed below:

1. The user gives the `xml export` command to the *Alexandria* server.
2. The `TAGTraverser` iterates over the MCT of the TAG document, using information from the TAG view;
3. The `TAGTraverser` generates a stream of `Events`; The `TAGTraverser` traverses over the nodes in topological order and creates `TextEvent`, `MarkupOpen` or `MarkupClose` events. A Text Node generates a TextEvent, and a Markup Node generates a MarkupOpen event of that Node. For the MarkupClose events, we need to keep track of the markup that is currently open by using stacks. For every Markup Node we add it to the relevant color stack and the global stack. Now for each Node, before we can generate a TextEvent or MarkupOpen event, we have to check the top of the relevant color stacks for Markup that is not a parent of the current Node. Those Markups generate MarkupClose events and can be removed from the color stacks and the global stack. After all Nodes have been processed, what Markup is left on the global stack generates the remaining MarkupClose events.

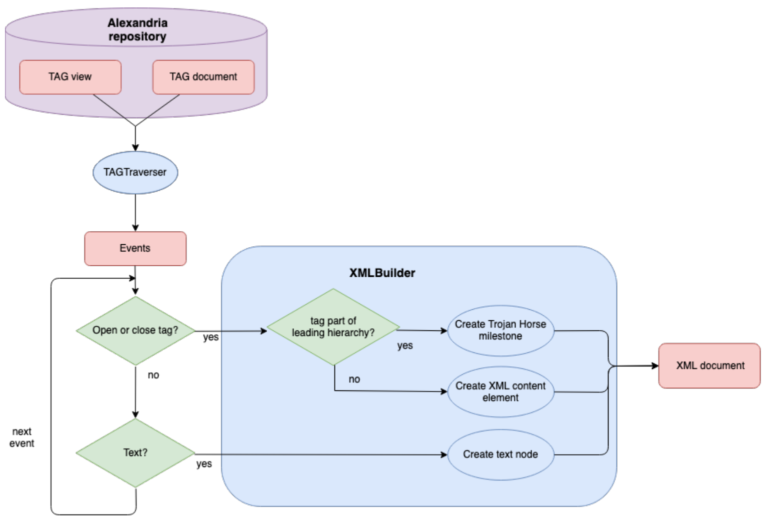

Figure 4: A flow chart visualizing the MCT-to-XML transformation. Information from the TAG view is used to traverse a TAG document and produce an XML output with one leading hierarchy.

4. For each `Event` is checked whether it is an open tag, a close tag, or text characters;
5. If the `Event` is text, the characters are transformed into an XML text node;
6. If the `Event` is an open tag or a close tag, it is checked if the the tag is part of the leading hierarchy;
   a) If not, the open tag or close tag is transformed into a Trojan Horse start or end element, respectively.
   b) If the tag is part of the leading hierarchy, the open tag or close tag is transformed into an XML open tag or an XML close tag.

6. Evaluation

By means of example, we return to the TAGML example containing overlapping markup (see section 4.2):

```
[s|+T,+D>This is an [del|D>easy [add|T>example<del] illustration<add]
of the TAGML syntax.<s]
```

Let's say that we've indicated in the view definition that the `D` layer should be leading in the XML export. The markup elements part of the `T` layer will thus be exported as Trojan Horse elements. The XML output looks as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xml xmlns:th="http://www.blackmesatech.com/2017/nss/trojan-horse" th:doc="D T">
    <s>This is an <del>easy <add th:doc="T" th:sId="add1"/>example</del>
    illustation<add th:doc="T" th:eId="add1"/> of
    <add th:doc="T" th:sId="add2"/>the
    <add th:doc="T" th:eId="add2"/> TAGML syntax.
    </s>
</xml>
```

Switching from flattened XML into hierarchically structured XML and back again is a common need in the XML community and there exists a large body of XSLT patterns designed specifically for this purpose.[5] Still, though “flattening” and “raising” the hierarchies in XML documents is by no means a novel activity, this practice usually takes place within a digital edition project, which means that the code is tailored to a specific text and rarely shared as part of the project’s output. The idea presented in this paper, by contrast, is to encourage editors to create a custom, modular workflow in which they can use both *Alexandria* and, via the TAGML-to-XMl export feature, generic XML-based tools.

While generating an XML tree from an MCT is computationally straightforward (a depth-first graph traversal), as far as we know it is unprecedented to put the user in control by including them in the conversion process, and to take this process it out of a customized, project-specific environment. As a consequence, we believe that *Alexandria* provides a more powerful and flexible approach to text modeling. This paves the way for an editorial modeling workflow that strikes the right balance between flexibility and universality.

As mentioned above, the TAG model is currently under active development. Among others, it is currently not possible for multiple users to collaborate in *Alexandria*, since the repository is now only initialized locally. We aim to allow the user of *Alexandria* to checkout and edit a TAGML document, and to upload the document again to *Alexandria*. Similar to Git, this upload first diffs the master document and the edited document. Any detected changes are subsequently merged into the master document. Such a diff and merge is quite straightforward for Git (as well as for stand-off systems), but because we want to track the edits in a TAGML document on the level of the markup as well as on the level of the text, we require a more advanced diff that is very challenging to implement. A second point of attention is that thinking about data transformations and working with *Alexandria* on the command line requires a level of technical know-how that is not available to all scholarly editors. For that reason, we are keen to provide training in TAG text modeling via workshops, summer schools, and online Jupyter Notebooks. Similar to the present contribution, these courses aim to illustrate the relationships between a conceptual, abstract model and its technical implementation(s).

7. Conclusion

This paper discussed the text modeling approach of *Alexandria* and focused on the intellectual process of translating a conceptual model to a technical implementation. Using an MCT data structure, *Alexandria* facilitates the modeling and encoding of cultural heritage text. Editors can express different research perspectives (“views”) on the source text and subsequently export a view to XML. If the XML output contains multiple overlapping hierarchies, the editor can indicate which hierarchy should be leading. The other hierarchical structures are expressed in Trojan Horse milestone elements. The XML-export function of *Alexandria* allows editors to build a custom pipeline in which they combine the best of both worlds: TAG’s flexible text modeling capacity with XML’s generic publication tools. In other words, an editorial workflow that strikes the right balance between flexibility and universality. The editor’s close involvement in every step of the text modeling process contributes to the their control over and insight into the various transformations the data undergoes.

[5][2] presents an extensive survey of at least seven possible approaches to raising flattened XML; four of which use XSLT.

References

[1] G. Barwell et al. *Authenticated Electronic Editions Project: A Progress Report.* 2001. URL: https://ro.uow.edu.au/cgi/viewcontent.cgi?article=1535&context=artspapers.

[2] D. J. Birnbaum, E. E. Beshero-Bondar, and C. Sperberg-McQueen. "Flattening and Unflattening XML Markup: a Zen Garden of XSLT and Other Tools". In: *Proceedings of Balisage: The Markup Conference.* Vol. 21. 2018. DOI: https://doi.org/10.4242/BalisageVol21.HaentjensDekker01.

[3] E. Bleeker, B. Buitendijk, and R. H. Dekker. "Marking Up Microrevisions with Major Implications". In: *Proceedings of Balisage: The Markup Conference.* Vol. 25. 2020. DOI: https://doi.org/10.4242/BalisageVol25.Bleeker01.

[4] "Faustedition". In: ed. by A. Bohnenkamp, S. Henke, and F. Jannidis. 2016. URL: http://beta.faustedition.net/.

[5] T. Bray et al. *XML 1.0.* Tech. rep. W3C, 2006.

[6] J. H. Coombs, A. H. Renear, and S. J. DeRose. "Markup Systems and the Future of Scholarly Text Processing". In: *The Digital Word: Text-Based Computing in the Humanities.* Ed. by G. P. Landow and P. Delany. 1993, pp. 85–118.

[7] A. Dekhtyar and I. E. Iacob. "A Framework for Management of Concurrent XML Markup". In: *Data & Knowledge Engineering* 52.2 (2005), pp. 185–208.

[8] R. H. Dekker and D. J. Birnbaum. "It's More Than Just Overlap: Text As Graph". In: *Proceedings of Balisage: The Markup Conference.* Vol. 20. 2017. DOI: https://doi.org/10.4242/BalisageVol19.Dekker01.

[9] R. H. Dekker and G. Middell. *CollateX.* 2019. URL: https://collatex.net/.

[10] R. H. Dekker et al. "TAGML: a Markup Language of Many Dimensions". In: *Proceedings of Balisage: The Markup Conference.* Vol. 21. 2018. DOI: https://doi.org/10.4242/BalisageVol21.HaentjensDekker01.

[11] S. J. DeRose. "Markup Overlap: A Review and a Horse". In: *Proceedings of the Extreme Markup Languages.* 2004.

[12] P. Durusau and M. O'Donnell. "Implementing Concurrent Markup in XML". In: *Extreme Markup Languages.* Vol. 2001. 2001.

[13] R. Hoekstra and M. Koolen. "Data Scopes for Digital History Research". In: *Historical Methods: A Journal of Quantitative and Interdisciplinary History* 52.2 (2019), pp. 79–94.

[14] C. Huitfeldt and C. Sperberg-McQueen. *TexMECS: An Experimental Markup Meta-Language for Complex Documents.* 2001. URL: http://www.hit.uib.no/claus/mlcd/papers/texmecs.html.

[15] H. Jagadish and L. Lakshmanan. "Colorful XML: One Hierarchy Isn't Enough". In: *Proceedings of the 2004 ACM SIGMOD international conference on management of data.* ACM, 2004. DOI: https://dl.acm.org/doi/abs/10.1145/1007568.1007598.

[16] JSON. *JavaScript Object Notation (JSON).* 2007. URL: https://www.json.org/json-en.html.

[17] G. Middell. *On the Value of Comparing Truly Remarkable Texts.* Presented at the symposium "Knowledge Organization and Data Modeling in the Humanities". 2012. URL: https://datasymposium.wordpress.com/middell/.

[18] M. Rehbein and C. Fritze. “Hands-on Teaching Digital Humanities: a Didactic Analysis of a Summer School Course on Digital Editing”. In: *Digital Humanities Pedagogy. Open Book Publishers. Retrieved from http://www. openbookpublishers. com* (2012).

[19] D. Schmidt. “A Model of Versions and Layers”. In: *DHQ: Digital Humanities Quarterly* 13.3 (2019).

[20] D. Schmidt and R. Colomb. “A Data Structure for Representing Multi-version Texts Online”. In: *International Journal of Human-Computer Studies* 67.6 (2009), pp. 497–514.

[21] C. Sperberg-McQueen. “Representing Concurrent Document Structures Using Trojan Horse Markup”. In: *Proceedings of Balisage: The Markup Conference*. Vol. 21. 2018. DOI: https://doi.org/10.4242/BalisageVol21.Sperberg-McQueen01.

[22] C. Sperberg-McQueen and C. Huitfeldt. “GODDAG: A Data Structure for Overlapping Hierarchies”. In: *Lecture Notes in Computer Science*. Ed. by P. King and E. Munson. Vol. 20-23. Berlin: Springer-Verlag, 2000.

[23] TAG. *Text As Graph*. Version alexandria 2.3. 2019. URL: https://huygensing.github.io/TAG/.

[24] TAGML. *Text as Graph Markup Language*. 2019. URL: https://github.com/HuygensING/TAG/tree/master/TAGML.

[25] TEI-Consortium. */TEI P5: Guidelines for Electronic Text Encoding and Interchange*. Version version 4.0.0. 2019. URL: http://www.tei-c.org/Guidelines/P5/.

[26] J. Unsworth. “Scholarly Primitives: What Methods Do Humanities Researchers Have in Common, and How Might our Tools Reflect This”. In: *Symposium on Humanities Computing: Formal Methods, Experimental Practice. King’s College, London*. Vol. 13. 2000, pp. 5–00.

[27] A. Witt. “Multiple Hierarchies: New Aspects of an Old Solution”. In: *Proceedings of the Extreme Markup Languages*. 2004. URL: http://www.mulberrytech.com/Extreme/Proceedings/html/2004/Witt01/EML2004Witt01.html.