Introducing Citation Structures

Hugh Cayless, Thibault Clérice, Jonathan Robie

Balisage
The Markup Conference

Balisage Paper: Introducing Citation Structures

Hugh Cayless

Thibault Clérice

Jonathan Robie

Balisage: The Markup Conference 2021
August 2 - 6, 2021

How to cite this paper

Cayless, Hugh, Thibault Clérice and Jonathan Robie. “Introducing Citation Structures.” Presented at Balisage: The Markup Conference 2021, Washington, DC, August 2 - 6, 2021. In *Proceedings of Balisage: The Markup Conference 2021*. Balisage Series on Markup Technologies, vol. 26 (2021). https://doi.org/10.4242/BalisageVol26.Cayless01.

Abstract

Text Encoding Initiative documents are notoriously heterogeneous in structure, since the Guidelines are intended to permit the encoding on any type of text, from tax receipts written on papyrus to Shakespeare plays or novels. Citation Structures are a new feature in the TEI Guidelines that provide a way for documents to declare their own internal structure along with a way to resolve citations conforming to that structure. This feature will allow systems ike the Distributed Text Services (DTS) API, which process heterogeneous TEI documents to handle tasks like automated table of contents generation, the extraction of structural metadata, and the resolution of citations without prior knowledge of document structure.

TEI documents often have an internal structure that is central to the way they are used, but because TEI can represent any kind of text, there is no one standard internal structure. Until recently, there was not a consistent way to declare their structure. This caused obvious problems for any software system needing to provide generic support to document structures encoded in TEI. Distributed Text Services (DTS) is an API for reading and querying collections of TEI-encoded texts.[1] DTS allows users to browse text collections, retrieve lists of citable structures for documents within those collections, and resolve citations to parts of the documents. A DTS client has no a-priori knowledge of these

structures, it must be able to discover them while browsing and querying a given work. As developers and implementers of the standard, we needed a way to allow TEI documents to declare their own internal structures, so we proposed a new element, `<citeStructure>`, that can unambiguously declare one or more mechanisms for citing a work encoded in TEI and also associate metadata with structural levels, using the `<citeData>` element. Our proposal was accepted and merged into the TEI Guidelines for the 4.2.0 release.[2]

The problem we faced will become clear immediately if we consider some of the types of text one might wish to deploy DTS for. The Digital Latin Library aims to produce critical editions of Latin texts from the Classical, Mediaeval, and Renaissance eras. In such collections, it would be unsurprising to find works organized by chapter, paragraph, and sentence next to ones cited by book and line number, or book, poem, and line number, or just poem and line number. An edition of Calpurnius Siculus's bucolic poetry contains seven poems of around 100 lines each, and is therefore cited by poem and verse line. A work of prose, like Livy, on the other hand, would be organized by book, chapter and sentence.

One of DTS's main features is the ability to resolve a citation to a chunk of text (e.g. the first five lines of book one of the Iliad). In TEI, these citations will map onto `<div>`, `<p>` or `<ab>` (text block), `<l>` (verse line), `<seg>` (arbitrary segment) containers, and also possibly empty milestone tags, like `<lb/>` (line beginning) or `<milestone/>`. Worse yet, these internal structures may vary even within the same work. A critical edition typically contains introductory materials in prose, for example. For a system to decide how to resolve citations without additional information is not a trivial task.

TEI has long had a mechanism for converting “canonical” citations to resolvable URIs. The `<cRefPattern>` element has a `@matchPattern` attribute and a `@replacementPattern` attribute. The first contains a regular expression and the second a URI containing backreferences to capturing groups in the regular expression. Using this mechanism, a reference like “3.1.2” can be converted to, e.g. a URL fragment identifier `#b3-p1-l2` or an XPath pointer `#xpath(//div[@n='3']/div[@n='1']/l[@n='2'])`. This system lacked some features DTS systems would need, however. DTS doesn't only resolve citations, it also generates them. A DTS system can, for example, tell you what the citable elements are in Book 3. Or give you a table of contents for a work. It is easy to produce such a thing from a TEI document as long as you understand the internal structure you are working with. Getting a list of all the elements with `@type` “chapter” and their headings from a TEI document is easy as long as you already understand the internal organization of that document. Figuring it out without being told the rules first is tricky.

What we needed then, was a mechanism to allow TEI documents to declare their own internal organization. Under the old method, it was (more or less) possible to do this using a combination of <cRefPattern>s and a moderate abuse of the <refState> element. The Perseus Digital Library's reference declaration for Julius Caesar's Civil War, for example, does this:

```
      <encodingDesc>
        <refsDecl n="CTS">
          <cRefPattern n="Section" matchPattern="(\w+).(\w+).(\w+)"
            replacementPattern="#xpath(/tei:TEI/tei:text/tei:body/tei:div/tei:div[@n='$1']
/tei:div[@n='$2']/tei:div[@n='$3'])">
            <p>This pointer pattern extracts Book and Chapter and Section</p>
          </cRefPattern>
          <cRefPattern n="Chapter" matchPattern="(\w+).(\w+)"
            replacementPattern="#xpath(/tei:TEI/tei:text/tei:body/tei:div/tei:div[@n='$1']
/tei:div[@n='$2'])">
            <p>This pointer pattern extracts Book and Chapter</p>
          </cRefPattern>
          <cRefPattern n="Book" matchPattern="(\w+)"
            replacementPattern="#xpath(/tei:TEI/tei:text/tei:body/tei:div/tei:div[@n='$1'])">
            <p>This pointer pattern extracts Book</p>
          </cRefPattern>
        </refsDecl>
        <refsDecl>
          <refState unit="book" delim="."/>
          <refState unit="chapter" delim="."/>
          <refState unit="section"/>
        </refsDecl>
      </encodingDesc>
```

[3]

The first `<refsDecl>` above lists mechanisms for extracting books, chapters, and sections from the edition using XPaths and regular expression replacements, so that a reference “1.2” can be mapped to an XPath `/tei:TEI/tei:text/tei:body/tei:div/tei:div[@n='1']/tei:div[@n='2']`, which would retrieve that portion of the document. The second `<refsDecl>` gives a structural map of the document and specifies how shorthand references are constructed. It is divided into book, chapter, and section, and references are to be delimited with periods. This is possibly an abuse of the `<refState>` feature, which was intended to reference milestone tags in the source, marking section divisions.[4] But this awkwardness is present exactly because the TEI Guidelines did not provide Perseus a way to declare the structure of their documents to their document processing system. Perseus needed this functionality because of the heterogeneity of its corpus. An analysis of its dataset shows 33 distinct document organization schemes across 609 classical texts in its Latin collection.[5]

The solution the authors proposed to the TEI Technical Council last year was Citation Structures. Citation Structures combine the ability to match references with the ability to declare structure, and do not require a knowledge of regular expression syntax in order to function. They are also more concise. The same ideas expressed in the example above can be rewritten:

```
      <citeStructure unit="book" match="//div" use="@n">
        <citeStructure unit="chapter" match="div" use="@n" delim=".">
          <citeStructure unit="section" match="div" use="@n" delim="."/>
        </citeStructure>
      </citeStructure>
```

To explain what's happening here in a bit of detail, the root `<citeStructure>` has a `@match` attribute that provides an XPath locating the elements corresponding to a root-level citation and a `@use` attribute, also an XPath, relative to `@match`'s XPath, that gives access to the citation value. `@delim` gives a string that separates levels

in a citation, such as the "." in "3.1". `@unit` is optional and can be used to give a label to the citation level.

Using this structure, we could map the citation “3.1.2” to an XPath `//div[@n='3']/p[@n='1']/seg[@n='2']` by splitting on the strings in the `@delim` attributes and then constructing the XPath using the information in the `@match` and `@use` attributes. An algorithm for doing this is given in the documentation on Citation Structures.[6]

Citation Structures are not limited to resolution. We could use the structure above to generate a list of resolvable citations or a table of contents. The new structure also provides a mechanism for associating data with different citation levels using the <citeData> element. If our example above provided book headings (via a `<head>` element), for example, we could rewrite our example thus:

```
<citeStructure unit="book" match="//div" use="@n">
  <citeData property="http://purl.org/dc/terms/title" use="head"/>
  <citeStructure unit="chapter" match="div" use="@n" delim=".">
    <citeStructure unit="section" match="div" use="@n" delim="."/>
  </citeStructure>
</citeStructure>
```

Now we can get title metadata for the books in the work by getting the content of the `<head>` element. This would be the link text for our table of contents, or metadata useful in navigating the document via DTS.

TEI documents need not have a consistent internal structure. Take Ovid's Tristia, for example, where books 1,3, 4, and 5 are composed of several poems, but book 2 is one long poem. Assuming the document uses TEI `<div>`s for books and poems, and `<l>` elements for lines, we could use a Citation Structure like:

```
<citeStructure unit="book" match="//div" use="@n">
  <citeStructure unit="poem" match="div" use="@n" delim=".">
    <citeStructure unit="line" match="l" use="@n" delim="."/>
  </citeStructure>
  <citeStructure unit="line" match="l" use="@n" delim="."/>
</citeStructure>
```

Here, the structure gives us an alternative at level two. Either we will find poems or lines (if we're in book 2). This too is an improvement over the previous mechanism, which relied on a flat list of possible matches.

We regard the new Citation Structure feature of TEI as a good start in helping TEI documents to play well in Linked Open Data systems. We do not think it is necessarily complete nor perfect, and would welcome suggestions for improvements. Some areas for further development include the question of whether a typology of structural types should be developed. Such a typology might allow for more intelligent document querying and processing. A related issue is the development of best practices around structural metadata: since different structural elements may be put to different uses (e.g. tables of contents, chunking large documents, and citation resolution), what properties should we use to mark these differences? We hope the answers to these questions will emerge as we implement systems using Citation Structures.

Turning to the practicalities, let's look at some concrete examples: we're

revamping the Digital Latin Library website, with an eye toward making publication workflows based on the content of a Git repository. Editions on the site have the following requirements:

1. They should be split up into manageable sections rather than delivered as one long page.
2. They should have a table of contents, allowing easy navigation between sections.
3. They should allow for the resolution of references (e.g. take me to poem 3, line 5).

Starting with the first edition the DLL published, Calpurnius Siculus's Bucolica, we'll walk through how to accomplish these tasks using citation structures. A basic citation structure for the edition will look like this:

```
<refsDecl>
  <citeStructure match="//front/div[@type='introduction']" use="'Introduction'">
    <citeStructure match="div[@type='section']" use="head" delim=", " unit="section">
    </citeStructure>
  </citeStructure>
  <citeStructure match="//front/div[@type='bibliography']" use="head">
    <citeStructure match="div[@type='bibliography']" use="head" delim=", " unit="section">
    </citeStructure>
  </citeStructure>
  <citeStructure match="//body/div[@type='edition']" use="head">
    <citeStructure match="div[@type='textpart']" use="@n" delim=" " unit="poem">
      <citeStructure match=".//l[parent::div or parent::lem]" use="@n" delim="." unit="line"/>
    </citeStructure>
  </citeStructure>
</refsDecl>
```

The edition has an introductory section with several sub-sections, a bibliography, likewise split into manuscripts, previous editions, etc., and the main text, split into poems. Poems are the things we're most likely to want to actually cite, and those are (unsurprisingly) composed of verse lines.

The first problem we face is figuring out how the document should be divided up. We need some way to mark sections as "chunkable." There are some sub-requirements here:

1. we probably wouldn't want to have, e.g., a document per line, but we also wouldn't want both the entire introduction and each separate section of the introduction to be chunks.
2. There are some interdependencies between sections as well. Most sections, and in particular the poems, rely extensively on things that are defined in the bibliography, so the bibliography has to travel along with each chunk as well as being a chunk in its own right.

We therefore will need to mark citation structures in a variety of ways. The implementation gives us a way to do this, using `<citeData>` elements. Recall that these permit you to map a URL (perhaps an RDF property, for example, though it could be a property defined within a TEI taxonomy, etc.) to an XPath that will return the value(s) of that property from the document. So what properties should we use?

We need to be able to mark structures as pieces to be split into separate

documents and we need to mark them as items to be linked in the table of contents. Dublin Core (DC)[7] is a generic metadata scheme that often serves well, but it doesn't seem to have an appropriate property, unfortunately. We could repurpose TEI attributes like @type and @n for this, but that means mixing communication strategies and also runs into some limitations. TEI @type isn't multi-valued, for one thing, and we'll certainly have sections that should both be split out and should appear in the table of contents. We could also decide that certain unit types are to be treated differently. In this example, the citeStructures that mark splittable sections have @unit with the values "section" or "poem". So we could decide that certain unit types automatically get split.

DC does have a "type" property, but unfortunately it means something completely different. The RDF Schema "label" property is tempting, but for now, let's just define a local property and call it "function". We can use "chunk" and "toc-entry" as values. DC does have a "requires" property, and we can use that to indicate sections that should travel along with a section to be split.

After adding the new properties, we have this:

```
<refsDecl>
  <citeStructure match="//front/div[@type='introduction']" use="'Introduction'">
    <citeStructure match="div[@type='section']" use="head" delim=", " unit="section">
      <citeData property="#function" use="'chunk'"/>
      <citeData property="dc:requires" use="//front/div[@type='bibliography']"/>
    </citeStructure>
  </citeStructure>
  <citeStructure match="//front/div[@type='bibliography']" use="head">
    <citeStructure match="div[@type='bibliography']" use="head" delim=", " unit="section">
      <citeData property="#function" use="'chunk'"/>
    </citeStructure>
  </citeStructure>
  <citeStructure match="//body/div[@type='edition']" use="head">
    <citeStructure match="div[@type='textpart']" use="@n" delim=" " unit="poem">
      <citeData property="#function" use="'chunk'"/>
      <citeData property="dc:requires" use="//front/div[@type='bibliography']"/>
      <citeStructure match=".//l[parent::div or parent::lem]" use="@n" delim="." unit="line"/>
    </citeStructure>
  </citeStructure>
</refsDecl>
```

This satisfies the requirements for splitting the sections out into separate documents. Now let's see what we need to do to build the table of contents. The table of contents will need:

1. Names for each referenced section
2. Links to that section
3. A way to indicate that the section should appear in the table of contents

The name is an interesting problem. Poem 1 will be cited as something like Bucolica 1, but it has a title in the edition, "Poem 1. [Corydon, Ornytus]", which is what we'll want to appear in the ToC. The poem's opening tag looks like this:

```
<div type="textpart" n="1" xml:id="poem1">
```

The convention DLL uses is to label citable elements with the @n attribute, and so this is what we see in the corresponding citation structure:

```
    <citeStructure match="div[@type='textpart']" use="@n" delim=" " unit="poem">
```

What this means is that, although `@use` is perfectly good for deriving or resolving a citation, it won't actually help us build the table of contents text. Worse, it won't help us build the links either, unless we can resolve XPaths in the browser. We can do that, of course, but it would be simpler to link directly to an `@xml:id` if there is one. Once again, `<citeData>` will do what we need. We can use a "dc:identifier" property to get the element's `@xml:id`, enabling us to link to it in the table of contents.

```
<refsDecl>
  <citeStructure match="//front/div[@type='introduction']" use="'Introduction'">
    <citeData property="dc:identifier" use="@xml:id"/>
    <citeData property="#function" use="'toc-entry'"/>
    <citeStructure match="div[@type='section']" use="head" delim=", " unit="section">
      <citeData property="dc:title" use="head"/>
      <citeData property="dc:identifier" use="@xml:id"/>
      <citeData property="#function" use="'toc-entry'"/>
      <citeData property="#function" use="'chunk'"/>
      <citeData property="dc:requires" use="//front/div[@type='bibliography']"/>
    </citeStructure>
  </citeStructure>
  <citeStructure match="//front/div[@type='bibliography']" use="head">
    <citeData property="dc:identifier" use="@xml:id"/>
    <citeData property="#function" use="'toc-entry'"/>
    <citeStructure match="div[@type='bibliography']" use="head" delim=", " unit="section">
      <citeData property="dc:title" use="head"/>
      <citeData property="dc:identifier" use="@xml:id"/>
      <citeData property="#function" use="'toc-entry'"/>
      <citeData property="#function" use="'chunk'"/>
    </citeStructure>
  </citeStructure>
  <citeStructure match="//body/div[@type='edition']" use="head">
    <citeData property="dc:identifier" use="@xml:id"/>
    <citeData property="#function" use="'toc-entry'"/>
    <citeStructure match="div[@type='textpart']" use="@n" delim=" " unit="poem">
      <citeData property="dc:title" use="head"/>
      <citeData property="dc:identifier" use="@xml:id"/>
      <citeData property="#function" use="'toc-entry'"/>
      <citeData property="#function" use="'chunk'"/>
      <citeData property="dc:requires" use="//front/div[@type='bibliography']"/>
      <citeStructure match=".//l[parent::div or parent::lem]" use="@n" delim="." unit="line"/>
    </citeStructure>
  </citeStructure>
</refsDecl>
```

The upshot is that citation structures appear to give us all the tools we need to de even quite sophisticated operations, which means we need not rely on systems managing TEI documents having prior knowledge of their internals. The documents themselves can tell us how to work with them.

---

[1] See https://distributed-text-services.github.io/specifications/.

[2] See the release notes at https://tei-c.org/release/doc/tei-p5-doc/readme-4.2.2.html.

[3] From https://github.com/PerseusDL/canonical-latinLit/blob/61837e978b0d45c8e2086a4b8be62824022800cc/data/phi0448/phi002/phi0448.phi002.perseus-lat2.xml#L53-L73.

[4] See https://www.tei-c.org/release/doc/tei-p5-doc/en/html/HD.html#HD54M. “This method is appropriate when only ‘milestone’ tags are available to provide the required referencing information.”

[5] The example above demonstrates the fragility of the regex replacement system provided by `<cRefPattern>`. Observant readers will already have noted that the order, book, chapter, section is reversed in the first `<refsDecl>`, presumably because the regular expressions provided are overly general. `\w+` (the pattern for "book") will happily match a reference "2.1.3", and `(\w+).(\w+).(\w+)` will match any word, such as “oopsie” or “antidisestablishmentarianism”. It matches any string at least five characters long. Clearly something more precise, like `(\d)\.(\d{1,3})\.(\d{1,2})` was meant, since there are three books, the longest with just over 100 chapters, and some chapters with over 10 sections. Further, the overuse of `<div>`s is probably a reflection of the need for as much homogeneity as possible in a system designed to process lots of documents. The "chapters" in this work are more like paragraphs, and are likely to be printed as such, and the "sections" are roughly sentence-level stretches of text. It might be more idiomatic TEI to treat chapters as `<p>` elements and sections as `<seg>`s.

[6] See https://tei-c.org/release/doc/tei-p5-doc/en/html/SA.html#SACRCS.

[7] https://www.dublincore.org/specifications/dublin-core/dces/

Hugh Cayless

Hugh is a Senior Digital Humanities Developer at Duke University Libraries.

Thibault Clérice

Thibault Clérice est responsable du master « Technologies numériques appliquées à l’histoire » de l’École nationale des chartes (Paris, France).

Jonathan Robie

Instigator in Chief at biblicalhumanities.org.

Balisage Series on Markup Technologies