Jochen Tiepmar and Gerhard Heyer

The Canonical Text Services in Classics and Beyond

Abstract: Starting with the project A Library of a Billion Words (ESF 100146395) and ongoing in the Big Data related project Scalable Data Solutions (BMBF 01IS14014B), the NLP group in Leipzig was tasked to develop a feature complete and generic implementation of the Canonical Text Services (CTS) protocol that is able to handle billions of words. This paper describes how this goal was achieved and why this is a significant step forward for the communities of humanists and computer scientists who work with text data.

1 Introduction

With the ongoing digitization of text data and the general trend for digital publications, the ability to persistently reference text snippets as digital resources across projects becomes increasingly important. For this purpose the Canonical Text Services (CTS) protocol was developed for the Homer Multitext project supported by the Center for Hellenic Studies of Harvard University.[1] CTS incorporates the idea that annotations can naturally be based on an inherent ontology of text passages such as chapters, paragraphs, sentences, words, and letters. It allows researchers to identify precise words and phrases in particular versions of a work without having to rely on particular editions. A Canonical Text Service can be characterized as a complex text retrieval webservice that provides persistent reference (CTS) URNs for hierarchical text elements (e.g. chapter, sentence, down to character) and request functions to retrieve text content and structural meta information for each of the references as well as each span between them. As such it provides citable reference points for every

1 https://www.homermultitext.org (last access 2019.01.31). See Smith (2009).

Note: Part of this work was funded by the German Federal Ministry of Education and Research within the project ScaDS Dresden/Leipzig (BMBF 01IS14014B) and by the European Social Fund in the project The Library of a Billion Words (ESF 100146395).

Jochen Tiepmar, Gerhard Heyer, Universität Leipzig

https://doi.org/10.1515/9783110599572-007

possible text passage in a document, making it a very valuable tool for (digital) humanists.[2]

A graph based and an XML based implementation provided the basic functionalities of the protocol but the more advanced functionalities like sub references and text spans proved to be problematic for these solutions. They were additionally developed around specific data sets and hard to adapt to external resources. Therefore and in order to expand the usefulness of CTS beyond the Classical languages (i.e. Greek and Latin), it seemed reasonable to develop a third implementation based on the documented learned lessons with a specific focus on efficient scalability and generic applicability.

2 The relevance of CTS in computer science

Tiepmar (2018) shows that CTS can be technically seen as a RESTFul webservice[3] that integrates well with existing technical solutions as they are for instance used in CLARIN[4] or more recently in projects like Das Digitale Archiv NRW.[5] Instead of being in competition with used systems, it provides huge potential for technical improvements as described in the following pages.

2.1 Normalized text access across data sources

Even though they are all modern and ongoing projects, examples like Deutsches Textarchiv, Perseus, Eur Lex (EU 2017) and Project Gutenberg show that each requires individual ways to access data.[6] Perseus offers a public GitHub repository and the other three projects specific websites. There is no obvious way to collect a dump of the data, which means that in order to work with the data sets locally, an individual web crawler has to be implemented or the data has to be requested via one of the contact possibilities.

Another problem is that digitized documents are often published in varying formats. Each of the four examples uses a specific markup to structure their

2 For a more detailed explanation about Canonical Text Services, see Smith (2009), Blackwell et al. (2017), Tiepmar et al. (2014) and Tiepmar (2018).
3 (Fielding 2000).
4 (Hinrichs and Krauwer 2014).
5 (Thaller 2013).
6 (Geyken et al. 2011); (Smith et al. 2000); (Hart 2017).

documents. DTA and Perseus offer texts in TEI/XML but the metadata markup is varying. Generally, to access individual text units it is required to know in which way the structure is marked in each document before being able to access it. For instance, to access individual lines it may be required to look for <l> or </lb> and paragraphs may be marked as <p> or . It may even be problematic to find out how or if the document is structured in the first place. This is a problem because it prevents the implementation of tools that can be reused without adaptation effort.

Because of the strict design of CTS, tools can be developed to work in such a generic way that they are able to work with any CTS endpoint. This makes it possible to exchange and access text data without having to learn how a certain data set should be accessed.

2.2 Separate structural meta information

Documents can be divided into a hierarchical system of text parts like for example chapters that consist of sentences or songs that consist of stanzas that consist of verses. This structural meta information is part of the metadata markup possibilities that are provided by TEI/XML or DocBook but, since this information is technically not different from any other meta information, it is hard to use it as input for tools.

Yet it showed that this information can be very useful and tools would benefit from a reliable generic way of accessing it. Since CTS URNs are built from this structural meta information, they also indirectly encode it as it is illustrated in the following example. The URNs have been shortened for better readability:

:1:1.1:1.1.1 O Christmas tree, O Christmas tree !
:1.1.2 How are thy leaves so verdant !
:1.1.5 O Christmas tree, O Christmas tree,
:1.1.6 How are thy leaves so verdant !
:1.2:1.2.1 O Christmas tree, O Christmas tree,
:1.2.2 Much pleasure doth thou bring me !
:1.2.5 O Christmas tree, O Christmas tree,
:1.2.6 Much pleasure doth thou bring me !

This problem could also be solved by agreeing on what is considered as a structural metadata tag, but this solution would still have the potential to create ambiguity as it is illustrated in the following example:

<chapter> This is a chapter that references chapter <chapter>1</chapter> </chapter>.

In this constructed example, a reference to another chapter is marked with the same tag that is used for the text passage. <chapter> is a reasonable (and the only) choice for a tag that describes structural information. But doing so means that its use as meta information in <chapter>1</chapter> would be interpreted as structural information, resulting in an additional sub chapter with the text content 1:

> <chapter> This is a chapter that references chapter <chapter> 1 </chapter> </chapter>.

While it can be discussed, which of the interpretations is "more right" and whether or not this example should be considered as realistic, it is obviously true that the technical interpretation can be ambiguous if meta information and document structure use the same markup.

With CTS URNs, this encoding of the hierarchical information in documents can be accessed separately from the meta information encoded in the metadata markup and can serve as the basis for new generic algorithmic approaches to text mining.

2.3 Granularity

Current text reference systems like for instance the PID handles that are used in CLARIN or the URNs that are used in Das Digitale Archive NRW allow to reference electronic resources.[7] In the context of text data such references mostly correlate to individual text files. CTS URNs additionally enable researchers to reference structural elements of digitized documents like chapters or sentences in a unified way.

This fine granular reference system is for instance one of the advantages that justified the inclusion of the CTS protocol in CLARIN as it is described by Tiepmar et al. (2017) and Grallert et al. (2017), because it allows text research infrastructures to provide persistent identifiers for the structural elements of a text with varying granularities.

2.4 Text streaming

The work described by Smith (2009) indirectly points out another advantage of the usage of CTS:

7 (van Uytvanck 2014); (Thaller 2013).

> "These Canonical Text Services URNs make it possible to reduce the complexity of a reference like First occurrence of the string 'cano' in line 1 of book 1 of Vergil's Aeneid to a short string that can then be used by any application that understands CTS URNs".

This also implies that it is possible to reduce long texts to CTS URNs and request them as they are needed. In this way the memory needed for software that handles texts or text parts can be reduced because the software does not have to memorize the text passages but instead memorizes the relative short CTS URNs and requests text information as it is needed.

Because of the hierarchical properties of CTS URNs, they may also allow specific caching techniques. Generally, books tend to include more text than can be shown on a monitor in a reasonable way. If a text passage is too big to be visualized as a whole, it may be more memory efficient to use a sliding window that spans some of the smaller text parts on a lower depth that correlates to the amount of text that is visible in one moment. This streaming technique can be especially valuable when working with systems that do not have access to vast amounts of access memory like smart devices or small notebooks. Figure 1 illustrates this by showing how sets of ten sentences are processed at one moment instead of the complete text.

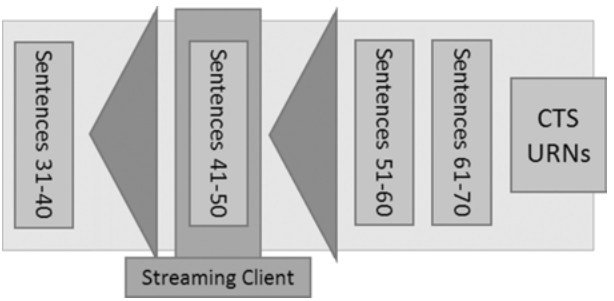

Figure 1: CTS URN based text streaming.

This technique is for example used in CTRaCE to limit the amount of cached content to a reasonable amount instead of handling the full document at any given time.[8]

8 (Reckziegel et al. 2016).

3 Index implementation

A detailed analysis by Tiepmar (2018) concludes that the following requirements must be met for the technical basis of a CTS implementation:

- (At least) UTF-8 support.
- Capability of online – especially multi user – handling.
- Established & Accessible (Usability).
- Independence from a specific input data type.
- Prefix string search or a similarly fitting implicit hierarchy retrieval mechanism.
- Support for sequential order index and range queries.

The implementation of the index itself is most efficiently done using a trie or prefix search tree[9] using prefix search based hierarchy retrieval that can be programmed using standard server SQL techniques.

3.1 Prefix search based hierarchy retrieval

Hierarchical information based on CTS URNs can be requested similar to how prefix based search is done in a trie. For instance, to find out which of the CTS URNs belong to *urn:cts:perseus(. . .):1.*, it is sufficient to traverse the trie according to the given URN. Any (recursive) child node is one of the structural child elements of the URN that was provided as input. Resolving the hierarchical information in CTS URNs can be done by applying the same algorithms that are used for string prefix search because the structural information in them is encoded by the continuation of their string representation. Parent URNs are always prefix sub strings and the set of child URNs is exactly the same as the result set of a string prefix search.

The result of this mapping of seemingly unrelated tasks is that the hierarchy retrieval in this context is technically not a task of data architecture but of information retrieval. String based methods can be used to extract the hierarchy information that is encoded in the CTS URNs. This especially means that the hierarchy information does not have to be modelled explicitly in the data set but is implicitly known to the system as soon as CTS URNs are added. The consequence is that the optimal hierarchy index for a Canonical Text Service is not

9 (Brass 2008).

necessarily a hierarchical data structure but a data structure that is optimised for prefix string search.[10]

An additional benefit of this approach is that it is very flexible. Prefix sub string search works with strings of any length and therefore this approach theoretically supports any possible citation depth. It also does not depend on the URN syntax or any kind of fixed formula and could also extract the hierarchical information from the following example that is far from a valid CTS URN notation[11]:

axl_cts_greekLit(tlg0003.tlg001<perseus_eng1)
axl_cts_greekLit(tlg0003.tlg001<perseus_eng1)buch1
axl_cts_greekLit(tlg0003.tlg001<perseus_eng1)buch1_3
axl_cts_greekLit(tlg0003.tlg001<perseus_eng1)buch1_3the5thverse

This approach is flexible enough that changes in the URN syntax or related future schemas can be supported. This especially means that this method can be applied to similar systems like the CITE[12] protocol – which uses references similar to CTS URNs for discrete objects and images – without significant effort.

3.2 Proposed index implementation

The proposed index implementation is based on MySQL Version 5 (Oracle 2018), or similar systems like MariaDB.[13] UTF-8 is supported, along with a vast number of other character sets. It is an established data storage technique in the context of online services that is often part of the pre-installed software packages for servers.[14] MySQL does not have a required input data format, the data has to be added and requested by the software that uses it. Responses are generally formatted into or from specific formats by the application software.

10 Which is more specific than *tree*, but would also include potential non tree prefix search methods.

11 *axl_cts_greekLit(tlg0003.tlg001<perseus_eng1)buch1* is the parent node of *axl_cts_greekLit (tlg0003.tlg001<perseus_eng1)buch1_3* and so on.

12 http://www.homermultitext.org/hmt-docs/cite/cite-overview.html (last access 2019.01.31).

13 https://mariadb.org (last access 2019.01.31).

14 SQL is included in software like Xampp, hosting services like Strato and Host Europe and requirement for Wordpress – one of the most established Blog/Website backends.

Sequential order indices as the basis for range queries can be implemented using an incrementing integer, that can simultaneously act as the primary key for the data rows. In order for this index to be useful to find left and right neighbour entries, it should be made sure, that the incrementing integer value is free of gaps.[15]

Prefix based string search can be implemented using the *LIKE* command with wildcard symbol % that matches any number of characters. *LIKE BINARY* makes sure that the search is done with case sensitivity. *LIKE BINARY* queries are significantly slower than *LIKE* queries because SQL does a complete scan for *BINARY*. Therefore every *BINARY* query is applied using a syntax similar to *LIKE AND LIKE BINARY*,[16] which means that SQL only does the expensive case sensitive lookup after the search room is limited to the case insensitive matches.

The following example illustrates the used prefix based string search:

urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:3.116
urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4
urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4.1
urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4.2
(. . .)
urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:40
urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:40.1

The LIKE BINARY query for urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4. returns the following result set[17]:

urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4.1
urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4.2

Both results are child URNs of the input CTS URN. Any child URN of the input CTS URN must be part of the result set because all of them start with the input CTS URN. It is important to append the delimiting characters to the request parameter.[18] If the prefix search would be done using *urn:cts:greekLit:tlg0003.*

15 The result from MySQL's *AUTO_INCREMENT* is not necessarily gap free.
16 SELECT urn WHERE urn LIKE "urn:cts:pbc:bible.parallel.eng.kingjames:2.1.%" AND urn LIKE BINARY "urn:cts:pbc:bible.parallel.eng.kingjames:2.1%".
17 SELECT urn WHERE urn LIKE BINARY "urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4.%".
18 The dot "." and the colon ":".

tlg001.perseus_eng1:4 as the input parameter instead of *urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4.*, the result would include the correct CTS URNs

urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4.1
urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4.2

as well as the incorrect CTS URNs

urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:40
urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:40.1

MySQL provides a B-Tree index that can be applied to text data and is used for LIKE comparisons if the input string does not start with the wildcard character. This results in a database table as shown in Table 1.

Table 1: CTS URN database table.

ID	URN	text
22	urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:3.116	The same (. . .)
23	urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4	NULL
24	urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4 .1	The spring (. . .)
25	urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:4 .2	About the (. . .)

Using this schema, every table row corresponds to exactly one structural element of the input document. The column *ID* is indexed as the primary key of the database and serves as the sequential index that is required for the range queries and the neighbour requests. The column *URN* is indexed using MySQL's B-Tree implementation and is used for the prefix based string search that serves as the hierarchical index. The column *text* is not indexed as it is not used for any kind of request.[19] Additional columns can for example be added to store language information or the type of each structural element.

19 The text column has been indexed due to the implementation of the fulltext search described in section 3.2, but this additional index is not required for the CTS index.

The text is only stored on the lowest hierarchical level because the text on higher levels is generated dynamically.

The advantage of this index implementation is that it naturally supports the data specific requirements without requiring a remarkably sophisticated technical setup to work. Since CTS requires server software by definition and some variant or version of (My)SQL is generally part of the package that is included in server software, this approach does not add any significant technical requirement for the average user. SQL databases are also not limited to any specific programming language. While this implementation is based on JAVA,[20] the basic programming logic of the index is handled using SQL queries. This means that it could be reimplemented in any other programming language without the need for a newly developed index technique.

The disadvantage of this approach as it is currently implemented is that the length of CTS URNs is restricted to 255, the maximum length of MySQL's *VARCHAR* data type. Since these references are supposed to be used as citations in human readable documents, this disadvantage should not be problematic.[21] Because CTS URNs are separated by namespaces, it can also be expected that this is not a future problem. Even if the allowed characters are arbitrarily limited to English letters, the potential combinations of delimiting namespace names – and therefore the set of supported text corpora – already include 26^n elements with n being the length of the namespace string.[22] If the number of possible namespaces is eventually too low, it could be multiplied by the use of a different URN namespace like *urn:cts2:*.

It is important to emphasize that this work does not propose that a CTS implementation must be done using SQL. SQL merely serves as the tool that is used to implement a B-Tree based trie data structure[23] and is especially fitting because of how established it is as part of server software packages. The hierarchical information is not necessarily stored in the B-Tree but in the way it is processed.

A detailed technical comparison by Tiepmar (2018) shows that this approach is a significant improvement over the graph and XML based CTS solutions.

20 JAVA was chosen because of its widespread support and its uncomplicated use as web applications (Servlets).

21 The CTS URN *urn:cts:pbc:bible.parallel.eng.kingjames:2.1.2* is 46 characters long.

22 456976 for $n = 4$.

23 A balanced tree that is processed in such a way that input and output is equal to that of a trie.

4 Unique features

4.1 Additional request functions

Since most of the additional features are not covered by CTS, it was necessary to implement the possibility for additional requests that do not interfere with future iterations of the CTS protocol. This is assured by using a different URL path than any of the CTS requests. Any of the official requests starts with the URL path http://cts.informatik.uni-leipzig.de/perseus/cts/.

The path for any of the additional requests starts with http://cts.informatik.uni-leipzig.de/perseus/plain/.

The added requests use a different optional URL branch than the official requests and therefore can not contradict the current and future CTS specifications.

The following (incomplete)[24] list of requests provide more convenient or efficient request possibilities compared to what would be necessary if only CTS requests are used:

- *editions*, *authors*, *titles* and *titlesandurns* provide a list similar to the content of the text inventory from the CTS request *GetCapabilities*. For text collections that contain several hundreds of thousands of documents, the text inventory file is a relatively large XML document.[25] This can create performance problems when the inventory is processed, especially because CTS does not provide any paging mechanism. The added features do allow paging and do not require XML parsing. The result is that data can be processed in chunks and the full data set can be requested faster and with less memory impact.[26]
- *metaforkey* provides a URN specific request possibility for any kind of document level meta information that is part of the CTS data set. Without this function, this information is only served as part of the text inventory file, which means that the full text inventory has to be

24 Requests like *getPassage* or *childList* are not considered because they work exactly like their official CTS counterparts.

25 At least the title, author, publication date and URN of each document.

26 Requesting the full *GetCapabilities* for the 62281 documents of the TextGrid data set using Firefox 52.0.2 (32-Bit) on a Intel(R) Core(TM) i5-4200U CPU @ 1.60GHz with 4 GB RAM (Windows 8 64 bit) resulted in an application crash after 1 minute and 50 seconds. Requesting *plain/editions* resulted in the full URN list after 21 seconds. *GetCapabilities* is the only specified source for document level CTS URNs.

processed any time a specific part of meta information for a document is required.
- Requests like *urncount* and *doccount* are added to provide useful statistics about the size of the text collection.
- Requests like *urnstypes*, and *urnstypestextlength* provide fine grained technical views on the data that might be more useful in a development environment. For instance, *urnstypestextlength* provides the structural markup of the document and the length of every text part as illustrated in the following example:

  urn:cts:greekLit:tlg0003.tlg001.perseus_eng1: edition -
  urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:1 book -
  urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:1.1 chapter 1335
  urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:1.2 chapter 2608
  urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:1.3 chapter 2304

This enables tool implementers for example to know beforehand how much more text parts can fit on a screen.

4.2 Configuration parameter

Since most the post processing features are additional – and therefore optional – functions, a configuration parameter is required to enable users to specify if an option should be activated or deactivated. The specifications specifically highlight

http://myhost/mycts?configuration=default request=GetCapabilities

as a valid URL and it can be assumed that this implies that additional parameters may be added to a request. If this interpretation is not correct, then the parameter has to be considered as an optional extension of the protocol.

Table 2 provides an overview about the parameters that are available.

Each of the parameters can be configured with the boolean values *true* or *false*. Multiple parameters can be combined using the underscore character as it is done in the following example:

configuration=seperatecontext=true_deletexml=false_usectsnamespace=false

An exemplary use of the parameter is described in section 4.3.

Table 2: Configuration parameters.

Parameter	Effect
divs	Document structuring using numbered <div*>s
epidoc	Document structuring using Epidoc (Bodard 2010)
newlines	Document structuring using newlines
maxlevelexception	Return *error* for unsupported citation level requests
escapePassage	XML-escape the text content
seperatecontext	Add (the optional) text context to a passage or separate it
smallinventory	Text inventory reduced to a URN list
xmlformatting	Pretty print XML
deletexml	XML markup deleted for increased readability
usectsnamespace	Use the CTS namespace for CTS specific XML tags

4.3 Text passage post processing

Since the text passage that is requested is generated dynamically, it is technically possible to influence the generation process in various ways. Therefore it is possible to implement different views on the same text data sample. The result of such a post processing mechanism can be considered as an additional automatically edited variant that is available without any need for individually edited documents. While the examples in this section only include basic post processing steps, it is possible to extend this feature as part of future work to provide automatically generated transcriptions into other lexical alphabets, complementary information like named entities or citation links and many other useful mechanisms.

The different views on the text passages are requested using the configuration parameter described in section 4.2.

Figure 2 shows an example text passage from the Perseus data set as it is requested using the configuration parameter

configuration=divs=true_deletexml=false_escapepassage=false

This combination of parameters structures the text passage using numbered <div*>s and includes the text content without escaping the XML characters or

```
- <passage>
  - <div1 n="1" type="card">
      <stage TEIform="stage">Enter the Chorus of Trojan guards.</stage>
    - <sp TEIform="sp">
        <speaker TEIform="speaker">Chorus</speaker>
      - <p TEIform="p">
          Go to Hector's couch. Which of you squires that tend the prince, or
          <milestone ed="p" n="5" unit="line" TEIform="milestone"/>
          from the warriors who were set to guard the assembled army during
          <stage TEIform="stage">Calls to Hector in the tent.</stage>
          Lift up your head! Prop your arm beneath it! Unseal that fierce eye
          <milestone ed="p" n="10" unit="line" TEIform="milestone"/>
          Hector! It is time to hearken.
        </p>
      </sp>
    - <sp TEIform="sp">
        <speaker TEIform="speaker">Hector</speaker>
      - <p TEIform="p">
          Who is this? Is it a friend who calls? Who are you? Your password?
        </p>
      </sp>
    - <sp TEIform="sp">
        <speaker TEIform="speaker">Chorus</speaker>
```

Figure 2: Configuration parameter Example 1.

deleting the XML content. If this configuration is used, it is possible to request invalid XML which would result in client and server side parsing errors. To avoid this problem, requests that include sub passage notation or spans of CTS URNs will ignore the *escapepassage* parameter and set it to *true*. This also happens if text content that could not be parsed as XML is part of the text of the source document. If static CTS URNs based in valid XML source files are requested, this problem cannot happen because every static text part is based on a valid XML node in the source file.

Figures 3 and 4 show the same text passage using different configurations and especially illustrate the difference in the handling of the structural markup and the meta information markup. Figure 3 uses the configuration parameter

configuration=divs=true_deletexml=false_escapepassage=true

This parameter configuration also uses numbered <div*>s to communicate the document structure but makes sure that any XML reserved character in the text content is escaped. This view illustrates the difference between the structural and the meta information markup especially well.

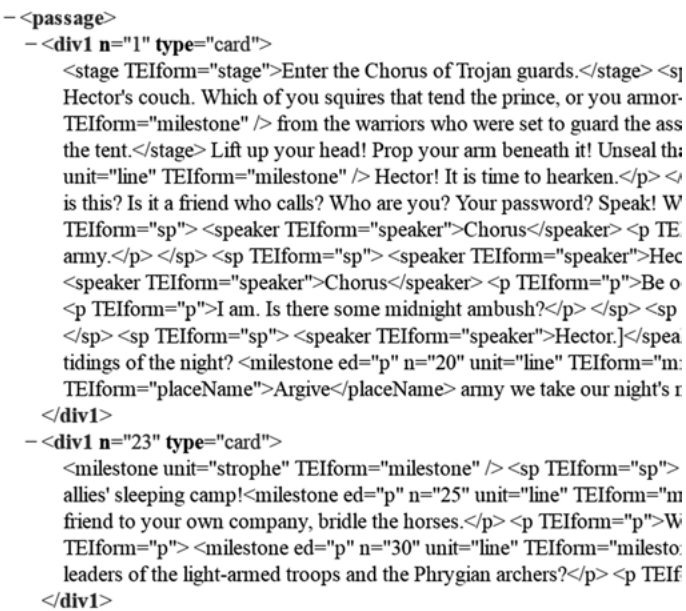

Figure 3: Configuration parameter Example 2.

Figure 4 uses the configuration parameter

configuration=epidoc=true_deletexml=true_escapepassage=false

This combination of parameters uses a notation similar to the Epidoc format (Bodard 2010) to provide the document structure. XML characters in the text content are not escaped. Instead anything that resembles an XML notation is deleted.[27] Depending on the structural markup quality, this configuration can already provide a relatively reader friendly way to serve the data. Yet, since there is no technical way to differentiate XML markup from text snippets like *1<3. 3>2.*, this view may delete text content that it should not. Deleting the XML from the text requires *escapepassage* to be *false*. This implies that *deletexml* does not work in cases that are problematic for *escapepassage*.

27 Anything that matches <*>.

```
- <passage>
  - <tei:TEI>
    - <tei:text>
      - <tei:body>
        - <tei:div n="1" type="card">
            Enter the Chorus of Trojan guards. Chorus Go to Hector's couch.
            receive fresh tidings from the warriors who were set to guard the
            head! Prop your arm beneath it! Unseal that fierce eye from its re
            Is it a friend who calls? Who are you? Your password? Speak! W
            army. Hector Why this tumultuous haste? Chorus Be of good co
            your post and rouse the army, unless you have some tidings of th
            armor?
          </tei:div>
        - <tei:div n="23" type="card">
            Chorus To arms! Hector, seek your allies' sleeping camp! Stir the
            Who will go to the son of Panthus? Who to Europa's son, captain
            the light-armed troops and the Phrygian archers? String your hor
          </tei:div>
        - <tei:div n="34" type="card">
            Hector Your tidings inspire now fear, now confidence; nothing is
            [Leaving your watch you rouse the army.] What does your noisy
            statement have you made.
          </tei:div>
```

Figure 4: Configuration parameter Example 3.

4.4 Licensing

Content licenses often require that a specific license text and the source of a document are disclaimed if parts of the content are re-used or published.[28] This is not considered in the specification of the CTS protocol, even though serving text passages has to be considered as a re-use or publication, especially when it is possible to request the text passage that is the complete document. Consequentially, many publicly available text corpora are excluded from being served by a Canonical Text Service.[29]

This implementation of CTS provides the possibility to serve a license- and a source text on document- and corpus level. The license text on corpus level is manually configured by the administrator of the CTS instance. The text on document level is extracted from the input files and therefore based on the information that was added by the document editors. The configured address of the CTS instance is added to the source text by the system as illustrated in the following example:

28 For instance CC-BY (Creative Commons 2018).

29 For example Das Deutsche Textarchiv (Geyken et al. 2011).

<reply>
<urn> urn:cts:dta:moritz.reiser02.de.norm:654 </urn>
<passage> Der Rektor hatte darin sehr Recht – denn der Vorfall wurde bald bekannt, und es hie"s nun: wie! </passage>
<license> Distributed under the Creative Commons Attribution-NonCommercial 3.0 Unported License. </license>
<source> http://www.deutschestextarchiv.de/moritz_reiser02_1786 (...) retrieved via Canonical Text Service www.urncts.de/dta/cts with CTS URN urn:cts:dta:moritz.reiser02.de.norm:654 </source>
</reply>

The server address has to be configured manually because certain network configurations include proxy mechanisms that make it difficult to detect the external server address automatically from within a network application.

4.5 CTS cloning

One of the benefits of a system like a Canonical Text Service is its potential use as an application independent archival tool that supports more spontaneous project specific archives and seamlessly connects them to organised central archival projects. In order to achieve this, it is required that the data can be moved from one physical address to another without reference changes. Since CTS URNs are application independent per definition,[30] the problem of reference changes is solved.

Using the possibility to request the structural information of a document along with the text content of each structural element[31] and the meta information from the text inventory, any document that is served by this implementation of CTS can be reconstructed in another CTS instance. This process is called CTS Cloning.

It is possible to implement such a system without the use of the combined structural and textual information and only by the means that the specifications provide. Yet this would require a relatively large number of requests because the text content of each structural element would have to be requested

30 And therefore service independent.
31 See the div-View described in section 4.3.

individually.[32] For this reason, CTS Cloning in its current form only works with CTS instances that are based on this implementation.

It is possible to filter the documents that are supposed to be cloned based on the information that is encoded in the document level CTS URNs. Text inventory based meta information filters are not implemented because they can be specified as a list of document level CTS URNs and therefore the ability to filter the documents by a specific piece of optional meta information is not required.

Document clones can be added to an existing CTS instance. Duplicates can only happen if one of the source data sets did not respect the already reserved CTS namespaces.[33] Duplicate CTS URNs are ignored.

Since document sets can be filtered and combined, it is easily possible to create subsets of text corpora that share a research question specific set of properties that was not considered as part of the originally created text corpora. For instance, it is possible to combine the texts from a specific time frame based on the TextGrid and Deutsches Textarchiv corpora or to compile a data set based on a specific set of topics, languages or genres. This compilation of documents can be used to investigate research question specific effects and provide the compiled data set along with the results.

The possibility to clone the documents enables users to manually change the text content of an established CTS URN. This corrupted data set can be used as a text reference if the corresponding CTS request is sent to the corrupted clone instead of the original CTS instance. For instance, the CTS request

> http://cts.informatik.uni-leipzig.de/perseus/cts/?request=GetPassage&urn=urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:1.1

could be redirected to another CTS request

> http://mysserver.de/pbc/cts/?request=GetPassage&urn=urn:cts:greekLit:tlg0003.tlg001.perseus_eng1:1.1

that might resolve the CTS URN based on manually changed text content. The response would be equal[34] except for the manually changed bits of information. Since URLs are often hidden behind a label to improve readability, it is possible

32 21'911'559 requests to recreate the structural information of the CTS instance containing the texts from the Deutsches Text Archiv. Using the combined information reduces this number to 8'190, the number of document level CTS URNs.

33 Like *urn:cts:dta:* or *urn:cts:perseus:*.

34 Potentially including the manually configured server address in the source text.

to hide corrupted CTS requests. This issue can be partly solved by making sure that CTS URNs are always requested from trusted sources that can be managed by a central service like the Namespace Resolver.[35] It is also advised to check the trustworthiness of any URL before clicking on it as it should be general practise for users of internet resources.

5 Conclusions

This paper describes the technical basis for the first feature-complete implementation of the Canonical Text Services protocol. During the course of this work it was possible to extend the protocol with useful features such as a licensing mechanism and more efficient request features that circumvent the disadvantages of the use of XML. As shown at the beginning of this paper, CTS is a helpful tool for software developers that can be the basis for numerous innovations, especially since it is now agreed on both by researchers in computer science and the humanities. Future work may include improvements in individual features, integration in existing infrastructures and application in similar work as for instance the aforementioned CITE protocol.

Bibliography

Blackwell, C.; Roughan, C.; Smith, D. (2017): "Citation and Alignment: Scholarship Outside and Inside the Codex". Manuscript Studies 1: 1. http://repository.upenn.edu/mss_sims/vol1/iss1/2 (last access 2019.01.31).

Bodard, G. (2010): "Epigraphic Documents in XML for Publication and Interchange". In: F. Feraudi-Gruénais (ed.): Latin on Stone: Epigraphic research and electronic archives. Lanham: Lexington Books, 1–17.

Brass, P. (2008): Advanced Data Structures. Cambridge: Cambridge University Press.

Creative Commons (2018): Attribution 4.0 International (CC BY 4.0). https://creativecommons.org (last access 2019.01.31).

EU (14.05.2017). EUR-Lex. http://eur-lex.europa.eu/homepage.html (last access 2019.01.31).

Fielding, T. (2000): Architectural Styles and the Design of Network-based Software Architectures. PhD Thesis. Irvine: University of California.

Geyken, A.; Haaf, S.; Jurish, B.; Schulz, M.; Steinmann, J.; Thomas, C.; Wiegand, F. (2011): Das Deutsche Textarchiv: Vom historischen Korpus zum aktiven Archiv. Cologne: Digitale Wissenschaft Stand und Entwicklung digital vernetzter Forschung in Deutschland.

35 (Tiepmar 2018).

Grallert, T.; Tiepmar, J.; Eckart, T.; Goldhahn, D.; Kuras, C. (2017): "Digital Muqtabas CTS Integration in CLARIN". CLARIN Annual Conference 2017. CLARIN ERIC.

Hart, M. (2017): Project Gutenberg. http://www.gutenberg.org (last access 2019.01.31).

Hinrichs, E.; Krauwer, S. (2014): "The CLARIN Research Infrastructure: Resources and Tools for E-Humanities Scholars". In: Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014. European Language Resources Association, 1525–1531.

Oracle (2018): MySQL Reference Manual 5.7. https://dev.mysql.com/doc/ (last access 2019.01.31).

Reckziegel, M.; Jaenicke, S.; Scheuermann, G. (2016): "CTRaCE: Canonical Text Reader and Citation Exporter". In: Digital Humanities 2016. Kraków: Jagiellonian University. http://dh2016.adho.org/static/data/485.html (last access 2019.01.31).

Smith, D. (2009): "Citation in Classical Studies". Digital Humanities Quarterly 3: 1. http://www.digitalhumanities.org/dhq/vol/3/1/000028/000028.html (last access 2019.01.31).

Smith, D.; Rydberg-Cox, J.; Crane, G. (2000): "The Perseus Project: A Digital Library for the Humanities". Literary and Linguistic Computing 15:1, 15–25. https://doi.org/10.1093/llc/15.1.15.

Thaller, M. (2013): Das Digitale Archiv NRW in der Praxis. Eine Softwarelösung zur digitalen Langzeitarchivierung. Kölner Beiträge zu einer geisteswissenschaftlichen Fachinformatik 5. Hamburg: Verlag Dr. Kovač.

Tiepmar, J. (2018): Implementation and Evaluation of the Canonical Text Service Protocol as Part of a Research Infrastructure in the Digital Humanities. PhD Thesis. Leipzig: Universität Leipzig.

Tiepmar, J.; Eckart, T.; Goldhahn, D.; Kuras, C. (2017): "Integrating Canonical Text Services into CLARIN's Search Infrastructure". Linguistic and Literature Studies 5:2, 99–104. http://www.hrpub.org/journals/article_info.php?aid=5844 (last access 2019.01.31).

Tiepmar, J.; Teichmann, C.; Heyer, G.; Berti, M.; Crane, G. (2014): "A New Implementation for Canonical Text Services". In: Proceedings of the 8th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH). EACL, 1–8. https://aclweb.org/anthology/W/W14/W14-0601.pdf (last access 2019.01.31).

van Uytvanck, D. (2014): PID policy summary. CLARIN. Utrecht: SCCTC.