Published in: Bański, Piotr; Biber, Hanno; Breiteneder, Evelyn; Kupietz, Marc; Lüngen, Harald; Witt, Andreas (eds.) (2015): Proceedings of the 3rd Workshop on Challenges in the Management of Large Corpora (CMLC-3). Mannheim: Institut für Deutsche Sprache, pp. 35-43.

Release of the MySQL based implementation of the CTS protocol

Jochen Tiepmar
ScaDS
Leipzig University
Ritterstrasse 9-13, 2.OG
04109 Leipzig
jtiepmar@informatik.uni-leipzig.de

Abstract

In a project called "A Library of a Billion Words" we needed an implementation of the CTS protocol that is capable of handling a text collection containing at least 1 billion words. Because the existing solutions did not work for this scale or were still in development I started an implementation of the CTS protocol using methods that MySQL provides. Last year we published a paper that introduced a prototype with the core functionalities without being compliant with the specifications of CTS (Tiepmar et al., 2013). The purpose of this paper is to describe and evaluate the MySQL based implementation now that it is fulfilling the specifications version 5.0 rc.1 and mark it as finished and ready to use. Further information, online instances of CTS for all described datasets and binaries can be accessed via the projects website[1].

1 Introduction

CTS is a protocol developed in the Homer Multitext Project[2] and, according to (Blackwell and Smith, 2014), "defines interaction between a client and server providing identification of texts and retrieval of canonically cited passages of texts" by using CTS URNs, that "are intended to serve as persistent, location-independent, resource identifiers".

These URNs are built in a way that resembles the hierarchy in- and outside the document.
The URN *urn:cts:demo:goehte.faust.de:1.2-1.4* refers to the text passage spanning from act 1 scene 2 to act 1 scene 4 of the document Goethe's Faust. The first part *urn:cts:* marks it as an URN of the CTS protocol. The second part *demo:* refers to the namespace that the text belongs to. *goehte.faust.de:* refers to the edition (document) and *1.2-1.4* specifies the text passage inside the document. With the addition of the @-notation for subpassages, like in *1.2@hu-1.4@d*, you can specify any text passage in any translation or edition.

The citation depth and structure can differ between documents - while one document can be structured on 4 levels, like book, chapter, section and sentence, it is also valid to structure another document (or even another edition of the same document) in a different way. This means that – for example – while the passage 2.1 in a bible can refer to part 1 of book 2, in Shakespeare's Sonnets, 2.1 refers to verse 1 of sonnet 2. By reducing the type of each text unit to a label, the protocol makes it possible to use any possible text. The worst case scenario would be that no information about the structure of a document is available, in which case it is still possible to use lines as text units.

Even if it might not be intended to be used as such by the authors of the specifications, CTS can serve as a way to standardize texts and therefore work as a text catalogue or -repository. Furthermore, any tool that uses the methods that CTS provides, can work with any data that is or will be added, basically making CTS a framework and standard for public access to text.

Smith (2007) points out another advantage of the usage of CTS: "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 flat string that can then be used by any application that understands CTS URNs". This also means that you can reduce long texts to URNs and then request them as they are needed and this way reduce the memory needed for software that handles texts or text parts.

Using it as a text repository requires a very fast and efficient implementation of the protocol. The

[1] www.urncts.de
[2] http://www.homermultitext.org/

prototype already showed potential for this goal by building maximal passages with response times averaging at 78 MS with a text collection that contains 100‘000 documents with 1‘281‘272‘600 tokens (Tiepmar et al., 2013). As I will show in chapter 7, the implementation still performs fast as it is finished.

While working on this project, 3 major text collections were published as instances of CTS. They are described in chapter 6.

2 Using Canonical Text Services

This chapter is intended to give a rough overview about the specifications defined in (Blackwell and Smith, 2014) and explain the workflow with CTS. Data from CTS is collected via HTTP requests. Each request has to include a GET parameter *request* which specifies, what function of CTS is requested. Attributes are added as GET parameters to the HTTP request. The following functions are available in CTS 5.0 rc.1.

2.1 GetCapabilities

GetCapabilities returns the text inventory of the CTS with all the URNs of works or editions as well as meta information for each entry. The extend or content of the meta information is not specified in CTS.

2.2 GetValidReff(urn,level)

GetValidReff returns all the URNs that belong to the given *urn*. *level* is a required parameter specifying the depth of the citation hierarchy.

2.3 GetLabel(urn)

The request *GetLabel* returns an informal description of the *urn*.

2.4 GetFirstUrn(urn)

GetFirstUrn returns the first URN in document order belonging to the given *urn*.

2.5 GetPrevNextUrn(urn)

GetPrevNextUrn returns the previous and next URN in document order from the given *urn*.

2.6 GetPassage(urn,[context])

GetPassage returns the text passage that belongs to this *urn*. *context* is an optional parameter specifying, how many text units should be added to the passage as contextual information.

2.7 GetPassagePlus(urn,[context])

GetPassagePlus returns the combined information from 2.2 to 2.6

2.8 The Response

The response for each request is a XML-document describing the request and the response from the CTS. For example the response for a GetPassage request is structured according to the following XML-document:

```
<GetPassage>
<request>
<requestName>
GetPassage
</requestName>
<requestUrn>
urn:cts:latinLit:phi1014.phi001.lat1:1
</requestUrn>
</request>
<reply>
<urn>
urn:cts:latinLit:phi1014.phi001.lat1:1
 </urn>
<passage>
(...)
</passage>
</reply>
</GetPassage>
```

It may seem odd that the URN is listed two times. If you do not specify the exact edition it can happen that both URNs differ. Requesting the text passage with urn:cts:latinLit:phi1014.phi001:1 may result in the text passage for urn:cts:latinLit:phi1014.phi001.*lat1*:1[3].

There are contradictory information about whether or not the XML elements must reference CTS as a namespace, like <cts:urn> instead of <urn>[4]. All XML elements in the replies of this implementation are unique and there is no need to differentiate them with namespaces. That's why I chose to not include them. This can be changed as soon as the specifications make it clear, which format should be used.

[3] According to the specifications, an implementation of CTS is free to choose any suitable edition if the edition is not fully specified in the URN.

[4] Compare for example https://github.com/cite-architecture/ctsvalidator/blob/master/src/main/webapp/testsuites/4-09.xml and https://github.com/cite-architecture/cts_spec/blob/master/reply_schemas/prevnext.rng

3 Validation

The specifications refer to a validator that checks whether or not an instance of CTS is compliant with the specifications. Unfortunately, some of the results that the validator expects contradict the specifications making it impossible to validate this implementation[5].

4 Data Structure

This chapter will give an abstract overview about the data structure used in this implementation. A more technical description can be found in (Tiepmar et al., 2013).

To implement an efficient CTS it was crucial that the underlying data structure is as efficient as possible. The best case would be a data structure that resembles the hierarchical structure that is encoded in CTS URNs and this way minimizes the overhead that is needed to describe the structural information. By storing this information in a tree you get a structure that can be modelled similar to the tree in Figure 1.

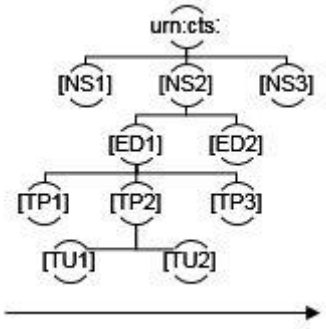

Figure 1, Visualization of the tree-like data structure
NS=Namespace (e.g. greekLit)
ED=Edition (e.g. Goethe's Faust)
TP=Text part (e.g. Chapter)
TU=Text unit (e.g. Sentence)

[TUx] contains the text content for each text unit. The nodes on [TU] level must be ordered as they appear in the document. This is done by using an incremental id indicated by the arrow.

To make sure that you cannot concatenate multiple editions, the CTS will always at least traverse down to edition level and return the first node on that level. Once the node for an URN is found, any related information can be returned. Parent child nodes can be calculated by deleting parts of the URN. The passage can be constructed by concatenating the text units that belong to the node. The child nodes resemble the URNs that belong to the given URN and the first and last child node correspond to the first and last child URN.

When searching for the URN urn:cts:[NS2]:[ED1]:[TP2] the implementation traverses through the tree to the node [TP2]. By this point it knows that this is a valid URN and can return any information associated with this node. If no suitable node is found, then the CTS knows that the URN is not valid. There may be a node [TP2] belonging to [ED2], but as soon as the CTS passed [ED1] this node is no longer in the potential result set.

Treelike data structures provide the benefit of logarithmic search times and (if implemented correctly) prefix- and suffix optimisation, which is beneficial for CTS because the URNs contain a lot of redundant prefixes.

MySQL uses B-Trees for string indices and therefore I considered it a perfect fit for CTS URNs. Another – maybe less technical and more intuitive – way of visualizing it, is that this implementation is using techniques that are generally used for automated completion of strings to build the hierarchy of CTS URNs.

5 Unique Features

There are four unique features to discuss: the possibility to post process the passage, the configuration parameter, the generated text inventory and possibility of multiple import methods. The following chapters will explain these features in detail, give examples of use cases and explain how they fit into the specifications.

5.1 Passage Post Processing

According to (Blackwell and Smith, 2014), the passage "may (…) be further structured or formatted in whatever manner was selected by the editor of the particular edition or translation". This means, that CTS does not restrict the content of the passage in any way as long as "The CTS implementation (…ensures…) that including the contents of the requested in the cts:passage element results in well-formed XML" (Blackwell and Smith, 2014)[6]. As long as it does not break the structure of the reply, the passage may be plain text or – for example – text that either contains XML tags as text or text with XML tags as meta information describing a part of the text.

The following examples help to illustrate the difference.

[5] See issue 26, 27, 28, 29 at https://github.com/cite-architecture/ctsvalidator

[6] The cts:passage element is the XML element in the CTS reply that contains the text passage specified the the URN

a) *The tag <speaker> refers to a speaker and must be closed by </speaker>*
b) *<speaker>Hamlet </speaker>To be, or not to be(...)*

While a) should clearly be seen as plain text describing the tag <speaker>, it is reasonable for an editor to prefer the structured output in example b).

Changing a) to

A) *The tag <speaker> refers to a speaker.*

it becomes obvious that this probably breaks the structure of the CTS reply.

One solution here would be to make sure that every document only contains valid XML. This means that you would either restrict your text to valid XML or have to make sure that anything that would potentially break the XML structure, must be escaped. This results in a lot of work for the editors since they cannot simply escape the whole text but have to differentiate structural tags used by the CTS (like <chapter>) from meta tags that are part of the text (like <speaker>).

The solution that I propose is to make it possible to adapt the content of the passage by the CTS to the needs of the individual text collection or even to the needs of the individual viewer or editor. As long as the post processing method, that is used to modify the passage, is not changed, the CTS still guarantees a persistent citation. One URN will always result in the same text passage, but the data is presented differently. The CTS does not change the textual content, but its representation (or the view on the data) changes.

On the side of the server, this is nothing different than the possibility to serve the text in "whatever manner was selected by the editor" (Blackwell and Smith, 2014). In general, this is the same as creating annotated editions of one document, which is already a common method in today's Digital Humanities as – for example – described in (Almas, 2013). Doing this on CTS level is just automating the process.

On the opposite side, the client can benefit from this by having options. Imagine someone who wants to develop a universal reader for documents in EpiDoc format. It would be very useful to be able to connect to a CTS and have the possibility to request any text in this format without the need to rebuild all the documents and add additional EpiDoc editions. Another reader wants to look up some text but the edition is heavily annotated, making it hard to read. A view without all the XML tags would probably be something nice.

To enable the client to control the format of the passage, it is required to give the possibility to specify a configuration that should be used. This can be achieved with the configuration parameter that I will discuss in the next chapter.

5.2 Configuration Parameter

The configuration parameter was added to this implementation to give any client the possibility to adapt the output of the CTS in different ways. Its use is not described in the specifications but a side note makes it clear, that it does also not violate them. One valid example URL is http://myhost/mycts?configuration=default&request=GetCapabilities[7]. Because this url is valid, it is allowed to add additional parameters to the requests. Therefore it does not contradict the specifications to use it to give the client the ability to configure the CTS as long as the results are still valid against the specifications. In especially the CTS must still make sure, that the reply results in valid XML and all of the required information is included.

It is possible to combine multiple parameters by combining them with "_". For example, the configuration *?configuration=div=true_stats=true* combines the parameters *div* and *stats*.

The following parameters are currently supported. The default values for each parameter can be defined for every CTS instance. The configuration that the client provides will overwrite this default configuration.

Div / Epidoc

The parameters *div* and *epidoc* are useful if you want to see the structure of the text passage – for example to render it nicely. *div* uses a notation with numbered elements and includes the type of the text units as a @type value.

```
<passage>
<div1 n="5" type="book">
<div2 n="1" type="line">
(TEXT)
</div2>
</div1>
</passage>
```

epidoc uses EpiDoc notation, a variation of TEI/XML.

```
<passage>
<tei:TEI>
```

[7] http://folio.furman.edu/projects/citedocs/cts/#client-server-communication

<tei:text>
<tei:body>
<tei:div n="1" type="song">
<tei:div n="1" type="stanza">
<l n="1">(TEXT)</l>
<l n="2">(TEXT)</l>
</tei:div></tei:div>
</tei:body>
</tei:text>
</tei:TEI>
</passage>

epidoc is ignored if *div* is set to true.

Stats

stats does not yet serve a useful purpose but illustrates this implementations flexibility nicely by adding some simple statistics as @-values in the numbered divs. This setting is ignored if *div* is set to false.

<div3 n="1" type="line" letters="24" tokens="4" avg_tokensize="6">
(TEXT)
</div3>

Escapepassage

escapepassage specifies whether or not the XML content of the passage should be escaped. This is always true if URNs with subpassage notation are requested to ensure the validity of the reply.

Seperatecontext

If *seperatecontext* is set to true, then the context that is specified for GetPassage or GetPassagePlus is returned in separate XML elements with the name context_prev and context_next. Else the context is added to the passage and returned inside the passage element.

Formatxml

formatxml configures whether or not the reply should be formatted. Formatted XML is easier to read but if you want to process it automatically, formatting may not be needed and influence the performance of the CTS negatively without having any benefit.

Smallinventory

smallinventory reduces the text inventory to a list of <edition> elements with their URNs. I noticed, that dealing with lots of documents can result in large text inventories that are hard to parse if all the meta information is included. This meta information may be unnecessary if you only need a list of the documents URNs.

Maxlevelexception

If you set *maxlevelexception* to true and then specify a level for GetValidReff that is higher than the levels that the document 'has left', it will return CTS error 4. Else it will return the URNs up to that level. For example if your document has two levels: chapter and sentence, and you request GetValidReff with level=100, then the CTS will return error 4 if this is set to true. It will return all the URNs that belong to the given URN if this is set to false.

The validator requires the CTS to return error 4 if you request a level higher than the document provides[8]. However since there is no way of knowing, how a document is structured and GetValidReff is the function that gives you this information, this would force a user to try out levels until they receive an error, which gets more complicated considering that the document structure is not fixed for the complete document. While in a document book 1 may have 3 levels – chapter, passage, sentence – book 2 of the same document may be structured in 2 levels – stanza, line. This means that you can never know, if you can request another level until you received an error. You can add this information as meta information in GetCapabilities but it is not required by CTS to do so and this solution would still make it problematic to work with documents containing different citation levels.

In my opinion it is more reasonable to ignore this error and make it optional for validation purposes.

This also fits with the specifications noting that "The GetValidReff request identifies all valid values for one on-line version of a requested work, up to a specified level of the citation hierarchy"(Blackwell and Smith, 2014)[9].

5.3 Dynamically Generated Text Inventory

GetCapabilities returns a text inventory containing all URNs that belong to works or editions. This text inventory is manually edited and serves as an overview about what texts are part of the CTS and as a guide for the CTS to know which XML tags of a document are part of the citation.

[8] See https://github.com/cite-architecture/ctsvalidator/blob/master/src/main/webapp/testsuites/3-19.xml

[9] http://folio.furman.edu/projects/citedocs/cts/#cts-request-parameters

Working with a big number of documents, it might be problematic to require someone to read all the documents, create citation mappings, collect the meta information for each document and store it in the inventory file.

While you still have to configure the citation mapping in this implementation, you do not need to do this for every document (you still can if you want). It can be configured in one line for all documents while setting up the CTS. This means that the text inventory is not required to import data, reducing its purpose to the output of GetCapablities. According to (Blackwell and Smith, 2014), the response of GetCapabilities is "a reply that defines a corpus of texts known to the server and, for texts that are available online, identifies their citation schemes". This information can be gathered in an automated process once the data is made available to the CTS.

This way a basic default text inventory is generated which contains all the referenceable editions without the need for manual editing. At the moment of writing, the label and author of an edition and the information, whether or not the edition can be parsed as valid XML, is added as meta information. This result is generated with every new request.

The following example shows the content that is currently included in the text inventory.

```
<TextInventory>
<textgroup urn="urn:cts:greekLit:tlg0003">
<groupname>tlg0003</groupname>
<edition urn="urn:cts:greekLit:tlg0003.
tlg001.eng1:">
<title>
History of the Peloponnesian War
</title>
<author>Thucydides</author>
<contentType>xml</contentType>
</edition>
</textgroup>
</TextInventory>
```

The citation mapping – as it is used to specify, which XML elements are used for citation in the CTS implementation based on a XML database – is not part of the generated inventory because from my understanding it is only useful for the data import. My argument is that once you reference texts with URNs, the citation mapping has only descriptive use and it is better located in the specific text passage or in the reply of the CTS request GetLabel. If you refer to a passage with a URN like urn:cts:demo:a:1.2, it is not relevant, whether the passage – 1.2 – refers to a sentence or verse or line. Adding it to the text inventory can however increase the complexity of the XML document making it harder to process the file. Especially consider that – in theory – every text unit that is referenced by an URN can have its own citation mapping. Mapping one unit to a sentence does not mean that every text unit is a sentence. In the worst case scenario, if citation mappings are included, the text inventory would have to contain one entry for any URN on level of the text units in the complete text collection.

By adding a file named inventory.xml, administrators can instead use one that is manually edited. It is a very reasonable workflow to save the generated inventory as inventory.xml and edit it further to manually add information.

5.4 Multiple Import Methods

The implementation is divided into two parts: one part imports the data into the database and the other part reads the data from the database. This separation makes it possible to plug in new import scripts. At the moment of writing, there exist 3 supported ways to import data.

Local import is the default way that this system uses.

CTS cloning makes it possible to clone one CTS. Since it relies on the div-configuration, it is currently only compatible with this implementation. In theory, this feature allows community driven decentralized data backups.

The third method relies on a MyCore installation that was used in the project "A Library of a Billion Words" and therefore might require a specific setup. However, together with this setup and using the possibility of timestamp related queries in OAI PMH, we created a self-updating CTS with support for versioning and this way created a persistent CTS with editable content[10].

6 Available Texts

While the implementation was still in progress, it was possible to collect 3 major text collections. For evaluation purposes another corpus containing 100‘000 editions with 1‘281‘272‘600 tokens was generated from random sentences.

[10] A cronjob collects the files, that were changed since the last update via OAI-PMH and timestamps as part of the URNs guarantees persistency.

6.1 DTA (Deutsches Text Archiv)

DTA includes 5136 editions from the German Text Archive of the BBAW in Berlin. All documents are published in 3 editions – .norm, .translit, .transcript – marking different states of normalization. The documents are structured with one citation level (sentence) and include 334‘820‘482 tokens.

6.2 PBC (Parallel Bible Corpus)

PBC is based on the project Parallel Bible Corpus and contains 831 translations of the bible (including 5 different german translations) with 247‘292‘629 tokens. The documents are structured in 3 citation levels (book, chapter, sentence).

6.3 Perseus

Perseus is the dataset from the Perseus project updated in November 2014. This is a well known text collection, containing mainly greek and latin documents that are manually annotated. The documents are structured heterogeneously and the citation depth varies for each document. This corpus adds another 27‘670‘121 tokens and is especially relevant since it is closely related to CTS (see Crane et al., 2014).

7 Evaluation

To evaluate this implementation I used a virtual machine (VM) that was part of our universities network. To make sure that the traffic outside of the VM does not interfere with the results, all requests were sent via localhost. I measured the time it needs to send the request and to get and read the response. Requesting the data from outside the VM would have been a more realistic scenario but would also have included the noise from the network. Since CTS cannot influence the latency of the network in any way, this would also not have been very constructive. Aside from whatever caching strategies are used by Apache Tomcat or MySQL, no caching is used by this implementation. Each response is generated as it is requested.

The test system has a Common KVM processor with one 2,4 GHz core and 1 GB memory. Only one dataset is loaded at any time during the tests and before any test is started, I rebooted the system.

All the URNs of editions were collected and for each one the passage spanning the 2 first URNs on citation level 1 was requested. If there was no second URN on level 1, then level 2 was used. If this was not possible, this edition is ignored.

Depending on the structure of the document, the passages can differ in text length. Passage 1-2 of Luther's “Die Bibel in Deutsch“ spans the books 1 to 2 while the same passage in Schillers “Kabale und Liebe“ as it is structured in this case includes the sentences 1 to 2. This means that the results are not comparable between the datasets. The average number of characters in the generated text passage is given for each diagram.

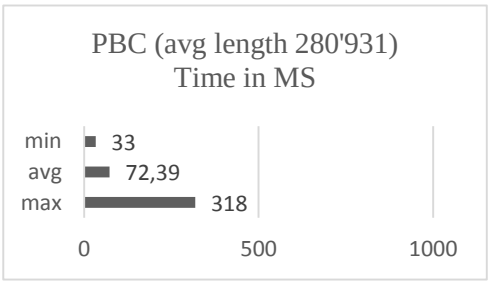

Figure 2, Minimum, average and maximum response times for the PBC dataset

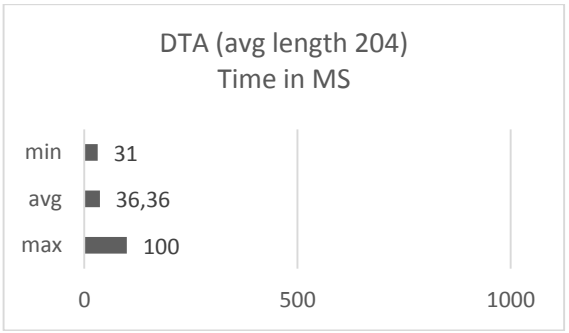

Figure 3, Minimum, average and maximum response times for the DTA dataset

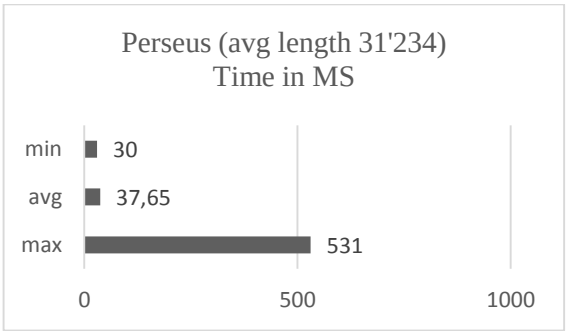

Figure 4, Minimum, average and maximum response times for the Perseus dataset

70/1176 editions of Perseus did not contain any text and were ignored.

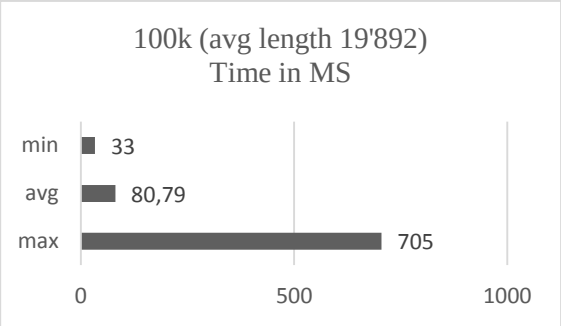

Figure 5, Minimum, average and maximum response times for the 100k dataset

4'800/100'000 documents consist of only 1 sentence and could therefore not deliver a passage 1-2.

In general the results show that the MySQL based implementation performs very well and stays under 1 second in any case. It seems like the response time depends more on the size of the passage that is requested than on the size of the text collection. If the passages length influences the response time, the average response time should reflect this if you limit the result set to 1/4 or 1/10 of the longest or shortest passages in one test run.

	Shortest 1/4 (MS)	Longest 1/4 (MS)
DTA	36,00	37,29
PBC	60,70	91,50
Perseus	33,76	47,86
100K	78,64	83,08

Table 1, Response times for 1/4 of the longest and shortest text passages

	Shortest 1/10 (MS)	Longest 1/10 (MS)
DTA	35.90	37,93
PBC	56,62	98,05
Perseus	33,59	60,24
100K	75,31	81,51

Table 2, Response times for 1/10 of the longest and shortest text passages

Unsurprisingly the length of the requested passage influences the response time (a little bit). However, the differences are small and background noise of the operating system might also have had an impact. It is hard to argue, that such small differences in milliseconds mean anything.

Comparing the results from DTA and PBC, it seems like other factors are also influencing the response time. The 3 longest passages in DTA are 1‘915, 1‘944 and 1‘974 characters long while the 3 shortest passages in PBC are 9‘099, 9‘718 and 9‘793 characters long. Any passage from PBC is longer and also deeper structured than any passage from DTA. Still the PBC CTS could often respond faster than the DTA CTS. This could indicate an influence of the documents structure.

Another interesting value is the response time needed to collect passages spanning complete documents. The following table shows the minimum, average and maximum values for a documents complete passage length and the response times for the corresponding GetPassage request.

	Passage length (in 1000 MS) min | avg | max	Response time (MS) min | avg | max
DTA	0.5 | 444 | 7‘406	32 | 182 | 3‘444
PBC	80 | 163 | 6‘655	57 | 548 | 4‘859
Perseus	35 | 170 | 8‘457	32 | 70 | 3‘088
100k	0.016 | 82 | 438	31 | 86 | 922

Table 3, Minimum, average and maximum response times compared to the minimum, average and maximum passage lengths

Perseus includes the longest document with 8‘457'677 characters and 1‘350'876 tokens. This request also took the maximum time in the dataset with 3‘088 MS. The longest document – and again the document with the highest value for the response time – in DTA is Abelinus Theatrum in its translit edition[11] containing 1’082’893 tokens or 7’406’366 characters.

Considering the hardware limitations and the very good and relatively stable response times, it seems reasonable to include a lot more data into future tests and especially test, at which point this implementation starts to struggle.

Factors that can also be investigated in future evaluations are the influence of the structure of the document and the length of individual text units.

8 Conclusion

This paper marks the release of the MySQL based implementation of the CTS protocol. It introduces features that are exclusive to this software and argues why they are useful additions to the protocol while not contradicting the specifications. Evaluation shows that the performance is very good and sets a baseline for future implementations. It has also shown that this implementation is easily capable of handling a text collection containing one billion words and can be used as a text repository.

[11] urn:cts:dta:abelinus.theatrum1635.de.translit:

Acknowledgements

Parts of the work presented in this paper is the result of the project “Die Bibliothek der Milliarden Wörter”. This project was funded by the European Social Fund. “Die Bibliothek der Milliarden Wörter” was a cooperation project between the Leipzig University Library, the Natural Language Processing Group at the Institute of Computer Science at Leipzig University, and the Image and Signal Processing Group at the Institute of Computer Science at Leipzig University. This project is part of the project Scalable Data Solutions (ScaDS) funded by BMBF. ScaDS is a cooperation project between the Leipzig University and TU Dresden. This projects number is 01/5140148.

Reference

Almas B, Beaulieu M. 2013. "Developing a New Integrated Editing Platform for Source Documents" in Classics in Oxford Journals Literary and Linguistic Computing, Volume 28, Issue 4.

Blackwell C, Smith N. 2014. Canonical Text Services protocol specification. Retrieved from http://folio.furman.edu/projects/citedocs/ctsurn/ and http://folio.furman.edu/projects/citedocs/cts/ 2015, February 19.

Crane G, Almas B, Babeu A, Cerrato L, Krohn A, Baumgart F, Berti M, Franzini G, Stoyanova S. 2014. Cataloging for a billion word library of Greek and Latin. In DATECH 2014: Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage.

Smith N. 2007. An architecture for a distributed library incorporating open-source critical editions. Retrieved from https://wiki.digitalclassicist.org/OSCE_Smith_Paper. 2015, February 19.

Smith N. 2014. Test suite to validate compliance of CTS instances with the CTS API. Retrieved from https://github.com/cite-architecture/ctsvalidator. 2015, February 19.

Tiepmar J, Teichmann C, Heyer G, Berti M and Crane G. 2013. A new Implementation for Canonical Text Services. in Proceedings of the 8th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH).

Project Website of this implementation http://www.urncts.de.

Deutsches Text Archiv http://www.deutschestextarchiv.de/

Parallel Bible Corpus http://paralleltext.info/data/all/

Perseus-CTS/XML https://github.com/PerseusDL/canonical/tree/master/CTS_XML_TEI/perseus