A New Implementation for Canonical Text Services

Jochen Tiepmar
Christoph Teichmann
Gerhard Heyer
Computer Science Department
Leipzig University
`billion-words@e-humanities.net`

Monica Berti
Gregory Crane
Humboldt Chair of Digital Humanities
Leipzig University
`monica.berti@uni-leipzig.de`
`crane@informatik.uni-leipzig.de`

Abstract

This paper introduces a new implementation of the Canonical Text Services (CTS) protocol intended to be capable of handling thousands of editions. CTS was introduced for the Digital Humanities and is based on a hierarchical structuring of texts down to the level of individual words mirroring traditional practices of citing. The paper gives an overview of CTS for those that are unfamiliar and establishes its place in the Digital Humanities research. Some existing CTS implementations are discussed and it is explained why there is a need for one that is able to scale to much larger text collections. Evaluations are given that can be used to illustrate the performance of the new implementation.

1 Introduction

Canonical Text Services (CTS) (Smith, 2009)[1] is a standard that resulted from research in the Digital Humanities community on citation in a digital context. It consists of two parts: an URN scheme that is used to express citations and a protocol for the interaction of a client and a server to identify text passages and retrieve them.

CTS is an attempt to formalize citation practices which allow for a persistent identification of text passages and citations which express an ontology of texts as well as links between texts (Smith and Blackwell, 2012). The same citation scheme can be used across different versions of a text, even across language borders.

All these properties make CTS attractive as an approach to the presentation of large, structured collections of texts. The framework will have little impact however as long as there is no implementation that can scale to the amount of texts currently available for Digital Humanities research and still perform at a level that makes automatic processing of texts attractive. Therefore the implementation of the scheme presented here allows for large repositories without becoming infeasibly slow.

2 Overview of Canonical Text Services

For readers unfamiliar with Canonical Text Services this section provides a short introduction to the CTS protocol and explains its role in the wider context of the CITE architecture. In order to make the explanations given in this section a little more concrete they are followed by example applications of CTS. Before we go into the technical details of the CTS format, a general review of the motivations and approaches behind CTS will be helpful.

CTS incorporates the idea that citations provide an inherent ontology of text passages. A citation of the first word of the first sentence of section 1 in this paper, when made in exactly that way, implies part-whole relationships between the word and the sentence, the sentence and the section and finally the section and the whole article. Canonical Text Services derive their name from the assumption that each text which is included in a CTS repository is associated with a *canonical* way of cit-

[1] `http://www.homermultitext.org/hmt-doc/index.html`

ing it which has been established by a community of researchers working with the text or texts similar to it. Where no such schemes exist they may be defined when a work first enters a repository. These canonical citation schemes are especially common in Classics research from which much of the work on CTS originated. Such schemes often abstract away from any concrete manifestation of a text[2] in favour of schemes that can be applied across different incarnations. Returning to the example task of citing portions of this article, one could cite the same word by referencing a specific line. The latter approach is problematic, since simply printing the article with a different font size could completely change the lines in which words appear. For this reason canonical citations generally rely on logical properties of the text.

Using logical properties of the text implies that citations can be carried over from one specific incarnation to another. It may even be possible to apply the same citation scheme to versions that are written in different languages. This means that different versions of a text can form another element of a hierarchy. Here the part-whole relations are repeated, with different versions of a text belonging to larger groups as explained in section 2.1. When such citations are coupled with a service that resolves the citations and is capable of giving an inventory of all the citations it can resolve, then this can be a powerful tool in Digital Humanities research.

2.1 The CTS URN scheme

We give a short review of the structure of a CTS URN used to identify a text passage[3]. Any canonical reference must start with the prefix:

`urn:cts`

which simply identifies the string as an URN for a canonical citation. This is followed by three parts that contain the main information for every citation. The first of these parts identifies a name space in which the following elements of the citation are meaningful. This part allows for different authorities to define their own schemes for citing works. This section is followed by an identifier of the work that is cited. Finally the URN is completed by a string identifying a text node or passage in the work that is cited, which could correspond to a specific word, a section or even the complete work. To summarize the format of a CTS URN is:

`urn:cts:name_space:work_identifier:passage_identifier`

where the final part can be dropped in order to identify the complete text of a work. The ontology for the work level is given by the URN scheme itself which requires that the work identifier has the structure:

`text_group.work.version.exemplar`

here only the text group part is mandatory. Every other section can be dropped if every following section is also dropped. The text group can be used for any collection of texts that the manager of a CTS service would like to group together such as all the works of an author, all the works from a certain area or all the works created at a certain time. The work portion identifies a specific text within that group. The version part refers to a specific edition or translation of the work and finally the exemplar identifier selects a specific example of a version of a text. The latter three parts of a work identifier correspond to levels of the hierarchy posited by the Functional Requirements for Bibliographic Records (FRBR).

CTS URNs end with a passage identifier. This identifier can further be divided into the parts:

`Citable_Node@Subsection`

where the citable node must correspond to some XML element within the text that is cited. The hierarchy that is used in these nodes is up to the person managing the citations. The hierarchy can be expressed by separating different levels with the delimiter "." and every level can be omitted as long as all following levels are also omitted. A subsection can be used to select a smaller part of a citable node by identifying words to select. There are some additional options that can be used in a CTS URN, among them the option to combine passages into new subsections by using a range operator, and the interested reader is encouraged to consult the official documentation for

[2]For example a specific printing.

[3]For a more extensive discussion see `http://www.homermultitext.org/hmt-docs/specifications/ctsurn/`

the standard.

CTS URNs can be used to identify and connect text passages. A natural task in connection with citations is the retrieval of collections of citations and the text sections associated with them. This task is addressed in the next section.

2.2 The CTS Protocol

This section summarizes the CTS protocol for the retrieval of text sections and citations[4].

The first main request that the protocol defines is:

`GetPassage`

which can be used to retrieve a passage of a text associated to an URN in order to fulfil the actual purpose of a citation. This request also shows one of the main uses of the ontology that is implied in the way works and passages are cited. When a work identifier is "incomplete" then the service is allowed to pick an instance of the work to deliver. When a passage identifier is "incomplete" then the passage delivered includes all passages with identifiers that could complete it.

The second main request is:

`GetValidReff`

which is used to obtain all the citations at a certain level that the current repository can support. Here it is possible to state how many levels should be present in the answer.

The final request that will be discussed in this section is:

`GetCapabilities`

which is used to obtain the different texts known to a server and the way that they can be cited i.e. the structure of their passage identifiers.

With the given requests it is possible to fulfil the main tasks of a citation software: find citations and/or resolve them. Other systems can then build on top of these requests. One example for an architecture that includes CTS capabilities in a wider framework is CITE which is explained in the next section.

[4]More information can be found at `http://www.homermultitext.org/hmt-docs/specifications/cts/`

2.3 CTS in the Context of the CITE Architecture

The Collections, Indexes and Texts (CITE) architecture is a large framework for reference to the objects of study in Digital Humanities[5]. The general design philosophy is to use URNs as a modern way of encoding citations.

Besides providing a general framework for referencing objects and texts, with the latter task being implemented by CTS, CITE also defines a standard for encoding relations between references. An example would be to link a section of a text about geometry to a drawing which it uses as an example. The CITE architecture also includes protocols for resolving and obtaining the references that can be defined within it. Since CTS takes care of citations concerning texts and the tasks associated with them, an implementation of the CTS protocol is an important first step towards a complete implementation of the CITE architecture.

2.4 Example Applications

In this section we review two example applications for the CTS/CITE infrastructure: the generation of digital editions for the Classics and creating editions of so called fragmentary texts.

2.4.1 New Features of Digital Editions

Several features of a true digital edition have already begun to emerge: they have been implemented and they offer demonstrable benefits that justify such added labour as they demand. Each of the following features requires the ability to identify precise words and phrases in particular versions of a work. The CTS/CITE architecture provides a mechanism to support core new functions within the emerging form of born-digital editions:

1. Translators must work with the realization that they are to be aligned to the original and that they will, in fact, help make the original source text itself intellectually accessible to readers with no knowledge of the source language. Every reader should use the Greek and the

[5]More information on CITE can be found at `http://www.homermultitext.org/hmt-doc/cite/index.html`

Latin. Ideally, translators should align their own translations to the source text and provide notes explaining where and why the source text and translation cannot be aligned.

2. We need multi-texts, i.e., editions that can encapsulate the entire textual history of a work so that readers can see not only variants from the manuscript tradition but also variations across editions over time. No reader should ultimately ever have to wonder how a new edition varies from its predecessors. Encapsulating the full textual tradition of every work will take a very long time but we can begin by representing not only textual variants but also providing more than one digitized edition. Again, scholars need the functionality of the CTS/CITE architecture to represent the relationships among different versions of a work.

3. Editors of Greek and Latin texts must encode, at the very least, their interpretations of the morpho-syntactic functions of every word in every text. This should, in fact, impose little extra cost if editors are agonizing, as they should, over every word. Where the editor thinks that there are multiple interpretations that should be considered, then these should be provided along with an explanation of each. The morpho-syntactic analyses are fundamental to modern linguistic analysis and also provide a wholly new form of reading support.

4. All proper names must be aligned to authority lists such as the Pleiades Gazetteer or the Perseus Smith Dictionary of Greek and Roman Biography. We also need conventions for encoding our textual evidence for the relationship between different named entities (e.g., X is the son of Y). Such annotations enable new methods of analysing and visualising our sources with methods from geographic information systems and social network analysis.

5. All instances of textual reuse need to be annotated, including cases where we have reason to believe particular words and phrases are either quoted or paraphrased.

2.4.2 Fragmentary Texts

Among various example applications (Smith, 2009; Smith and Blackwell, 2012; Almas and Beaulieu, 2013), the CTS/CITE Architecture is being implemented by the Perseus Project for representing fragmentary texts of Classical lost authors. By fragmentary texts we mean texts preserved only through quotations and reuses by later authors, such as verbatim quotations, paraphrases, allusions, translations, and so on (Berti et al., 2009; Almas and Berti, 2013).

The first need for representing such texts is to visualize them inside their embedding context and this means to select the string of words that belong to the portion of text which is classifiable as reuse. The CTS/CITE Architecture provides us with a standard identifier syntax for texts, passages, and related objects and with APIs for services which can retrieve objects identified via these protocols (Smith and Blackwell, 2012).

For example, the following set of identifiers might be used to represent a reuse of a lost text of the Greek author Istros, which has been preserved by Athenaeus of Naucratis in the Deipnosophists, (Book 3, Chapter 6)[6] (Almas and Berti, 2013):

`urn:cts:greekLit:tlg0008.tlg001.perseus-grc1:3.6@`Ἴστρος[1]-συκοφάνται[1]

is a CTS URN for a subset of passage 3.6 in the perseus-grc1 edition of the work identified by tlg001 in the group of texts associated with Athenaeus, identified by tlg0008. The URN further specifies a string of text in that passage starting at the first instance of the word "Ἴστρος" and ending at the first instance of the word "συκοφάνται".

`urn:cite:perseus:lci.2`

is a CITE URN identifier for the instance of lost text being reused. This URN identifies an object from the Perseus Collection of Lost Content Items (lci) in which every item points to a specific text reuse of a lost author as it is represented in a modern edition.

[6]For a prototype interface see `http://perseids.org/sites/berti_demo/` (source code at `https://github.com/PerseusDL/lci-demo`)

These URNs represent distinct technology-independent identifiers for the two cited objects, and by prefixing them with the `http://data.perseus.org` URI prefix (representing the web address at which they can be resolved) we create stable URI identifiers for them, making them compatible with linked data best practices [7]:

`http://data.perseus.org/citations/urn:cts:greekLit:tlg0008.tlg001.perseus-grc1:3.6@`Ἴστρος[1]-συκοφάνται[1][8]

`http://data.perseus.org/collections/urn:cite:perseus:lci.2`

The CITE Objects URNs may be organized into various types of collections of data, such as representations of text reuses in traditional print editions, all text reuses attributed to a specific author, all text reuses quoted by a specific author, all text reuses dealing with a specific topic, all text reuses attributed to a specific time period, etc. CITE collections are used to define and retrieve distinct digital representations of discrete objects, including associated meta data about those objects. Example CITE collections used to support the encoding of text reuses for this project include the abstract lost text entities themselves, digital images of manuscripts of the extant source texts that quote those lost texts, commentaries on instances of text reuse and linguistic annotations of the quoted text (Almas et al., 2014).

3 Existing Implementations

There are two general purpose implementations of the CTS protocol that the authors of this paper are aware of. The first is an implementation based on a XML database. This implementation is part of the Alpheios project[9]. Using a XML database seems natural considering the fact that the CTS architecture requires data to take the form of XML files. It would be interesting to compare the performance of this implementation with that of the one that will be presented here, but since the Alpheios tool is not yet complete and has only been tested with a few hundred texts as input[10] any comparison would seem unfair.

The second project to implement the CTS protocol that we are aware of is based on a SparQL endpoint[11]. Similar to the XML based approach the use of SparQL for CTS is intuitive. The part-of relations that are implied by the structure of URNs could easily be modelled with triple expressions. The implementation has not yet been optimized to work with large numbers of input texts and is therefore not suited to a comparison with the tool presented in this paper. While the use of triples to encode the logical relations seems natural, it is necessary to reconstruct all relations already implied by the structure of the URN Strings. This means that there is a potential for optimization that can be exploited by using the structure of these strings in order to store all information implicitly.

4 A New Implementation

So far this paper has argued that Canonical Text Services can provide an important infrastructure for Digital Humanities research. Recently it has also been highlighted (Crane et al., 2012) that repositories of texts such as the Internet Archive[12] have the potential to allow Digital Humanities researchers to work with text collections that encompass billions or even trillions of words. CTS is one tool in the attempt to handle this mass of data without being overwhelmed by it. Since existing implementations of the CTS protocol are not yet able to scale to the data quantities that the Digital Humanities community could provide, we found it necessary to create a new implementation. In order to find out whether our implementation can deal with such a large number of texts, it will be necessary to give an evaluation of performance. This section introduces the main ideas concerning this new implementation and shows that it is indeed capable of the required scaling.

[7]`http://sites.tufts.edu/perseusupdates/beta-features/perseus-stable-uris/`

[8]At the time of this writing, complete implementation of the CTS standard for resolution of passage subreferences at the data.perseus.org address is still pending.

[9]`http://alpheios.net/`

[10]Personal communication with Bridget Almas, the main developer of the Alpheios CTS implementation.

[11]The implementation can be found at `https://github.com/neelsmith/sparqlcts`.

[12]`https://archive.org/index.php`

4.1 Using the Tree Structure of the Data

The main technical problem that needs to be solved in order to generate an efficient implementation of the Canonical Text Services protocol is the efficient mapping of URNs to texts, sections in these texts and the required meta data. Both tasks require the fast mapping of possible prefixes of valid identifiers. There are two obvious solutions to this problem.

The first is the use of a prefix tree or trie in order to be able to deal with underspecified data. This would make it possible to read in the portion of the URN that is specified and then either have a copy of the text or text section associated with this prefix associated with the tree node or construct the necessary information by visiting all daughter nodes. The former choice would be more efficient in terms of nodes visited, but the latter choice would require less memory.

The second option is the use of the lexicographic ordering of the URNs. Consider the set of strings $S = \{a.a.a, a.b.a, a.b.b, a.b.c, a.c.a, \ldots\}$. If all the strings are moved into a data structure that respects the lexicographic ordering of the strings, then all the strings matching $a.b\_^*$[13] can be found by locating the position of the largest string that is lexicographically smaller than or equal to $a.b$ [14] and then visiting all following entries in the data structure until one lexicographically equal to or greater than $a.c$[15] is found. Since MySQL[16] already implements the B-Tree (Bayer and McCreight, 1972) data structure to manage its table indexes we chose this second approach for our implementation. It is used for the work identifiers to select a text that matches a prefix. In the case of passage identifiers all nodes that match a certain prefix are visited and the required text is constructed. The first approach of using prefix trees was also tested but did not lead to a significant decrease in the time or memory requirements since it was not native to the database used.

[13]Here $\_^*$ denotes an arbitrary sequence of characters.

[14]In this case $a.a.a$.

[15]In this case $a.c.a$.

[16]See `www.mysql.com`.

4.2 Putting Everything into a Relational Database

With the problem of handling the URNs solved by tree structures, all that remains is to manage the data that can be found by using the URNs and keeping an index of the URNs. Because the CTS standard requires that the URNs of a work are ordered, this also means that this ordering needs to be preserved. This is achieved by simply keeping a column that stores a numbering. It is ensured that this numbering is sequential without gaps. This means that it is possible to retrieve a certain number of neighbours by simply incrementing and retrieving passages according to this counter. As a result the efficient retrieval of passages that span a range of URNs is possible with only 3 requests, implemented by retrieving the number of the first and last URNs in the range and then merging all text chunks in this range into one passage.

As mentioned earlier, the text of a retrieved section is built up from smaller parts when a node higher in the hierarchy is retrieved. We thereby reduce the amount of memory required since only segments of the data need to be stored. This is unlikely to be a bottleneck, since we assume that the length of a text is not a variable that can grow arbitrarily.

Meta data on the edition level is stored as a simple data entry. For each individual URN we store the language and type of its associated content.

4.3 Evaluation

Here we want to show that our implementation is able to scale to the large amounts of data potentially available to Digital Humanities researchers today and that it can handle the large amounts of data potentially generated by cooperative editing. In order to do this we designed tests that can be used to access the performance of our Canonical Text Services implementation. The following Tests were used:

1. retrieve a list of all editions, then get all the valid URNs and the full passage for each edition

2. collect the first 1000 editions, then obtain the first URN at the lowest level within

each edition and its second neighbour, retrieve the first full word for both[17], finally get the subsection between both words.

Test 1 measures the speed with which the data store can be explored even with a large number of editions and how quickly a passage spanning the whole edition can be constructed. It can be assumed that the time needed to execute will increase with the amount of editions that are added to a repository and with the length of the individual texts.

Test 2 checks how quickly the implementation can find subsections and is not expected to take substantially longer for our implementation as the number of editions increases. It is mainly intended to show that behaviour on single texts is not impacted by the number of editions managed and that the construction of larger passages from elementary chunks is handled efficiently.

Both tests were run by using a small seed of data[18] that was copied repeatedly in order to arrive at the number of necessary editions. The data will be made available. Our implementation ran on a server with a 2.4 GHz CPU and 1GB of RAM. The requests necessary for our tests ran on a different machine in order to factor in the problem of communication. In future tests it would be possible to distribute the requests between different clients to focus more on this point.

Figure 1 contains the results for Test 1. The amount of time taken is linear in the number of editions since every new text was generated once. While the construction of all the texts took several hours for the larger collections, the list of all editions was retrieved within a second or less. There is a surprising spike that could be due to factors external to our program which could have a strong impact on such comparatively short time measurements.

Figure 2 gives the results for Test 2. As expected the behaviour is not greatly impacted by the number of editions in the collection. The variation between the different numbers of editions is within a second for the complete task and the average time needed per retrieval task varies by only ten milliseconds.

[17]A word not containing special characters and longer than 2 characters.

[18]1000 editions.

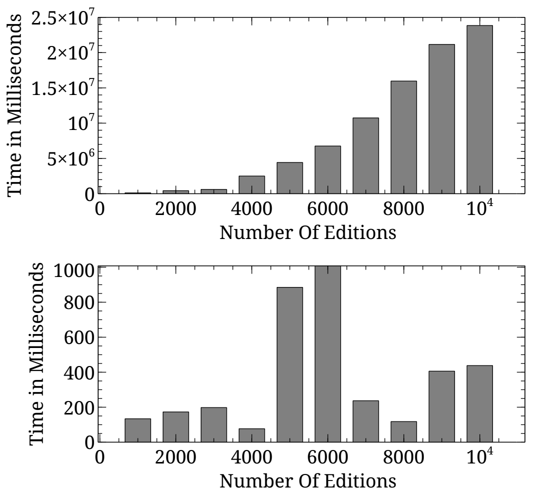

Figure 1: Evaluation results for test 1. The upper graph shows the overall amount of time taken to complete the test for different numbers of editions. The second graph shows the time it took to just retrieve the list of all the editions in the collection.

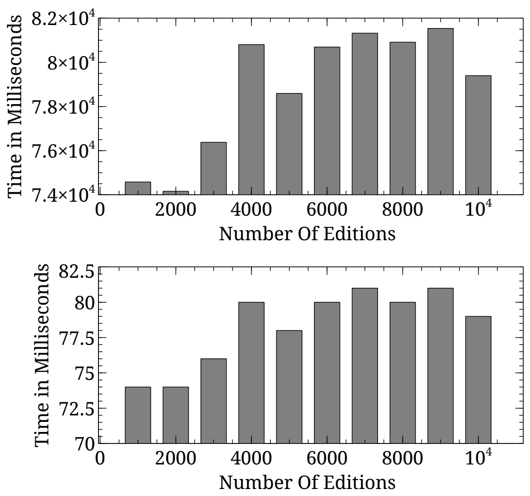

Figure 2: Evaluation results for test 2. The upper graph gives the amount of overall time elapsed in the retrieval of the subsections. The lower graph gives the amount of time needed on average per subsection retrieved. The average was rounded down.

Both measures show a slight increase as the number of editions goes over 3000 but then stabilise.

Overall the experiments show that handling thousands of text is indeed feasible with our implementation on a relatively modest server even for the hardest possible task of reconstructing all the texts in the collection from their smallest parts. Subtasks that do not require retrieving all the texts show little impact from increasing the number of editions.

5 Conclusion

This paper gave a short introduction into the use of the Canonical Text Services Protocol for Digital Humanities research. It also presented a new implementation of the CTS protocol that can handle large amounts of data. The tools that we presented will be made available at:
`http://ctstest.informatik.uni-leipzig.de/`

This address is also used to house the data presented in the evaluation as well as some additional statistics that were generated.

At the time of this writing a new version of the CTS standard was close to completion. As soon as it is published we plan to make our implementation fully compliant. Currently there are still some details in which our implementation diverges from this newest version of the standard. Once this process is complete the next step will be the creation of a permanent CTS capable repository that will be integrated with the CLARIN research infrastructure (Boehlke et al., 2013).

Acknowledgements

Parts of the work presented in this paper are the result of the project “Die Bibliothek der Milliarden Wörter”. This project is funded by the European Social Fund. “Die Bibliothek der Milliarden Wörter” is 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.

References

Bridget Almas and Marie-Claire Beaulieu. 2013. Developing a new integrated editing platform for source documents in classics. *Literary and Linguistic Computing*, 28(4):493–503.

Bridget Almas and Monica Berti. 2013. Perseids collaborative platform for annotating text reuses of fragmentary authors. In *DH-Case 2013. Collaborative Annotations in Shared Environments: metadata, vocabularies and techniques in the Digital Humanities.*

Bridget Almas, Monica Berti, Dave Dubin, Greta Franzini, and Simona Stoyanova. 2014. The linked fragment: TEI and the encoding of text reuses of lost authors. paper submitted to the Journal of the Text Encoding Initiative - Issue 8 - Selected Papers from the 2013 TEI Conference.

Rudolf Bayer and Edward Meyers McCreight. 1972. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189.

Monica Berti, Matteo Romanello, Alison Babeu, and Gregory Crane. 2009. Collecting fragmentary authors in a digital library. In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital Libraries*, pages 259–262.

Volker Boehlke, Gerhard Heyer, and Peter Wittenburg. 2013. IT-based research infrastructures for the humanities and social sciences — developments, examples, standards, and technology. *it - Information Technology*, 55(1):26–33.

Gregory Crane, Bridget Almas, Alison Babeu, Lisa Cerrato, Matthew Harrington, David Bamman, and Harry Diakoff. 2012. Student researchers, citizen scholars and the trillion word library. In *Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries*, pages 213–222.

D. Neel Smith and Christopher W. Blackwell. 2012. Four URLs, limitless apps: Separation of concerns in the Homer Multitext architecture. In *Donum natalicium digitaliter confectum Gregorio Nagy septuagenario a discipulis collegis familiaribus oblatum: A Virtual Birthday Gift Presented to Gregory Nagy on Turning Seventy by His Students, Colleagues, and Friends*. The Center of Hellenic Studies of Harvard University.

D. Neel Smith. 2009. Citation in classical studies. *Digital Humanities Quarterly*, 3(1).