`LitText`: Building Literary Corpora for Computational Literary Analysis A Prototype to Bridge the Gap between CL and DH

Andrew U. Frank, Christine Ivanovic
Geoinformation TU Wien, Comparative Literature University Vienna
Gusshausstr. 27-29 1040 Wien Austria, Sensengasse 3a 1090 Wien Austria
frank@geoinfo.tuwien.ac.at, christine.ivanovic@univie.ac.at
{frank, christine}@gerastree.at

Abstract

The design of `LitText` follows the traditional research approach in digital humanities (DH): collecting texts for critical reading and underlining parts of interest. Texts, in multiple languages, are prepared with a minimal markup language, and processed by NLP services. The result is converted to RDF (a.k.a. semantic-web, linked-data) triples. Additional data available as linked data on the web (e.g. Wikipedia data) can be added. The DH researcher can then harvest the corpus with SPARQL queries. The approach is demonstrated with the construction of a 20 million word corpus from English, German, Spanish, French and Italian texts and an example query to identify texts where animals behave like humans as it is the case in fables.

Keywords: linked-data, semantic-web, RDF, SPARQL, digital humanities

1. Introduction

Literary analysis, like all other fields of Digital Humanities (DH) based on the evaluation of texts in natural languages, can benefit from advances in computational linguistics (CL) (Pustejovsky et al., 2017a). Many language tools for linguistic analysis are available, and produce results which could be useful for corpus-based literary analysis. Moretti (2005) and others have demonstrated viable approaches to computational literary analysis, and have produced valuable results (e.g. (Fischer et al., 2017)). Projects like DARIAH (Blümm et al., 2016), LAPSS (Ide et al., 2015), and others have brought together CL tools in unified frameworks to make the results of CL easier to use. Despite this, the uptake of CL in literary studies has been slow so far, and very few of the immense number of literary publications show use of CL.

More efforts in bridging the gap between CL and DH are required. It could be beneficial to take a step back from technical issues, and instead focus on the research practices of literary studies researchers. These usually consist of (1) selecting texts which will be analyzed; and (2) reading the texts and identifying parts with properties relevant to the research goal by underlining them. The approach is probably similar for most text-based DH. A tool for DH should therefore be presented following this model of (1) building a corpus and (2) selecting text pieces of interest.

CL can identify parts with specific text properties in a corpus quicker than a human reader, it can process texts systematically, and it can browse a corpus larger than any human could peruse. The DH researcher has *only* to describe what properties the relevant text parts should have. Lord et al. (2006) followed an approach where the researcher gives some example texts she is interested in, and the search engine then finds all text similar to the given one, with the risk that the automatic detection misses the intention of the researcher. `LitText` is based on explicitly formulated queries using SPARQL (see section 8.).

`LitText` focuses on the methods available for researchers to query processed texts. It reduces manual preprocessing to a minimum, to make it possible to build substantial, research-focus-specific corpora (e.g. of 19th century English novels) which includes texts in multiple languages. `LitText` is centered around a query facility to describe the properties relevant text pieces should have, and retrieves these together with metadata about the texts; other data useful for the analysis and available in linked-data format can be added. Results can be further processed using spreadsheet software.

The DH researcher only has to master two kinds of coding:

- a simple markup language to add metadata to text files and to delimit the literal text of interest;
- a standardized, widely available query language (SPARQL) for which extensive instructional materials and a wide choice of query processing software are available.

Results can be displayed in a regular web browser, or further processed with spreadsheet software. Many DH researchers are already familiar with such programs for sorting, computing, counting etc.

A method to build a specialized corpus with a query facility that can retrieve pieces of text and then manipulate them in a spreadsheet is a processing model that is conceptually close to current research methods in text-based humanities. Empowering text-based DH researchers to build their own corpora processed on their own hardware under their control can also reduce fears of copyright infringement (for a technical solution, see: (Pustejovsky et al., 2017a; Zeng et al., 2014)).

There are three novel aspects in `LitText`, all justified by the focus on "ease of use" for the DH researcher:

1. Processing texts in multiple languages, hiding the technical complexity from the DH researcher.
2. Storing the result of the NLP analysis as linked data (Berners-Lee et al., 2009; Manola et al., 2004).

3. Using the standardized query language SPARQL(Harris et al., 2013) to search for relevant text parts.

Section 2 shows an example task and query which will be used to demonstrate the power of `LitText` in Section 8. Section 3 compares the approach with other, similar efforts. Section 4 will cover the preparation of the text, while Section 5 addresses the processing of text with NLP. In Section 6 we discuss adding a processed text to the corpus, and in Section 7 the inclusion of additional data. Section 8 shows how the example query is formulated and executed. Section 9 reports on actual performance, and moves on to future extension and conclusions.

2. Example Task for Literary Analysis

The literary analysis we envision is a very broad field, and CL support must go beyond mere statistical style analysis (Holmes and Kardos, 2003). The running example query used is to identify "animal fables", considered here as texts in which animals are reported to behave like humans, i.e. they think and use intelligent communication. Examples are e.g. the classic fables ascribed to Aesop; systematic search reveals that modern literary texts include similar literary tropes. Such an analysis requires more language analysis than can be provided by simple word frequency analysis etc.

The proposed algorithm to identify such texts finds sentences where the subject is an animal, and the verb expresses either "think" or "communicate". The approach is to lemmatize the text and to construct dependency trees in which subject and verb can be identified and checked whether the subject noun has "animal" among its hypernyms, as well as the verbs "think" or "communicate". We will show how such a query can be formulated in section 8..

Processing natural language is error-prone, reflecting the flexibility and context-sensitivity of human language. The outlined approach is obviously affected by errors in NLP tools when identifying the dependency structures, i.e. the subject and the verb of a sentence. The current system only considers sentences where the subject is a noun; `LitText` does not yet use the coreference analysis produced by some NLP tools, e.g. Stanford coreNLP (Recasens et al., 2013). The method currently implemented relies on the hypernyms in WordNet (Fellbaum, 1998) and considers only the first, most often encountered WordNet sense. It produces omission errors by failing to identify all nouns describing animals, as well as commission errors by identifying nouns which are not animals. Fortunately, for literary analysis, such errors can be tolerated; the human interpreter reads the identified sentences, decides whether they are useful for his research and discards irrelevant ones.

3. Design Compared to Previous Work

DARIAH is an European infrastructure incorporated in 2014, connecting multiple nationally funded projects to collect tools useful for a wide range of humanity fields to allow them to analyze natural-language text (http://www.dariah.eu/). It uses standard tools (Manning et al., 2014) to process text and allows for the storage of input text and results. The goal is to overcome non-trivial technical installation difficulties, and assure the availability of NLP tools — a first step for researchers from the humanities to apply these tools to their specific problems.

The somewhat comparable project LAPPSgrid (Ide et al., 2015) has similar goals (http://www.lappsgrid.org/). It adds considerations for copyright restrictions attached to the texts to permit the holding of copyright-restricted texts, their processing, and making results available in accordance with those restrictions (e.g. HathiTrust) (Pustejovsky et al., 2017b).

`LitText` splits the work of the DH researcher into two tasks: (1) building the corpus, and (2) searching the corpus. The software to build the corpus applies NLP tools to text which is marked and retains the result for later analysis; the same set of processes is applied to all texts; addition of NLP tools requires changes in the code. The corpus is represented as RDF triples and is searched with standard SPARQL query processors (Harris et al., 2013). The researcher can add new text to the store and they are automatically processed and entered in the corpus, ready to be queried.

4. Preprocessing

Preparing a text for processsing by NLP tools is the first impediment for literary scholars trying to use NLP. The goal of `LitText` is to facilitate the collection of texts into a corpus relevant to a particular investigation. In order to collect many texts into a corpus, the effort to prepare the texts must be minimal, as well as flexible to allow use of different sources. The researcher must (a) add metadata to identify the text and other aspects a researcher would want to include in a query; (b) distinguish the part of text that is to be included in the analysis (here called "literal text") from other text parts in the file; and (c) identify the language a text piece is written in (multiple languages per file are allowed).

In `LitText`, we opt for a simplistic markup for text files. Preparing a text means finding a source, assuring proper UTF-8 encoding (e.g. with iconv), adding some markup for metadata (title, author, year of publication), and separation of the actual text from other text material included in the file (e.g. table of content, preface, original file name) - all of which can be done easily in any text editor.

```
.originalFile http:.../174.txt.utf-8
.publication 1890
.Language: English
.Title: The Picture of Dorian Gray
.Author: Oscar Wilde

.ignoreTo
*** START OF THIS ...
Produced by .... HTML version by .....
.ignoreEnd

The artist is the creator of beautiful things. To
...
```

Much of the material we used comes from Project Gutenberg (Hart, 1992) and we prepared a small tool that produces most of the markup automatically and fixes some issues with UTF-8 encoding; downloading and marking up a literary text from Project Gutenberg takes less than

2 minutes. Corresponding tools for other frequently used sources can be constructed with any suitable programming language (all that is needed is to read a text input and add some markup).

5. Processing

A text with markup is processed by NLP programs, and the result translated into the N-Triples format, which is a W3C recommendation for the semantic web (Beckett, 2014). The conversion of a text file to subject-predicate-object triples is carried out in two phases:

1. The metadata, the layout, and the surface texture (e.g. sections, paragraphs) and, if desired, the full text can be captured and translated into the N-Triples format for later use in query formulation (see example).

2. The text is isolated and sent to language-specific NLP services. Using the markup, the text is split into parts in a single language. Differences in language-specific NLP processing are handled internally. At the moment, `LitText` is prepared to handle English, German, French, Spanish and Italian text, using Stanford Core NLP (Manning et al., 2014) and TinT (Palmero Aprosio and Moretti, 2016). Five server processes are set up at different ports; in addition, German text is lemmatized using TreeTagger as a service at an additional port (Schmid, 2013). Adding a language requires an adaptation of the `LitText` program and setting up a server for the language specific NLP.

An example of the translation of a text head and a line of text as RDF triples:

```
<http://gerastree.at#kafka-kafka_urteil_1913>
    <http://gerastree.at/lit_2014#author>
      "Franz Kafka"@de ;
    <http://gerastree.at/lit_2014#dedikation>
      "für Fräulein Felice B."@de ;
    <http://gerastree.at/lit_2014#titel>
      "DAS URTEIL"@de ;
    <http://gerastree.at/lit_2014#untertitel>
      "EINE GESCHICHTE VON FRANZ KAFKA"@de ;
    a <http://gerastree.at/lit_2014#Werk> .

<http://gerastree.at#kafka-kafka_urteil_1913/L00003>
    <http://gerastree.at/layout_2017#lineNumber> 3 ;
    <http://gerastree.at/layout_2017#lineText>
      "DAS URTEIL"@de ;
    <http://gerastree.at/layout_2017#pageNumber>
      "[54/0002]" ;
    a <http://gerastree.at/layout_2017#Line> ;
    <http://www.w3.org/2000/01/rdf-schema#partOf>
      <http://gerastree.at#kafka-kafka_urteil_1913> .
```

The result of the NLP process in XML format is analyzed and translated into triple structure, preserving the produced Treebank codes (Taylor et al., 2003) as much as possible. Dependency codes are already from the Universal Dependency set (Schmid, 2013) (see http://universal dependenciesorg/). The minimal differences between NER codes (e.g. for German text we find I-LOC, etc. and for Spanish text LUG) can be normalized without loss. The processing produces a file in the N-Triple format, i.e. one linked data triple - subject, predicate, object - per line. Tools to convert the result to a more human-readable format like Turtle are available (e.g. rapper by Beckett (2001)).
The programs are designed to regularly scan a directory and convert any new markup files into N-Triples files. Adding a new markup file to a directory is sufficient to trigger the conversion — and later the inclusion in the corpus — automatically.

6. Storing

The N-Triples are stored in any triple store, some available as open-source or free. We use Jena triplestore (Khadilkar et al., 2012), using utilities we constructed for loading the N-Triple files into the store which relies on standardized HTTP protocol SPARQL update commands (Harris et al., 2013); they should work with any other triplestore. The programs are designed to incrementally build a store by scanning all N-Triples files in a directory, and storing new ones into the triple store.

7. Additional Data Sources

WordNet data for hypernyms are required to answer the example query. WordNet can be downloaded in triple format (McCrae et al., 2014) based on the lemon schema and loaded in a triplestore. The resources, especially the hypernym relations, can be accessed in SPARQL queries.
We opted to store the set of nouns in hypernym relations to "animal" — or, for verbs, "think" (in wordnet "cerebrate´) or "communicate" — separately with a SPARQL update query for later use in the analysis. Storing the hypernym relation reduces the complexity of later query formulation and speeds up processing (see section 8.). An example update query to insert a list of the nouns which are hypernyms to "animal", considering only the first WordNet sense (the SPARQL prefix abbreviations are given in the appendix):

```
insert
  {graph <http://gerastree.at/al>
    { ?lexentry lit:nounClass wn:Animal .
      ?lexentry lemon:writtenRep ?lem .
    }
  }
using <http://gerastree.at/wn>
WHERE
{
  ?lexentry wn:part_of_speech wn:noun .
  ?lexentry lemon:sense ?sense .

  ?sense wn:sense_number ?sensenum .
  filter (?sensenum = 1) .
  ?sense lemon:reference ?synset.

  ?synset wn:hypernym+ ?obj .
  ?obj rdfs:label ?lo .
  filter (?lo = "animal"@eng ) .

  ?lexentry lemon:canonicalForm ?cf .
  ?cf a lemon:Form .
  ?cf lemon:writtenRep ?lem .
}
```

Adding other semantic web data resources is equally straightforward; of particular interest are likely geographic gazetteers, the movie database, and Wikipedia.

8. Querying

A triplestore offers a HTTP protocol query facility, the so-called SPARQL endpoint, where queries are submitted, and answers are returned. SPARQL is a very powerful, general-purpose triplestore query language (Harris et al., 2013). It is somewhat patterned after SQL but logically simpler because storage is always in binary relations (Manola et al., 2004; Prud'Hommeaux et al., 2008).

An example query shows the potential to formulate complex queries: ”find all texts in the store in which “animals” behave like humans, e.g. “think” and “communicate” ”. The result includes the sentences in which such literary forms occur, as well as the author, the title of the text, the subject, and the verb.

```
SELECT ?author ?werk  ?sentform ?subjlemma ?verblemma
FROM <http://gerastree.at/c>
#the corpus of literary text
FROM <http://gerastree.at/a1>
#the graph with the hypernyms
WHERE  {
  ?dep nlp:dependency "NSUBJ" .
  ?dep nlp:dependent ?subj .
  ?dep nlp:governor ?verb.

  ?subj nlp:lemma3 ?subjlemma.
  ?verb nlp:lemma3 ?verblemma.

  ?verb nlp:pos ?catv.
  filter (strstarts (?catv, "V")).

  ?subj nlp:pos ?cats.
  filter (strstarts (?cats, "N")).

  ?lexs lemon:writtenRep ?subjlemma .
  ?lexs lit:nounClass wn:Animal .
  ?lexv lemon:writtenRep ?verblemma .
  {?lexv lit:nounClass wn:Communicate}
      union {?lexv lit:nounClass wn:Cerebrate } .

  ?subj rdfs:partOf ?sent .
  ?sent nlp:sentenceForm ?sentform .

  ?sent rdfs:partOf ?snip .
  ?snip rdfs:partOf ?para .
  ?para lit:inWerk ?werk .
  ?werk lit:author ?author .
}  order by ?werk
```

The literal text is stored in graph `c` and the extracted hypernym relations in graph `a1`. The query progresses top-down: first it fills the variable `?dep` with a node in a dependency tree which is a subject-noun phrase, and takes the subject and the verb part (`?subj` and `?verb`). Next, check that `?verb` is indeed a verb and `?subj` a noun, and retrieve (for subject and verb) the corresponding lemmata as `?subjlemma` and `?verblemma`. The next steps are tests whether the subject is an animal, and whether the verb indicates either communicating or thinking. The remainder of the query is tracing from the sentence to the text (`?werk`) and the author.

The output of the result is stated in the `SELECT` clause, and gives the qualified sentences as well as the author and title of the text containing that sentence.

The processing approach is somewhat simplistic as it omits constructions where the subject is an animal but not a noun - using coreference information would probably reduce such errors. It incorrectly includes constructions where the first sense of the noun is an animal, but the noun is used in a different sense. Sentences in which a dog barks, a cat meows etc. are excluded with special rules (see 9.1.). Texts which should be considered “animal fables” (in the present narrow definition) are characterized by frequent occurrences, while other texts have only a few stray occurrences of qualifying constructions.

Queries to obtain text statistics (distinct word count, length of sentences, etc.) can be formulated with SPARQL `GROUP BY` and `COUNT`.

9. Simplification of Query with Preprocessing

The SPARQL query in the previous section 8. is likely not workable for the intended DH researcher. Several preprocessing steps are possible to simplify the query close to the CPQ style(Hardie, 2012) query
`[pos="noun" & hypernym="animal"] .`

9.1. Construct Hypernym List and Classify Words in Text

A list of all words in WordNet which are hyponyms to some interesting wordclasses (e.g. animal, artefact, communicate) are easy to construct. Using such a list to classify all tokens in a text and mark each with the hypernym of the interesting classes allows to test a token with e.g.
`?tok a nlp:Animal.`
The list of “interesting” wordclasses contains entries like:
`_:animal lit:wordClassPattern wn:Animal ; lit:wordClassRep "animal"@eng ; lit:wordClassKind wn:noun.`
and is easily adaptable.
Stored hypernym relations provide a hook for corrections for undesirable relations. For example, to exclude “mademoiselle” from the class “Animal” requires a triple
`wn:mn wn:isNotClass "mademoiselle"@eng .`
or to exclude “dogs bark” from communication
`wn:bark wn:isNotNounVerbNoun "dog"@eng; wn:isNotNounVerbVerb "bark"@eng.`

9.2. Use Universal Dependency and POS tags

Using the Universal Dependency and PoS tags removes differences between the coding of texts of different languages. CoreNLP (in version 3.9.0) produces UD dependency annotations but not POS tags from the UD.POS list. We can translate the PennTreeTags produced by coreNLP to UD.POS tags or better, use a pipeline which directly produces UD.POS tags (Straka and Straková, 2017).
The dependency graph can be represented in triples with one triple per dependency, at the cost of deviating from the XML structure produced by CoreNLP processors.
E.g. `?verb nlp:nsubj ?subj.`

9.3. Simplified queries

These two preprocessing steps searching for nouns which are hyponyms of animal becomes

```
select ?word
{ ?tok nlp:pos "NOUN" .
  ?tok a nlp:Animal.
  ?tok nlp:lemma3 ?word.
}
```

which is not far from the desired CPQ like query. The more complex query for sentences with animals which think or communicate becomes:

```
select str(?s1) str(?v1) str(?sf)
FROM <http://gerastree.at/english1>
FROM <http://gerastree.at/tk13>
where  {
  ?verb nlp:nsubj ?subj .
  ?subj a  wn:Animal.
  {?verb a wn:Cerebrate } union
    {?verb a wn:Communicate } .
```

```
  ?subj nlp:lemma3 ?sl .
  ?verb nlp:lemma3 ?vl .
  ?subj rdfs:partOf ?sentence .
  ?sentence nlp:sentenceForm ?sf .
}
```

The query is finding a verb with a noun subject (first line) and then to check that the subject is an animal (second line), the verb is in the wordclasses cerebrate or communicate (third line). The remaining lines find the lemmata for subject and verb and the text of the sentence, in which the construct occurs.

10. Performance

Overall performance, even on dated and limited hardware, is satisfactory. We tested on a PC with an Intel i5 processor and 3.2 GHz with 24 GB of memory (the memory is mostly necessary for NLP processing of literary texts like "Ulysses" by Joyce; ordinary grammatical text is processed using less than 8 GB). The java engine was started with 6 GB of memory for each of the coreNLP services and with 10 GB for the jena-fuseki triplestore. The example corpus we built consists of about 266 literary texts (mostly books) for a total of 20.1 million words.

- Download and markup of one text from Project Gutenberg takes less than 2 minutes.
- NLP processing a literary text runs at about 70 words per second for an English text including co-reference analysis, 190 words/second for a German text (with separate lemmatization process) and 530 words/sec for Italian text.
- A word in text produces about 10 triples.
- Storing the triples in the triplestore processes about 4,000 triples per second.
- The preprocessing of the full corpus takes a total of about 2 days.
- the text files take 0.67 GB, the processed and stored triples 2.2 GB and the triplestore 75 GB.

Processing the example query 9.3. shown earlier takes 6 seconds (corpus of English text of 2 M words and hardware as before).
The code can be downloaded from a github account (`andrewufrank.github.com/LitText`) and combined with a suitable triple store (e.g. `jena.apache.org/download/index.cgi`).
The SPARQL endpoint is accessible on `http://nlp.gerastree.at:3030`.

11. Future Work

A number of improvements are envisioned:

- integrate hypernym data for other languages without making the query writing more complex is challenging.
- use ideas from Abstract Meaning Representation Language (AMR) (Vanderwende et al., 2015; Winiwarter, 2015; Xue et al., 2014) to simplify queries across texts in multiple languages.
- use co-reference data produced by NLP programs.
- add other language servers to `LitText` (e.g. Dutch, Japanese, Farsi)
- build multi-lingual corpora in order to facilitate computational comparative literature studies (Ivanovic, 2017); a prototype is to be developed with original texts and translations author Yoko Tawada, who writes both in Japanese and German.
- include safeguards for the protection of copyrights, to allow integration of data not in the public domain.

12. Conclusions

`LitText` is a suite of programs for researchers in DH to build multi-language corpora for projects as linked-data, and to explore them using the standardized SPARQL query language. The design goal was to minimize the amount of technical detail a DH researcher would need to use the system. The GUI should follow a conceptual model familiar to most DH researchers:

- build the corpus to be studied
- parts of text of interest to the research

Texts require minimal preparation with a simple markup language to add metadata about the text and identify the sections of text to be studied; multiple languages in a single file are possible, and the NLP processing is transparent to the user. The results are stored in a triple store, and can be queried with the standardized query language SPARQL. A DH researcher needs to learn a minimum set of NLP codes, e.g. a few Universal Dependency codes. Further processing of results is possible with spreadsheet software. Texts can be added anytime simply by adding files with markup to a directory. When queries are repeated, they search the increased corpus. The motto "all methods applied to all texts" (Ivanovic and Frank, 2015; Ivanovic and Frank, 2016) is respected: the full corpus can be reprocessed in a few days and the reevaluation of queries takes only minutes.
Technically, `LitText` revolves around a linked data triple store and the corresponding SPARQL update and query language. The main program takes a text file with markup, sends the pertinent text to the required, language-specific NLP processes, and converts the result to a triple format. Utilities to download text from research-specific sources and to automate markup as far as automatically possible are small extensions. Storage and query can be done with any SPARQL 1.1 conformant program; loading triples into triple stores and accessing the SPARQL endpoint uses the HTTP protocol and can be done in a web browser. Additional resources which are available as linked data can easily be incorporated (e.g. WordNet, Wikipedia). The use of standards makes a plethora of additional software available and reduces the number of utilities which must be specially programmed.

Acknowledgments

We thank an anonymous reviewer who suggested a CQP style query formulation (Hardie, 2012) which inspired improvements in the query.

Bibliographical References

Beckett, D. (2001). Raptor RDF parser toolkit. *University of Bristol.[Online]. Available: http://www. redland. opensource. ac*.

Beckett, D. (2014). RDF 1.1 -triples: A line-based syntax for an RDF graph. *W3C Recommendation*.

Berners-Lee, T., Bizer, C., and Heath, T. (2009). Linked data-the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22.

Blümm, M., Schmunk, S., Gietz, P., Horstmann, W., and Hütter, H. (2016). Vom Projekt zum Betrieb: Die Organisation einer nachhaltigen Infrastruktur für die Geisteswissenschaften DARIAH-DE. *ABI Technik*, 36(1):10–23.

Fellbaum, C. (1998). *WordNet*. Wiley Online Library.

Fischer, F., Göbel, M., Kampkaspar, D., Kittel, C., and Trilcke, P. (2017). Network dynamics, plot analysis: Approaching the progressive structuration of literary texts. In *Digital Humanities 2017 (Montréal, 8–11 August 2017). Book of Abstracts*.

Hardie, A. (2012). CQPweb – combining power, flexibility and usability in a corpus analysis tool. *International journal of corpus linguistics*, 17(3):380–409.

Harris, S., Seaborne, A., and Prudhommeaux, E. (2013). SPARQL1.1 query llanguage.

Hart, M. (1992). The history and philosophy of project gutenberg. *Project Gutenberg*, 3:1–11.

Holmes, D. I. and Kardos, J. (2003). Who was the author? an introduction to stylometry. *Chance*, 16(2):5–8.

Ide, N., Pustejovsky, J., Cieri, C., Nyberg, E., DiPersio, D., Shi, C., Suderman, K., Verhagen, M., Wang, D., and Wright, J. (2015). The language application grid. In *International Workshop on Worldwide Language Service Infrastructure*, pages 51–70. Springer.

Ivanovic, C. and Frank, A. U. (2015). Corpus-based research in Computational Comparative Literature. In Francesco Mambrini, et al., editors, *Proceedings of the Workshop on Corpus-Based Research in the Humanities (CRH). Warsaw, Poland*, pages 69–78.

Ivanovic, C. and Frank, A. U. (2016). Korpusanalyse in der computergestützten Komparatistik. In *Digital Humanities deutsch (DHd)*.

Ivanovic, C. (2017). Die Vernetzung des Textes: Im Möglichkeitsraum digitaler Literaturanalyse. *Zeitschrift für digitale Geisteswissenschaften*.

Khadilkar, V., Kantarcioglu, M., Thuraisingham, B., and Castagna, P. (2012). Jena-hbase: A distributed, scalable and efficient rdf triple store. In *Proceedings of the 2012th International Conference on Posters & Demonstrations Track-Volume 914*, pages 85–88. CEUR-WS. org.

Lord, G., Smith, M. N., Kirschenbaum, M. G., Clement, T., Auvil, L., Rose, J., Yu, B., and Plaisant, C. (2006). Exploring erotics in Emily Dickinson's correspondence with text mining and visual interfaces. In *Digital Libraries, 2006. JCDL'06. Proceedings of the 6th ACM/IEEE-CS Joint Conference on*, pages 141–150. IEEE.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., and McClosky, D. (2014). The Stanford CoreNLP natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60.

Manola, F., Miller, E., McBride, B., et al. (2004). RDF primer. *W3C recommendation*, 10(1-107):6.

McCrae, J., Fellbaum, C., and Cimiano, P. (2014). Publishing and linking WordNet using lemon and RDF. In *Proceedings of the 3rd Workshop on Linked Data in Linguistics*.

Moretti, F. (2005). *Graphs, maps, trees: abstract models for a literary history*. Verso.

Palmero Aprosio, A. and Moretti, G. (2016). Italy goes to Stanford: a collection of CoreNLP modules for Italian. *ArXiv e-prints*, September.

Prud'Hommeaux, E., Seaborne, A., et al. (2008). Sparql query language for rdf. *W3C recommendation*, 15.

Pustejovsky, J., Ide, N., Verhagen, M., and Suderman, K. (2017a). Enhancing access to media collections and archives using computational linguistic tools. In *CDH@ TLT*, pages 19–28.

Pustejovsky, J., Verhagen, M., Rim, K., Ma, Y., Ran, L., Liyanage, S., Murdock, J., McDonald, R. H., and Plale, B. (2017b). Enhancing access to digital media: The language application grid in the HTRC data capsule. In *Proceedings of the Practice and Experience in Advanced Research Computing 2017 on Sustainability, Success and Impact*, PEARC17, pages 60:1–60:3, New York, NY, USA. ACM.

Recasens, M., de Marneffe, M.-C., and Potts, C. (2013). The life and death of discourse entities: Identifying singleton mentions. In *HLT-NAACL*, pages 627–633.

Schmid, H. (2013). Probabilistic part-of speech tagging using decision trees. In *New methods in language processing*, page 154. Routledge.

Straka, M. and Straková, J. (2017). Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDpipe. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99.

Taylor, A., Marcus, M., and Santorini, B. (2003). The penn treebank: an overview. In *Treebanks*, pages 5–22. Springer.

Vanderwende, L., Menezes, A., and Quirk, C. (2015). An AMR parser for English, French, German, Spanish and Japanese and a new AMR-annotated corpus. In *Proceedings of NAACL-HLT*, pages 26–30.

Winiwarter, W. (2015). Jamred: a japanese abstract meaning representation editor. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*, page 11. ACM.

Xue, N., Bojar, O., Hajic, J., Palmer, M., Uresova, Z., and Zhang, X. (2014). Not an interlingua, but close: Comparison of English AMRs to Chinese and Czech. In *LREC*, volume 14, pages 1765–1772.

Zeng, J., Ruan, G., Crowell, A., Prakash, A., and Plale, B. (2014). Cloud computing data capsules for non-consumptiveuse of texts. In *Proceedings of the 5th ACM workshop on Scientific cloud computing*, pages 9–16. ACM.

Appendix

The common prefix values for SPARQL are:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX wn: <http://wordnet-rdf.princeton.edu/ontology#>
PREFIX nlp: <http://gerastree.at/nlp_2015#>
%PREFIX li: <http://def.seegrid.csiro.au/isotc211/iso19115/2003/lineage#>
prefix lemon: <http://lemon-model.net/lemon#>
prefix lit: <http://gerastree.at/lit_2014#>
prefix layout: <http://gerastree.at/layout_2017#>

%prefix dcterms: <http://purl.org/dc/terms/>
%prefix dc: <http://purl.org/dc/elements/1.1/>
%prefix pgterms: <http://www.gutenberg.org/2009/pgterms/>
%prefix marcrel: <http://id.loc.gov/vocabulary/relators/>
%prefix dcam: <http://purl.org/dc/dcam/>
%prefix cc: <http://web.resource.org/cc/>
```