Douglas Cunningham, Petra Hofstedt, Klaus Meer, Ingo Schmitt (Hrsg.): INFORMATIK 2015
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015

GIET: Generic Information Extraction using Triple Store Databases

Ronald Smith Djomkam Yotedje[1]

Abstract: Information extraction (IE) is a natural language processing (NLP) task which aims at retrieving specific type of information from natural language text. Before the actual extraction of information, a natural language text usually undergoes some processing, such as part-of-speech (POS) tagging. Nowadays, information extraction finds practical use in smart phones, which for example are able to identify a phone number in an SMS text and provide a link for either calling that number or storing it. There exist various approaches in extracting information from text, e.g., pattern-based information extraction which uses predefined rules to extract entities and ontology-based information extraction. These approaches often concentrate on a particular domain whose information is to be extracted. This paper provides a light-weight, unified, and scalable methodology which uses a generic approach for extracting information. This approach does not depend on any domain or rule.

1 Introduction

Information extraction (IE) is a technology based on analyzing natural language text with the aim of extracting specific type of information [Cu06, HDG00]. Depending on the system, the information usually extracted may be directly presented to the user requiring it, or saved in a database, or even used as a base for indexing purposes in information retrieval (IR) applications such as web-based search engines, e.g., Google and Duckduckgo. People sometimes tend to confuse between information extraction and retrieval. Compared to an information extraction system, an information retrieval system is a technology which acquires information resources that are related to the user's requirements. These information resources are often obtained from a large bank of information resources.

An example of an IE task is named entity recognition (NER). Named entity recognition searches and retrieves entities such as names, places, and dates. This is used in applications such as Google Mail, where dates are automatically recognized in texts and represented in such a way that one can easily create a calendar event from them.

There exists different approaches to IE, such as ontology-based information extraction, where unstructured or semi-structured natural language texts are processed through a mechanism guided by ontologies to extract certain types of information. The output is then presented using ontologies [WD10]. We present a light-weight, unified, and scalable methodology for IE, which makes use of triple store databases to store extracted information as triples. This provides a back-end architecture, which has the main function of combining various IE approaches, such as Named Entity Recognition and Co-reference

[1] Universität Bremen — FB3, AG Rechnerarchitektur, Bibliothekstrasse 1 (MZH), 28359 Bremen, smith@cs.uni-bremen.de

Resolution. We further use SPARQL as query language for extracting information through the back-ends.

The remainder of the paper is divided into five sections. Section 2 gives an overview of related work. Section 3 describes the main idea of this work followed by an algorithm. In Sections 4 and 5 we present respectively the system developed in this work and the experimental evaluation of this system, and finally, Section 6 gives an overall conclusion of the work.

2 Related work

Most of the fields related to our work has to do with natural language, linguistic annotation, and semantic web. Sonal Gupta [GM14] developed a system with two major functions. At first, it provides an information extraction system capable of extracting entities from a given text using patterns with bootstrapping. An advantage of this methodology is that the patterns used for entities extraction are not fixed but generated from the text. Further, the system provides a way of visualizing the extracted entities and patterns. The methodology described in [CJ11] is an amelioration of standard template-based information extraction, which uses predefined templates to fill semantic roles. This methodology removes the requirement of having a fix template, but instead introduces the learning of domains' template from the input text.

[ARR13] proposes a methodology for extracting information based on ontology, i.e., an ontology is used along to guide the extraction process. This methodology has a common point with the methodology proposed in this paper in that, it is generic, since it can be applied to any domain. [Ar13] proposes a methodology for automatic information extraction from natural language texts, based on the integration of linguistic rules, multiple ontologies and inference resources, integrated with an abstraction layer for linguistic annotation. [Ru12] proposes a way of representing linguistic annotation using the RDF framework.

NIF[3] and LemonWordNet[4] are both existing RDF wrappers for Standford Core NLP and WordNet respectively. Our methodology is not limited to a special nlp tool, as it allows someone to easily wrap an existing nlp tool for integration with other wrappers in the overall system. [KLA13] lets a user describe corpus-wide extraction tasks in a declarative language and permits him to run interactive rule refinement queries. The methodology used here has a similarity with ours, since the extraction actually takes place after issuing database queries. But the difference is that, we use a triple store database (TSDB) since it doesn't require a schema. This allows an easy customization of back-ends, which will be explained in detail in Section 4. Further, [KLA13] uses the declarative language AQL[5] for extraction. Our system uses the SPARQL RDF query language (SPARQL),[6] since the extracted information is stored in a TSDB.

[3] http://persistence.uni-leipzig.org/nlp2rdf
[4] http://wordnet-rdf.princeton.edu
[5] http://bigdataconsultants.blogspot.ch/2013/07/aql-annotation-query-language.html
[6] http://www.w3.org/TR/rdf-sparql-query

Mathias Soeken et al. [So14] proposed a triple store database information extraction method, which also uses the SPARQL query language for extraction. Our methodology proposes an improvement of this work with a back-end architecture, which enables different information extraction approaches to be combined.

3 Abstract concept

In this section we describe the main idea of our work and the algorithm used to develop the GIET system.

3.1 Main Idea

We propose a new methodology for extracting information from texts using triple store databases. This method is not only light-weight, but also unified and scalable. Light-weight is achieved by providing a back-end architecture with a simple interface that requires almost no dependencies. The method is also said to be unified since upon call of the simple interface from the user, all required back-ends are activated. The activated back-ends are responsible to retrieve the necessary information. Finally, the ability to extract just the necessary information as requested by the user makes the method scalable.

Example 1: Consider the following sentence: "Smith goes to Africa occasionally." The task is to retrieve named entities, and their types from the sentence. Further consider we have three different back-ends available. One back-end extracts words, another extracts named entities and the last one extracts part-of-speech tags from the sentence. The SPARQL query looks like the following:

```
SELECT ?word ?namedEntity
WHERE {
   ?w word ?word.
   ?w is ?namedEntity.
}
```

Each predicate of the query statement represents the type of information needed. There are two query statements. The first statement consists of a subject which has a word and the second statement says that this same subject should be a named entity. The word and the named entity will be saved respectively in the variables *word* and *namedEntity*. Since just named entities and their types are needed, the back-end extracting part-of-speech tags will not be considered anymore from the interface. The two other back-ends now retrieve corresponding information from the sentence and save them as triple in the triple store database. The following shows the triples stored in the database:

```
<goes-2> word "goes"
```

```
<Africa-4> word "Africa"
<Africa-4> is "LOCATION"
<Smith-1> word "Smith"
<Smith-1> is "PERSON"
<to-3> word "to"
<occasionally-5> word "occasionally"
```

Notice that each word in the sentence has a corresponding word-triple. This is very important, as a sentence may contain the same word more than one time. Applying the above query to the above triple store database yields the following query result:

?namedEntity	?type
Africa	LOCATION
Smith	PERSON

Tab. 1: Query result for example 1

Figure 1 gives an abstract overview of how this method is used to extract information. As observed in the picture, inputs are a sentence as information source and a query. The simple interface executes an algorithm as described in the next section. In this algorithm, we determine from the query the type of information to be retrieved, and which back-ends to involve for retrieving this information. The back-ends to be involved for retrieval depends on the type of information determined from the query. A back-end aims at retrieving information from the input sentence. There are no restrictions to how or which information is extracted from the sentence. This means back-ends can retrieve information as is, or enrich this information by using external dependencies. The advantage is that the simple interface does not know about the existence of any external dependencies, which makes it less coupled to the back-ends.

Figure 1 also shows a triple store database, where information is stored as triple after being retrieved from the sentence. Afterwards, the query gets executed on the triple store database to yield the requested information from the user.

3.2 Algorithm

This section describes the algorithm executed in the simple interface of the back-end architecture. As mentioned in the previous section, inputs to the algorithm are a sentence as information source and a query. Algorithm 1 shows the pseudo code of the algorithm.

The first step of the algorithm consists of determining the type of information from the query. As a next step, iteration takes place over the list of available back-ends where only those back-ends are saved which can retrieve the information as determined from the query. Further, each required back-end retrieves information from the sentence and stores it as triple in a triple store database. A triple represents the kind of relation which exists between two entities. Since the database used to store information is a triple store database, SPARQL was chosen as query language for extraction.

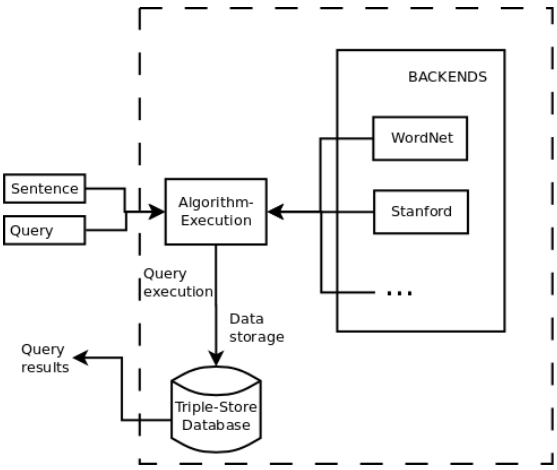

Fig. 1: Abstract overview of information extraction using our methodology

Algorithm 1

Input: $sentence \in String$
Input: $query \in String$
Output: $result \in QueryResult$
1: $informationType \leftarrow determineInfosType(query)$
2: $requiredBackends \leftarrow \{\}$
3: **for** $backend \in availableBackends()$ **do**
4: **if** $canRetrieveInfosOfType(backend, informationType)$ **then**
5: $requiredBackends \leftarrow requiredBackends \cup backend$
6: **end if**
7: **end for**
8: $listOfTriples \leftarrow \{\}$
9: **for** $backend \in requiredBackends$ **do**
10: $triples \leftarrow extractInformation(backend, sentence)$
11: $listOfTriples \leftarrow listOfTriples \cup triples$
12: **end for**
13: $store \leftarrow storeInDatabase(listOfTriples)$
14: $result \leftarrow execute(query, store)$

4 GIET system

As seen in Section 3, our main idea is to provide a more generic way of extracting information from a text. Below we present some freely available tools used.

- **Application service:** The application service[7] is a scalable tool realized in the Java programming language, which helps to develop client-server applications using the transmission control protocol (TCP) with very less effort. In using this tool, one has

[7] https://github.com/rosmith/application-service

to concentrate only on the logic of the application to be developed. Everything else like the network communication is handled by the application service, provided it is appropriately configured.

- **Stanford CoreNLP:** CoreNLP from the Stanford university is a Java annotation pipeline framework, which provides most of the common NLP steps, from tokenization through to co-reference resolution [Ma14].
- **Apache Jena:** Apache Jena[8] is a free and open source Java framework for building semantic web and linked data. The main goal of this tool is to provide facilities for both the application and the system programmers. The application programmer can access and manipulate graph data through higher level interfaces whereas a system programmer wishing to manipulate data as triples has a simple minimalist view of RDF graphs [Ca04].
- **Clear NLP:** This is another Java framework for NLP, which was developed by Jinho [CP11]. This NLP tool has almost same components as the Stanford CoreNLP, like part-of-speech (POS) tagging and dependency parsing, just that both use different approaches. Clear NLP provides an additional component known as semantic role labeling (SRL) which is not yet provided by the Stanford CoreNLP.
- **Extended Java WordNet Library (ExtJWNL):** WordNet is an online lexical database designed for use under program control. This database links English nouns, verbs, adjectives, and adverbs to sets of synonyms that are in turn linked through semantic relations that determine word definitions. WordNet contains more than 166,000 word form and sense pairs, where a word form is represented by a string of ASCII characters, and a sense is represented by the set of (one or more) synonyms that have that sense [Mi95]. ExtJWNL [9] is a free and open source Java framework for creating, reading and updating dictionaries in WordNet format.
- **Behind The Name (BTN) & Jsoup:** BTN[10] is a website which on request, gives the etymology of a name, which can be either a person's name or surname or even a place's name. Unfortunately, there is no existing library at the moment which enables one to use the functionalities of BTN programmatically. For that reason, one can make use of the library Jsoup,[11] which is a Java library for working with real-world HTML, to navigate in the HTML tree of a search result page and get some relevant information like the gender or description of the searched name.

The GIET system is implemented in Scala[12]. The source code is available in GitHub[13] under the MIT License, and a documentation can be found in the webpage[14], with an

[8] https://jena.apache.org/documentation/inference/
[9] http://extjwnl.sourceforge.net/
[10] http://www.behindthename.com
[11] http://jsoup.org
[12] http://scala-lang.org
[13] https://github.com/rosmith/giet
[14] http://giet-nlp.de

interactive user interface to play around with. Fig 2 shows the workflow of information extraction of the GIET system.

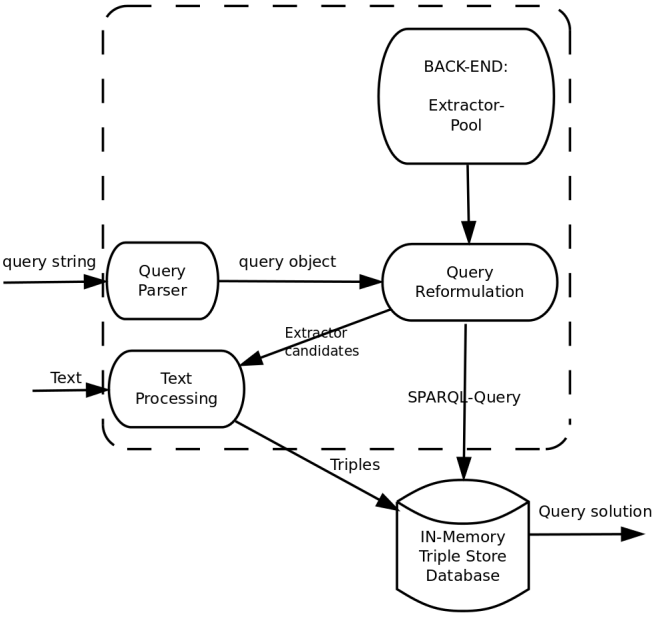

Fig. 2: Information extraction workflow of the GIET system

The GIET system provides a scalable back-end architecture. This allows combining different information extraction approaches to form a unified IE system. The back-end architecture consists of a pool of built-in extractors. An available extractor interface allows an application programmer to provide a custom information extractor, which retrieves information according to the needs of the application programmer. It should be emphasized that annotations can be represented by both data and object properties. According to the task done by an extractor, the programmer decides which representation best suites. Following are some important built-in extractors in the GIET system with their corresponding namespaces in brackets:

- **Word extractor (word)** wraps the Stanford tokenizer[15] and aims at saving token annotations of the sentence.
- **Tagger extractor (tag)** wraps the Stanford tagger [To03] and aims at saving part-of-speech tag annotations for each token in the sentence.
- **Dependencies extractor (dependency)** wraps the Stanford parser [KM03] and aims at saving typed dependencies annotations such as nominal subject (nsubj), adjective (jj), etc.
- **Coreference extractor (coref)** wraps the Stanford co-reference resolution system [Le11] and aims at saving existing co-reference annotations in the text.

[15] http://nlp.stanford.edu/software/tokenizer.shtml

- **Named entity extractor (is)** wraps the Stanford named entity recognition system [FGM05] and aims at saving named entity annotations.
- **SRL extractor (srl)** wraps the SRL component of the ClearNLP system [FGM05] and aims at saving semantic role annotations.
- **BTN extractor (btn)** uses both BTN and Jsoup to save people's gender and name definition. This shows that the GIET system is very flexible and allows someone to add new information, which is not present in the sentence.
- **WordNet extractor (wordnet)** uses the extended JWNL to save lexical fields information for the syntactic categories nouns, verbs, adjective, and adverbs. An example of a lexical field[16] for a verb is *motion* which contains verbs walking, flying and swimming.

All of the above extractors are available in the extractor pool at the beginning of the extraction. Each extractor has a unique namespace, with which it can be addressed. Extraction of information using the GIET system occurs as follow:

- The query string has a very simple format which abstracts the SPARQL query language. This query string is first parsed to obtain a query object. A query of all tokens in a sentence will look like the following:

```
SELECT ?token
WHERE {
   ?x word ?token.
}
```

  In the above query string, one can observe that the triple statement has a predicate without namespace. The predicate itself is a namespace and hence corresponds to the extractor with the namespace *word* which is the *WordExtractor*, and will be the only extractor retained for extraction, the rest will be kept uninstantiated in the pool.

- The query object helps to infer which extractors (built-in and custom) from the pool are needed for extraction. After resolving the needed extractors, the query object is reformulated to the following full SPARQL query.

```
PREFIX giet: <http://www.giet-nlp.de/annotator#>

SELECT ?token
WHERE {
  ?x giet:word ?token.
}
```

- The resolved extractor candidates are then used to retrieve information from the text. The resulting triples are then saved in the in-memory triple store database which is provided by the apache jena API as described above.
- The generated SPARQL query is then applied on the triple store database to extract the required information.

[16] https://wordnet.princeton.edu/man/lexnames.5WN.html

5 Use case description

For the use cases, we used the mixed case Test Data 4 provided by the Message Understanding Conference (MUC-4)[17]. The MUC was initiated by the former Naval Ocean Systems Center (NOSC) and financed by the Defense Advanced Research Projects Agency (DARPA) with the aim of encouraging the development of new methods for extracting information [GS96]. The test data consists of one hundred texts, all talking about terrorist activities in Latin Amerika.

Example 2: To illustrate some important use cases, we make use of the following two sentences from the second text of the test data.

> *The prosecutor of the military court trying the people responsible for the killing of 124 inmates accused of terrorist activities held in the Lurigancho Prison on 19 June 1986 has asked for 25 years imprisonment for the person responsible for the killing.*
>
> *Prosecutor Juan Carbone Herrera requested the 25 years imprisonment for General Rolando Cabezas Alarcon of the Republican Guard "for ordering the shooting of 124 of the San Pedro prison inmates (Lurigancho)."*

The above sentences contain two named entities of type person, two of type location, one of type date, and two others of type number and duration, which are actually referring to the same entity, i.e. 124 and 25 years, respectively. With the following query, we aim at extracting any entity being dependent of a verb. We are also interested in the verb governing this entity, the part-of-speech of both the entity and its' governor (verb), the type of entity it is, and the position of both in the sentences. According to the text, we expect to have one result.

```
select ?dependent ?governor ?namedEntity ?pos ?gpos ?tag ?gtag
where {
  // nsubj relates a verb and its subject.
  ?x nsubj ?y. // nsubj acts as an object property
  ?x word ?governor.
  ?y word ?dependent.
  ?y is ?namedEntity.
  ?x position ?gpos.
  ?y position ?pos.
  ?x tag ?gtag.
  ?y tag ?tag.
}
```

It should be noted that, the user is the one providing the sentence and the query. The query mostly depends on what the user needs to extract from the sentence. After giving

[17] http://www-nlpir.nist.gov/related_projects/muc/index.html

the above sentence and user-defined query to the system, the system creates a query object from the above query string, and infers from it, which of the extractors in the pool should be instantiated for retrieving the required information.

In our case, we have eight query statements with five different namespaces from different extractors. These extractors will retrieve all corresponding information from the sentences and save them in the triple store database. After applying the query on the database, we receive exactly one result as expected:

?dependent	?governor	?namedEntity	?pos	?gpos	?tag	?gtag
Herrera	requested	PERSON	4	5	NNP	VBD

Tab. 2: Query result for example 2

Further constraints could be applied, such as restricting the named entities to be of type location, which will of course have no result, or restricting the governor to belong to the lexical field *communication*, e.g.:

```
select ?dependent ?governor ?namedEntity ?pos ?gpos ?tag ?gtag
where {
  ?x nsubj ?y.
  ?x word ?governor.
  ?y word ?dependent.
  ?y is ?namedEntity.
  ?x position ?gpos.
  ?y position ?pos.
  ?x tag ?gtag.
  ?y tag ?tag.
  ?w communication ?d.
  ?w word ?c.
  FILTER (?governor = ?c)
}
```

Due to the three last lines of the above query, the WordNet extractor will be instantiated to retrieve words belonging to the lexical field *communication* and the filter makes sure that there is at least one word belonging to the lexical field *communication* which is the same as the governor.

Our system relies much on external dependencies, such as Stanford CoreNLP and WordNet, which have been used to implement the built-in extractors. This makes the quality of the query results to be strongly dependent on these external dependencies. In the above query, we could further constrain the query, such that the dependent have the gender male. However, there will be no result with this additional constraint, since the dependent in the sentence is *Herrera*, which is not available in the database of Behind The Name, used by the BTN extractor.

6 Conclusion

In this paper, we provided a generic methodology for information extraction. This methodology provides a scalable back-end architecture, which enables to combine different approaches for extraction. The methodology is light-weight due to the unawareness of the eventual dependencies used in the back-end architecture and the fact that with a single query, different information can be extracted, also makes the methodology unified.

In future work, we plan to develop a declarative domain specific language (DSL) for querying. The motivation is that, not every programmer knows what is SPARQL and how to use it, so a DSL will give the chance to these programmers to make use of the advantages of our system, without having to know how a SPARQL query works. We also intend to output the results of our system according to state-of-the-art formats for representation of linguistic annotation like NIF.

Acknowledgement

Allow me to dedicate my acknowledgment of gratitude toward the following significant advisors and contributors:

First and foremost, I would like to thank Dr. Mathias Soeken for his support and encouragement. He kindly read my paper and offered invaluable detailed advices on grammar, organization, and the theme of the paper. Second, I would like to thank Prof. Dr. Rolf Drechsler and Prof. Dr. phil. John A. Bateman to read my paper and to provide valuable advices.

References

[Ar13] de Araujo, Denis A.; Rigo, Sandro J.; Muller, Carolina; Chishman, Rove: Exploring the inference role in automatic information extraction from texts. In: The Joint Workshop on NLP & LOD and SWAIE. pp. 33–40, September 2013.

[ARR13] Anantharangachar, Raghu; Ramani, Srinivasan; Rajagopalan, S: Ontology Guided Information Extraction from Unstructured Text. International Journal of Web & Semantic Technology (IJWesT), 4(1):19, January 2013.

[Ca04] Carroll, Jeremy J.; Dickinson, Ian; Dollin, Chris; Reynolds, Dave; Seaborne, Andy; Wilkinson, Kevin: Jena: Implementing the Semantic Web Recommendations. In: The Thirteenth International World Wide Web Conference. Association for Computing Machinery (ACM), New York, New York, pp. 74–83, May 2004.

[CJ11] Chambers, Nathanael; Jurafsky, Dan: Template-Based Information Extraction without the Templates. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL). pp. 976–986, June 2011.

[CP11] Choi, Jinho D.; Palmer, Marther: Transition-based Semantic Role Labeling Using Predicate Argument Clustering. In: Proceedings of the ACL 2011 Workshop on Relational Models of Semantics (RELMS 2011). 2011.

[Cu06] Cunningham, H: Information Extraction, Automatic. Encyclopedia of Language & Linguistics, 5:665–677, 2006.

[FGM05] Finkel, Jenny Rose; Grenager, Trond; Manning, Christopher D.: Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In: Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005). pp. 363–370, 2005.

[GM14] Gupta, Sonal; Manning, Christopher D.: SPIED: Stanford Pattern-based Information Extraction and Diagnostics. In: Proceedings of the ACL 2014 Workshop on Interactive Language Learning, Visualization, and Interfaces (ACL-ILLVI). 2014.

[GS96] Grishman, Ralph; Sundheim, Beth: Message Understanding Conference: A Brief History. MUC-6, pp. 466–471, 1996.

[HDG00] Humphreys, Kevin; Demetriou, George; Gaizauskas, Robert: Two Applications of Information Extraction to Biological Science Journal Articles: Enzyme Interactions and Protein Structures. In: Pacific Symposium on Biocomputing. pp. 490–501, 2000.

[KLA13] Killias, Torsten; Loeser, Alexander; Andritsos, Periklis: INDREX: In-Database Distributional Relation Extraction. In: ACM 6th International Workshop on Data Warehousing and OLAP (DOLAP). 2013.

[KM03] Klein, Dan; Manning, Christopher D.: Accurate Unlexicalized Parsing. In: Proceedings of the 41st Meeting of the Association for Computational Linguistics. pp. 423–430, 2003.

[Le11] Lee, Heeyoung; Peirsman, Yves; Chang, Angel; Chambers, Nathanael; Surdeanu, Mihai; Jurafsky, Dan: Stanford's Multi-Pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task. In: Proceedings of the CoNLL-2011 Shared Task. 2011.

[Ma14] Manning, Christopher D.; Surdeanu, Mihai; Bauer, John; Finkel, Jenny; Bethard, Steven J.; McClosky, David: The Stanford CoreNLP Natural Language Processing Toolkit. In: Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations. Association for Computational Linguistics, Baltimore, Maryland, pp. 55–60, June 2014.

[Mi95] Miller, George A.: WordNet: A Lexical Database for English. COMMUNICATIONS OF THE ACM, 38(11):39–41, November 1995.

[Ru12] Rubiera, Emilio; Polo, Luis; Berrueta, Diego; El Ghali, Adil: TELIX: An RDF-Based Model for Linguistic Annotation. In (Simperl, Elena; Cimiano, Philipp; Polleres, Axel; Corcho, Oscar; Presutti, Valentina, eds): The Semantic Web: Research and Applications, volume 7295 of Lecture Notes in Computer Science, pp. 195–209. Springer Berlin Heidelberg, 2012.

[So14] Soeken, Mathias; Harris, Christopher B.; Abdessaied, Nabila; Harris, Ian G.; Drechsler, Rolf: Automating the Translation of Assertions Using Natural Language Processing Techniques. In: Forum on Specification and Design Languages - Proceedings. o.A., 2014.

[To03] Toutanova, Kristina; Manning, Christopher D.; Klein, Dan; Singer, Yoram: Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: Proceedings of HLT-NAACL. pp. 252–259, 2003.

[WD10] Wimalasuriya, Daya C.; Dou, Dejing: Ontology-based information extraction: An introduction and a survey of current approaches. Journal of Information Science, 36(3):306–323, 2010.