Knowledge Representation of Requirements Documents Using Natural Language Processing

Aaron Schlutter
Technische Universität Berlin
DCAITI, Germany
aaron.schlutter@tu-berlin.de

Andreas Vogelsang
Technische Universität Berlin
DCAITI, Germany
andreas.vogelsang@tu-berlin.de

Abstract

Complex systems such as automotive software systems are usually broken down into subsystems that are specified and developed in isolation and afterwards integrated to provide the functionality of the desired system. This results in a large number of requirements documents for each subsystem written by different people and in different departments. Requirements engineers are challenged by comprehending the concepts mentioned in a requirement because coherent information is spread over several requirements documents. In this paper, we describe a natural language processing pipeline that we developed to transform a set of heterogeneous natural language requirements into a knowledge representation graph. The graph provides an orthogonal view onto the concepts and relations written in the requirements. We provide a first validation of the approach by applying it to two requirements documents including more than 7,000 requirements from industrial systems. We conclude the paper by stating open challenges and potential application of the knowledge representation graph.

1 Introduction

In practice, complex systems are often developed in terms of several loosely coupled subsystems. Requirements specifications often exist, if at all, only on the level of subsystems. Mostly written in natural language, the requirements documents contain the specific knowledge necessary to develop the single subsystem. With the advent of more complex features that require several subsystems to interact, it is getting harder for requirements engineers to comprehend the relevant knowledge that is spread over several requirements documents.

Automotive software systems are a good example of this phenomenon. The different subsystems within a car are usually developed by completely separate teams that employ their own requirements engineering process and documentation with a focus on their own subsystem and limited knowledge about the relations to other subsystems. In an earlier study, we have shown that developers of single subsystems of an automotive system are not aware of more than half of the dependencies that their subsystem has to other subsystems [VF13].

In this paper, we describe a natural language pipeline that transforms a set of natural language statements (e.g., requirements) into a knowledge representation graph. The purpose of this knowledge representation graph is to summarize and structure all concepts and relations contained in the requirements over all subsystem specifications. The graph represents this knowledge by a set of triples inspired by RDF (Resource Description

Framework)[1]. To create the graph, our NLP pipeline first performs a number of preprocessing steps on the requirements to normalize the data and then uses Semantic Role Labeling [CM04] to extract single triples. The triples are afterwards merged into a single knowledge graph.

As an early validation, we applied the pipeline to two requirements documents, one academic example with 150 requirements and one industrial document with over 7,000 requirements. Both documents contain requirements from the automotive domain. The resulting knowledge graph for the smaller document contains 345 nodes, while the graph for the larger document consists of over 13,000 nodes. These results show how much conceptual information is contained in requirements documents and motivates why knowledge representation may help analyzing the semantics embedded in the requirements documents.

Although we have not yet elaborated on specific applications of this knowledge graph, we envision benefits for assessing the amount of implicit knowledge in requirements documents, trace link generation, and support for reviewing requirements documents.

At the end of this paper, we detail some of the challenges that need to be solved to leverage the full potential of this knowledge representation including identification of semantic similarity and representation of causal relationships.

2 Background

Knowledge representation is useful in many disciplines, whenever collecting, managing and sharing information and facts is necessary. The form of knowledge representation may vary from expressing knowledge by natural language to technical solutions that aim at creating complex logical views on the information. As a technical term, the topic evolved in the context of artificial intelligence and reasoning systems in the 1950's. Knowledge representation focuses on the depiction of information that enables computers to solve complex problems.

Knowledge representation supports users and computers to handle large amount of information. The basic idea of knowledge representation systems is to store complex information or facts in a knowledge base and make them available for various application. Therefore, searching the knowledge base is a crucial task. How information can be queried depends on the way the information is stored. Reasoning systems, for example, aim at extracting information to deduce new facts by an inference engine. Deduced information can be added to the knowledge base. These systems and their knowledge representation are often very formal and limited to solve specific constraints.

One possibility to represent knowledge is provided by the RDF. In RDF, each statement/fact is represented by a triple that contains a subject, an object, and a predicate describing the relationship between them. The parts of an RDF triple are also categorized into different classes, e.g., a subject can be an instance of the class *resource* or of the class *literal*. One triple's object can be another triple's subject which allows to concatenate multiple triples. A set of triples can be represented as an RDF graph where subjects and objects are represented as nodes and predicates as edges.

In addition to RDF, it is possible to define an RDF schema (RDFS) and build up an ontology. Such a schema contains specifications about the classes used in an RDF model. Therefore, subclasses are defined, e.g., of resource or literal specifying a limited value range or defining permissible relationships to other classes.

Knowledge bases can be built manually or extracted automatically from other knowledge representations. If the source for such an extraction is natural language, NLP pipelines are usually used for information extraction. In case of building up an ontology, this process is called ontology learning and is often semi-automatic. Due to the ambiguity of natural language, those pipelines usually result in knowledge representations that are imprecise and incomplete up to a certain point. Despite these shortcomings, such structural semi-formal knowledge bases enable computers to execute a wider range of algorithms.

However, for knowledge contained in industrial requirements specifications, a (semi-)automatic extraction is often the only chance for proper knowledge representation.

2.1 Related Work

Borgida et al. stated already in 1985 that knowledge representation is the basis for requirements engineering [BGM85]. They argue that in addition to the functional specification of a system, it is essential to have more knowledge about its environment, i.e., "knowledge of terms, technical and scientific rules, everyday procedures and conventions". The proposed Requirements Modeling Language (RML) can be seen as a predecessor of an

[1] `https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`

ontology while they proposed to use such a model as the basis to exchange knowledge in a formal, yet easy to understand, way.

OntoLearn [VFN13], a pipeline for ontology learning, consists of several steps to learn the taxonomy of an ontology. At first, they identify and extract terms inside a corpus, search for their hypernyms using additional knowledge like web glossaries and documents (e.g., WordNet[2]), and build up a "noisy hypernym graph". Afterwards this graph is filtered by given upper terms of a specific domain. Lastly, this graph is pruned to a tree-like taxonomy to reduce cycles and hypernyms for most nodes. In case important edges were pruned, they are optionally recovered at the end. This workflow results in a directed acyclic graph.

Browarnik and Maimon proposed to build graphs via the concept "from clauses or subsentences to RDF triples and RDFS" [BM15]. They argued that the common attempt for ontology learning, the so called layer cake, is not useful because each step (layer) analyzes the whole corpus at a single point of view which is very often faulty and this error is passed on to all depending layer. This results in a generally low recall and precision (less than 0.6). To circumvent this shortcoming, they propose alternative models for ontology learning by reviewing and evaluating existing papers.

One use of ontologies in the automotive context is recommendation-based decision support [HK16]. They are searching for similar already finished safety analyses to support a human expert in hazard analysis and risk assessment, prescribed by the ISO 26262 standard. While the ontology containing extensive expert's knowledge, spreading activation as a highly configurable semantic search algorithm is used to query the knowledge base for relevant information.

Dermeval et al. report on the use of ontologies in requirements engineering in their systematic literature review [DVB+16]. They reviewed 67 papers from academic and industrial application contexts, dealing with different types of requirements and identified and categorized typical problems solved by ontologies in different RE phases. Most citations deal with problems like (1) ambiguity, inconsistency, and/or incompleteness, (2) requirements management/evolution and (3) domain knowledge representation. While only 34% of the papers reuse existing ontologies in their solutions, most of them specified their own ontology. The largest number of papers in the review rely on textual requirements as the RE modeling style, especially in the specification phase.

3 Natural Language Processing Pipeline

Figure 1 shows our pipeline to build a graph from natural language statements. The core parts of our pipeline, Stanford CoreNLP [MSB+14] and SENNA [CWB+11] for semantic role labeling, are pipelines themselves and will be described in detail in the following subsections along with the graph building.

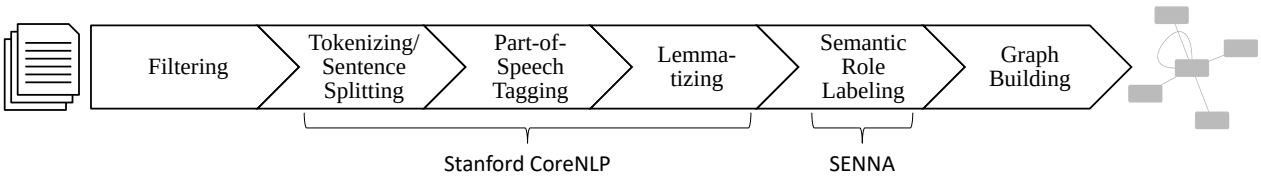


Figure 1: NLP pipeline to transform NL requirements specifications to knowledge representation graphs

The current pipeline only supports English as natural language. Therefore, we do some filtering to remove statements, that contain languages other than English. Information representation such as tables or images are not interpretable by the pipeline and are also ignored.

The graph building at the end of the pipeline requires tokens annotated with semantic roles, lemma and part-of-speech. Lemmatizing requires already annotated part-of-speech tokens, while semantic role labeling only requires tokens itself. Since SENNA as a standalone tool requires tokenized sentences as input, Stanford CoreNLP is positioned prior to it.

3.1 Stanford CoreNLP

The Stanford CoreNLP is a collection of common NLP tasks, assembled in a pipeline[3]. We are using parts of it, particularly the tokenization, sentence splitting, part-of-speech tagging and lemmatizing (morphological analysis).

[2] https://wordnet.princeton.edu/
[3] https://stanfordnlp.github.io/CoreNLP/

The first two tasks determine all tokens (words, punctuation marks, etc.) and sentences in a natural language text. Part-of-speech tagging categorizes the tokens by their grammatical role inside of a sentence, e.g., as noun, verb or article. Lastly, each word is annotated with its lemma, a basis version depending on the part-of-speech tag. For example, the lemma of "walking" (verb) is "walk" (verb), the lemma of "walkers" (noun) is "walker" (noun). In contrast to this, stemming of these words would result in "walk" as the word stem.

3.2 Semantic Role Labeling

Semantic role labeling (SRL) [CM04] is a sentence-based NLP task that consists of two steps. In the first step, all predicates or verbs of a sentence are determined. In the second step, all semantic arguments of each verb are associated/classified with their roles within this sentence. In general, the first argument is called the agent, the second the patient, all other semantic roles depend on the verb. For example, "John broke the window with a rocket", contains the verb "broke". Its arguments are [$_{\text{A0}}$John], [$_{\text{A1}}$the window] and [$_{\text{AM-MNR}}$with a rocket]. The numbered arguments would be the same even if the syntax of the sentence would change, e.g., "[$_{\text{A1}}$The window] was [$_{\text{V}}$broken] [$_{\text{AM-MNR}}$with a rocket] [$_{\text{A0}}$by John]". The semantic roles are predefined in a database called PropBank[4]. As stated in [CM05, Table 1], most of the identified propositions have at least one argument, about two-third also have a second argument.

The CoNLL (Conference on Computational Natural Language Learning) shared task for semantic role labeling was first published in 2004 [CM04] and extended in 2005 [CM05]. Within these competitions, the PropBank corpus was annotated with semantic roles as training data. While the top submission of the first shared task receives a precision of 72.43% and a recall of 66.77%[5], the top submission in 2005 already performs with a precision of 81.18% and a recall of 74.92% with an enlarged corpus[6]. While the competitions are conceived to run just for several months and further submissions are no longer possible, there are still new systems evolving, which are compared under those conditions [GCW+16, FTGD15, PP14].

We use SENNA[7] as state-of-the-art implementation for SRL tagging. It uses a unified neural network architecture to perform SRL and achieved an F1 score of 75.49% for CoNLL in 2005.

3.3 Building the graph

As last step of the pipeline, an exporter builds a graph from the annotated natural language. The exporter searches the text for SRL verbs and their arguments. If at least a first and second argument exist, it creates an RDF triple with the first SRL argument as subject, the second one as object, and the verb as predicate. All other arguments are stored as metadata of the triple inside the relation. If there is no second argument, a reflexive relation is built and exported.

SRL annotates the tokens in a sentence as they are (like "the window" or "broke"). However, this is not useful when building a graph with nodes and edges identified by unique labels. This would lead to duplicates of nodes and edges, if there are multiple occurrences of the same phrase in different grammatical variation. To circumvent this issue, the exporter also trims the words by using their lemma or removing articles on the basis of their part-of-speech tags.

4 Early Validation

We applied the pipeline from Section 3 to two requirements documents. Both have an automotive background and contain real industrial data but they differ in size and do not relate to each other. For this reason, we applied the pipeline to each of them separately. The first one is the system requirements specification *Automotive System Cluster (ASC)*[8], which describes two subsystems, ELC (exterior light system) and ACC (adaptive cruise control). The document is a public case study, which is derived from real requirements but largely reduced and modified by the original authors in size and detail. The other document is provided by our industry partner, describing a single subsystem called "Charging of electric vehicles". Table 1 contains the central measures about the two requirements documents and the knowledge representation graphs that we extracted for both documents.

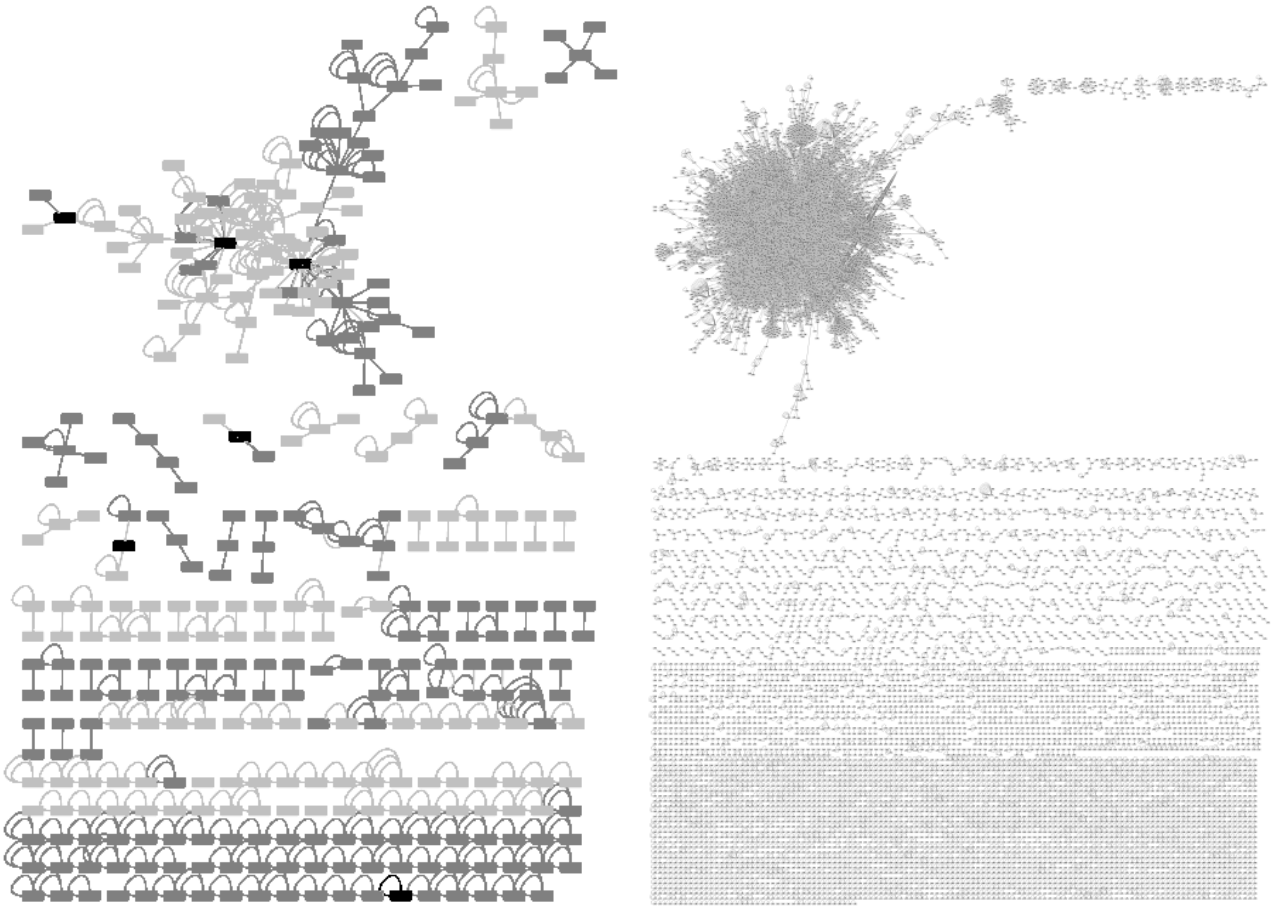
The graphs for both requirements documents are shown in Figure 2. The graph from Figure 2a has different colors representing from which of the two contained systems the nodes and edges originate. Nodes and edges

[4] https://verbs.colorado.edu/~mpalmer/projects/ace.html
[5] https://www.cs.upc.edu/~srlconll/st04/st04.html
[6] https://www.lsi.upc.edu/~srlconll/st05/st05.html
[7] https://ronan.collobert.com/senna/
[8] https://www.aset.tu-berlin.de/fileadmin/fg331/Docs/ASC-EN.pdf

Table 1: Study objects

	ASC	**Charging System**
requirements	153	7,238
authors	4	>66
sentences	153	9,213
tokens (words and marks)	4,140	186,478
nodes [main graph]	345 [149]	13,676 [6,702]
edges [main graph]	440 [228]	19,769 [12,715]

extracted from ELC are colored grey, ACC nodes and edges are light grey. If they are contained in both systems, they are colored black.



(a) Automotive System Cluster (ASC)

(b) Charging System

Figure 2: Knowledge representation graphs extracted from two requirements documents

Both graphs in Figure 2 show some similarities as both contain a "main graph" in the upper left with a lot of nodes and edges linked to each other, several "sub graphs" around the "main graph" with some nodes, and edges a lot of "single graphs" with just two or three edges on the lower part of the figures. The number of nodes and edges for each graph is stated in Table 1, as well as the sum of the "main graph" and the "sub graphs" as [main graph].

While most nodes and edges represent concepts and relations that are only contained in one system, there are some black nodes. Two of them in the middle of the main graph are "vehicle" and "driver". They are central because they are typically stated in almost every requirements document of automotive software. The other black nodes represent "ASIL-B", which stands for an automotive safety integrity level, and "function" respectively "user function", which exists in both systems as a common phrase.

These results cannot be interpreted correctly in detail due to a lack of corresponding quality criteria and currently non-existing use cases. Of course, the pipeline is just a prototype which not able to extract every information correctly nor to build a valid ontology or RDF graph. Nevertheless, it is observable from Figure 2a, that the two systems have no explicit described, conjointly interfaces. This observation can also be found in the original text document.

The sub graphs and single graphs contain information without any connection to the main graph, which indicates, that this connection is not explicitly stated in the requirements document. One reason might be, that this is knowledge the author considers to be common and therefore not documented.

5 Open Challenges

As stated in Section 4, our approach is not able to capture every bit of information contained in the original requirements documents. The major reason for this is the loss that you have to consider whenever you are using NLP techniques on real data. Each step in the pipeline is prone to some errors (missed or incorrectly identified information). Each requirements document has its own structure and is often extended with additional metadata. The pipeline does not consider these characteristics of a document as it interprets each sentence independently step by step. One of the challenges that we want to consider for the future is to integrate structural properties of the document into the knowledge graph (e.g., sentences within one requirement, requirements arranged under one headline).

Causal Relations: Requirements, especially functional ones, often describe causal relationships like "If [A] then [B]". If [A] contains a subject, predicate, and object, SRL annotates those accordingly and annotates the words in [B] as a single adjunct argument like general-purpose or cause. Currently, we do not consider such causal relations as the graph is build only based on the first two arguments of the SRL. Integrating such causal relations the current graph model is not possible since [A] is already a graph which cannot be related (connected via an edge) to another [B] which is a graph itself. Causal relations that we extract from the requirements may be represented as inference rules in RDF schema.

Semantic Similarity: Another problem we observed in the data is, that some nodes and edges seem redundant because different words or phrases are used with the same meaning. Some of them are just wording issues like "speed set point of the cruise control" and "cruise control speed set point" in ASC, but there are also other difficulties like "vehicle" and "car". A major challenge for our approach is to identify such semantic similarities to keep the knowledge graph free of redundancies. The according research field dealing with these kind of challenges is called semantic similarity. Determine a semantic similarity between words or phrases would enable us to identify edges which are identically in the meaning of their concept. These edges could be connected to each other or merged into a single one and would result in a more connected "main graph" and less "sub graphs" or "single graphs".

There are already existing approaches to identify the semantic similarity. A very promising solution is "Align, Disambiguate and Walk: A Unified Approach for Measuring Semantic Similarity" [PJN13]. It is applicable at multiple lexical levels, from single words up to complete texts, comparing each word within its semantic signature. A disadvantage to our use case is the dependency on WordNet as we want to analyse requirements specifications from the automotive context which is not part of WordNet in detail.

Another opportunity are word embeddings like *word2vec* [MCCD13] or *GloVe* [PSM14]. They calculate one-dimensional vectors to represent words. This enables a comparison to the similarity of these vectors. One downside of them is, that the calculation requires a lot of data to build appropriate vectors. Also, these word embeddings rely on the distributional hypothesis, the assumption that semantically similar words appear in similar contexts, which is not always the case [MST+16].

No Ontology: As stated before, the graph does not meet the RDF specification nor is a valid ontology. The nodes and edges are not classified or categorized, e.g., into *literal* or *property* in RDF graph, nor does the current pipeline examine, whether a node or edge is part of the schema or must be part of the instance level. To leverage the full potential of knowledge reasoning, we need to enhance the data with such additional metadata. Depending on the application of this pipeline, this could be done with a predefined schema where the data should fit in or those information could be extracted from the graph itself.

6 Potential Applications

Once we have a good representation of the knowledge contained in a number of requirements documents, we see a number of potential applications.

Knowledge querying: In semantic networks, a requirements engineer can efficiently search for specific information that is contained in any requirements document. In contrast to simple full-text search, a user can express more complex and semantically enriched queries such as "Return all subsystems that interact with my subsystem". Besides querying the semantic net for information that is already encoded in the network, there are also algorithms that can be used to query for associated knowledge (e.g., spreading activation [Cre97]). This allows the user to state queries such as "Return a list of the most relevant stakeholders given a certain feature". If these advanced querying features are integrated in an interactive tool, requirements reviewers may greatly profit from that.

Trace link generation: There is already a number of papers that propose using information retrieval techniques for trace link recovery [HDO03]. In such approaches, single requirements are formulated as queries to find related requirements in a large set of requirements documents. The resulting requirements are considered trace link candidates. With our approach, we may improve these approaches by leveraging the structure of the knowledge graph.

Integration of other NL sources: Requirements documents are not the only natural language documents that contain knowledge about the system to be developed. We plan to integrate knowledge from other natural language artifacts such as system test cases or safety analysis documents into the knowledge representation graph as well.

Detection of implicit knowledge: Implicit knowledge is a potential source of misunderstandings and errors. When requirements authors assume that certain knowledge is obvious or clear to the reader, they do not document this knowledge explicitly. However, sometimes, this assumption is not true. Knowledge representation graphs can indicate implicit knowledge by small subgraphs with concepts that are not related to other concepts. This indicates that the authors presume that additional relations of these concepts are clear to the reader. Another possibility to detect implicit knowledge is to compare the total number of relations that a concept has over all documents with the number of relations that this concept has in one document. Concepts with many relations are "rich" concepts in the sense that authors provide much information about these concepts in their documents. If there is only little information about such "rich" concepts provided in one specific document (i.e., only a small number of relations originate from one document), this may indicate that the author assumed much implicit knowledge.

7 Conclusions

In this paper, we have proposed an NLP pipeline for extracting knowledge from requirements documents and expressing this knowledge as a graph. We use Semantic Role Labeling within the pipeline to extract RDF triples from requirements sentences. As a first validation, we have applied the NLP pipeline to two requirements documents, one academic example with 150 requirements and one industrial document with over 7,000 requirements. The validation shows that both resulting knowledge graph consists of one main subgraph that contains and connects between 40% and 50% of all nodes and several smaller subgraphs with less than 10 connected nodes. In the future, we plan to enhance the approach by identifying semantic similarities between concepts and integrating causal relations expressed in requirements to enable reasoning about the knowledge. We have identified a number of potential applications, where such a knowledge representation may support RE tasks.

References

[BGM85] Alexander Borgida, Sol Greenspan, and John Mylopoulos. Knowledge representation as the basis for requirements specifications. *Computer*, 18(4), 1985.

[BM15] Abel Browarnik and Oded Maimon. Ontology learning from text: Departing the ontology layer cake. *International Journal of Signs and Semiotic Systems*, 4(2), 2015.

[CM04] Xavier Carreras and Lluís Màrques. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Computational Natural Language Learning (CoNLL)*, 2004.

[CM05] Xavier Carreras and Lluís Màrquez. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Computational Natural Language Learning (CoNLL)*, 2005.

[Cre97] Fabio Crestani. Application of spreading activation techniques in information retrieval. *Artificial Intelligence Review*, 11(6), 1997.

[CWB+11] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Computing Research Repository (CoRR)*, abs/1103.0398, 2011.

[DVB+16] Diego Dermeval, Jéssyka Vilela, Ig Ibert Bittencourt, Jaelson Castro, Seiji Isotani, Patrick Brito, and Alan Silva. Applications of ontologies in requirements engineering: A systematic review of the literature. *Requirements Engineering*, 21(4), 2016.

[FTGD15] Nicholas FitzGerald, Oscar Täckström, Kuzman Ganchev, and Dipanjan Das. Semantic role labeling with neural network factors. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[GCW+16] Jiang Guo, Wanxiang Che, Haifeng Wang, Ting Liu, and Jun Xu. A unified architecture for semantic role labeling and relation classification. *International Conference on Computational Linguistics (COLING)*, 2016.

[HDO03] Jane Huffman Hayes, Alex Dekhtyar, and Jeffrey Osborne. Improving requirements tracing via information retrieval. In *11th IEEE International Requirements Engineering Conference (RE)*, 2003.

[HK16] Kerstin Hartig and Thomas Karbe. Recommendation-based decision support for hazard analysis and risk assessment. In *Eighth International Conference on Information, Process, and Knowledge Management (eKNOW)*, 2016.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Computing Research Repository (CoRR)*, abs/1301.3781, 2013.

[MSB+14] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, 2014.

[MST+16] Nikola Mrksic, Diarmuid Ó. Séaghdha, Blaise Thomson, Milica Gasic, Lina Maria Rojas-Barahona, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. Counter-fitting word vectors to linguistic constraints. *Computing Research Repository (CoRR)*, abs/1603.00892, 2016.

[PJN13] Mohammad Taher Pilevar, David Jurgens, and Roberto Navigli. Align, disambiguate and walk: A unified approach for measuring semantic similarity. In *51st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2013.

[PP14] Gerhard Paass and Bhanu Pratap. Semantic role labeling using deep neural networks. In *International Workshop on Representation Learning (RL)*, 2014.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global vectors for word representation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[VF13] Andreas Vogelsang and Steffen Fuhrmann. Why feature dependencies challenge the requirements engineering of automotive systems: An empirical study. In *21st IEEE International Requirements Engineering Conference (RE)*, 2013.

[VFN13] Paola Velardi, Stefano Faralli, and Roberto Navigli. OntoLearn reloaded: A graph-based algorithm for taxonomy induction. *Computational Linguistics (CompLing)*, 39(3), 2013.