Using Domain-specific Corpora for Improved Handling of Ambiguity in Requirements

Saad Ezzini*§, Sallam Abualhaija*§, Chetan Arora‡*, Mehrdad Sabetzadeh†*, Lionel C. Briand*†
*SnT Centre for Security, Reliability and Trust, University of Luxembourg, Luxembourg
‡Deakin University, Geelong, Australia
†School of Electrical Engineering and Computer Science, University of Ottawa, Canada
Email: {saad.ezzini, sallam.abualhaija}@uni.lu, chetan.arora@deakin.edu.au, {m.sabetzadeh, lbriand}@uottawa.ca

***Abstract*—Ambiguity in natural-language requirements is a pervasive issue that has been studied by the requirements engineering community for more than two decades. A fully manual approach for addressing ambiguity in requirements is tedious and time-consuming, and may further overlook *unacknowledged ambiguity* – the situation where different stakeholders perceive a requirement as unambiguous but, in reality, interpret the requirement differently. In this paper, we propose an automated approach that uses natural language processing for handling ambiguity in requirements. Our approach is based on the automatic generation of a domain-specific corpus from Wikipedia. Integrating domain knowledge, as we show in our evaluation, leads to a significant positive improvement in the accuracy of ambiguity detection and interpretation. We scope our work to coordination ambiguity (CA) and prepositional-phrase attachment ambiguity (PAA) because of the prevalence of these types of ambiguity in natural-language requirements [1]. We evaluate our approach on 20 industrial requirements documents. These documents collectively contain more than 5000 requirements from seven distinct application domains. Over this dataset, our approach detects CA and PAA with an average precision of ≈80% and an average recall of ≈89% (≈90% for cases of unacknowledged ambiguity). The automatic interpretations that our approach yields have an average accuracy of ≈85%. Compared to baselines that use generic corpora, our approach, which uses domain-specific corpora, has ≈33% better accuracy in ambiguity detection and ≈16% better accuracy in interpretation.**

***Index Terms*—Requirements Engineering, Natural-language Requirements, Ambiguity, Natural Language Processing, Corpus Generation, Wikipedia.**

I. Introduction

Natural language (NL) is the de-facto medium for specifying requirements in industrial settings. A key advantage of NL is that it facilitates shared understanding among different stakeholders who may have different backgrounds and expertise [2]. Despite this advantage, NL requirements are prone to a variety of quality issues, one of the most notable of which is ambiguity [2], [3]. Ambiguity is an inherent phenomenon in NL, occurring when a text segment is open to multiple interpretations [4]. Ambiguity in requirements can lead to misunderstandings and inconsistencies among the stakeholders, and this can have a potential negative impact on the overall success of a project [5].

Handling ambiguity in requirements is challenging due to two main reasons. First, requirements specifications vary across domains and thus use a domain-specific vocabulary [6]. This has an impact on the likely interpretations of the requirements and consequently on what can be considered as ambiguous. Second, ambiguity can be *unacknowledged*, meaning that multiple readers, being unaware of such ambiguity, may have different interpretations for the same requirement. In contrast to *acknowledged* ambiguity where the reader recognizes ambiguity, unacknowledged ambiguity might lead to serious problems due to unconscious misunderstandings [7]. A fully manual analysis of ambiguity is expensive and also likely to overlook unacknowledged ambiguity. There is therefore a need for effective, automated ambiguity-handling approaches that can help companies focus their often limited quality-assurance budget on requirements that are more likely to be problematic.

Ambiguity has been widely studied in the requirements engineering (RE) literature [3], [8]–[11]. Both manual approaches based on reviews and inspections [9], [12], and automated approaches based on natural language processing (NLP) [6], [7], [13], [14], have been proposed for detecting ambiguity in requirements. Some recent works use domain-specific corpora for detecting terms that are likely to be ambiguous due to different meanings across domains [6], [15]–[17]. Current research on ambiguity in RE, as we elaborate later, has three main limitations. First, the research focuses exclusively on detecting ambiguity and does not address automated interpretation for requirements in which no genuine ambiguity exists. The lack of automated interpretation impedes further automated analysis, e.g., automated information extraction from requirements [18], [19]. Second, existing methods for detecting domain-specific ambiguity are restricted to identifying merely words with different meanings across domains, and further require the domain of interest to be specified a priori. Finally, while the negative consequences of unacknowledged ambiguity are known in the RE literature [7], the question of how accurately unacknowledged ambiguity can be detected through automated means has never been investigated empirically.

Motivated by addressing the above limitations, we propose an automated approach for improved ambiguity handling – both ambiguity detection and interpretation – in NL requirements. Ambiguity detection is concerned with finding the requirements that are genuinely ambiguous. Interpretation, in contrast, is concerned with providing the most likely meaning where the potential for ambiguity exists, but where there is no ambiguity. Our approach incorporates domain knowledge

§Joint First Authors

by automatically generating domain-specific corpora, without any a-priori assumption about the domain. These corpora alongside a set of structural patterns and heuristics are used for handling ambiguity in requirements. In our evaluation, we analyze the impact of domain knowledge on ambiguity handling. We further assess how well our automated approach can detect unacknowledged ambiguity in different domains.

Our work in this paper concentrates on *coordination ambiguity* and *prepositional-phrase attachment ambiguity* [3], [20], [21], hereafter referred to as CA and PAA, respectively. Targeting these (syntactic) ambiguity types is motivated by their prevalence in NL requirements [1]. In our document collection, as we will discuss later in the paper, out of 5156 requirements, 1098 (21%) are subject to CA analysis and 1328 (26%) to PAA analysis. Within these, human annotators acknowledged ambiguity or had different interpretations (unacknowledged ambiguity) in ≈57% of the requirements.

Coordination is a structure that links together two sentence elements (called conjuncts) using a coordinating conjunction (e.g., "and" or "or") [22]. CA can potentially occur when the two conjuncts are preceded or followed by a modifier [21]. The sentence could then be interpretable in two ways, depending on whether only the conjunct next to the modifier is being modified or both conjuncts are being modified [3]. Fig. 1 shows an example requirement, R1, with two potential interpretations. The first interpretation, *first read*, hereafter, *FR*, occurs when the modifier "LEO" (low-earth orbit) modifies the two conjuncts "satellites" and "terminals" (Fig. 1 (a)). The second interpretation, *second read*, hereafter, *SR*, occurs when the modifier "LEO" modifies "satellites" only (Fig. 1 (b)).

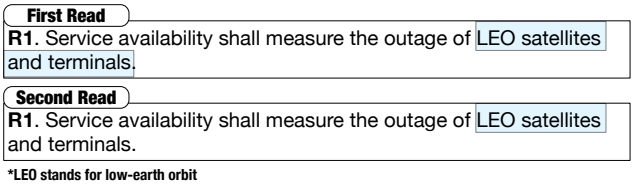

Fig. 1. Example of Coordination Ambiguity (CA).

A prepositional-phrase (PP) attachment is a PP preceded by a verb and a noun phrase [20]. Virtually all PP attachments have the potential for PAA, because they could be interpretable in two ways, depending on whether the PP is an adverbial modifier (attached to the preceding verb) or a noun attribute (attached to the preceding noun phrase). Fig. 2 shows an example requirement, R2, with two potential interpretations due to the presence of a PP attachment. The first interpretation, *verb attachment*, hereafter, *VA*, occurs when the PP "with discrete tags" is attached to the verb "categorize" (Fig. 2 (a)). The second interpretation, *noun attachment*, hereafter, *NA*, occurs when the PP is attached to the noun "outages" (Fig. 2 (b)).

R1 and R2 have the potential to suffer from CA and PAA, respectively. The question is whether these are genuine ambiguities or merely situations that human experts can decisively interpret with little room for divergent interpretations. Existing techniques do not incorporate domain knowledge for providing a likely interpretation of CA; instead, they rely on frequency-based computations derived from a generic corpus [23]. For example, using existing techniques, attempting to interpret the coordination in R1 would yield *FR*. This interpretation is incorrect; with domain knowledge, the coordination would be interpreted as *SR*. As for the PP attachment in R2, existing techniques are unable to provide an interpretation, although the attachment is interpretable as *VA* with domain knowledge.

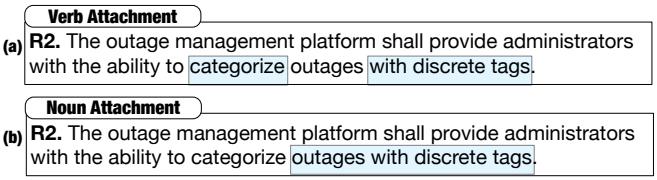

Fig. 2. Example of Prepositional-phrase Attachment Ambiguity (PAA).

Contributions. We take steps toward addressing the limitations outlined above. Our contributions are as follows:

(1) We propose an automated approach for handling CA and PAA in NL requirements. Our approach uses an ensemble of structural patterns and heuristics. Specifically, we match requirements against a set of structural patterns, leveraging and enhancing existing patterns in the literature. In tandem, we attempt to interpret all requirements with coordination and PP-attachment structures using heuristics that are based on semantic, morphological and frequency information. Some of these heuristics are novel; others are borrowed from the literature and enhanced where necessary. By combining these patterns and heuristics, we attempt to tell apart the requirements that can be disambiguated via automated interpretation from the requirements that are genuinely ambiguous.

(2) We devise a novel domain-specific corpus generator. Without assuming any a-priori knowledge about the domain, we first automatically extract keywords from an input requirements document. Our corpus generator then assembles a large corpus of Wikipedia articles relevant to the terminology (and thus the domain) of the given requirements document. This automatically generated corpus is utilized for increasing the accuracy of the heuristics that rely on frequency-based information. For example, the occurrence of the word "capital" in a requirements document within the aerospace domain differs in frequency and co-occurring words from the same word occurring in a requirements document within the financial domain. Generating and using a domain-specific corpus for ambiguity handling lies at the heart of our proposed approach.

(3) We empirically evaluate our approach on 20 industrial requirements documents. These documents collectively contain 5156 requirements covering seven distinct application domains. The ground truth for our evaluation was prepared by two trained annotators (linguistics experts and non-authors). Our results indicate that: (i) our approach detects CA and PAA with a precision of ≈80% and recall of ≈89% (≈90% for cases of unacknowledged ambiguity); (ii) the automatic interpretations by our approach have an average accuracy of ≈85%; and (iii) using domain-specific corpora leads to substantial gains in accuracy for ambiguity handling, improving detection by an average of ≈33% and interpretation by an average of ≈16%. We have developed a tool, named

MAANA, which implements our approach for the domain-specific handling of ambiguity. Specifically, MAANA detects requirements that potentially contain CA or PAA. The tool and the non-proprietary requirements we use in our evaluation are publicly available at https://github.com/SNTSVV/MAANA.

Structure. Section II discusses and compares with related work. Section III presents our approach. Section IV describes our empirical evaluation. Section V addresses validity considerations. Section VI concludes the paper.

II. Related Work

We focus on handling CA and PAA in NL requirements. Our approach, discussed in Section III, builds on and further enhances the existing structural patterns and heuristics from the RE and NLP literature for CA [23]–[33] and PAA [3], [34], [35]. Our work, to our knowledge, is the first to bring these patterns and heuristics together for handling CA and PAA. Below, we position our work against the related work on ambiguity handling in both the RE and NLP communities.

A. Ambiguity Handling in the RE Community

Ambiguity in requirements has been extensively studied from different perspectives, including understanding the role of ambiguity in RE [8], [36]–[38], analyzing the linguistic causes of ambiguity [3], [39]–[41], and ambiguity prevention [42]–[46]. Automated ambiguity detection solutions in RE are mainly based on matching NL requirements against pre-defined structural patterns using regular expressions, NLP, or both [6]. Numerous approaches and tools have been proposed to this end [10], [13], [30], [45], [47]–[52]. In addition to these, some recent works attempt to detect lexical ambiguity – the situation where a word has different meanings depending on the domain [40] – by integrating domain knowledge from Wikipedia [6], [15]–[17].

CA detection has been investigated to some extent in the RE literature. Chantree et al. [23] address CA detection using structural patterns and frequency-based heuristics. Their work has been extended over the years [27], [29], [53] with additional heuristics [25], [31], and for anaphora ambiguity detection [13], i.e., ambiguity due to multiple interpretations of pronouns. Though considered a prevalent ambiguity type in requirements [3], [8], [40], to our knowledge, automated handling of PAA has not been previously studied in RE.

Our work differs from or enhances the above research in several ways. First, none of the existing approaches address the automated interpretation of (potentially ambiguous) coordination structures. As for PAA, the topic has not been tackled in RE before. Our approach handles both CA and PAA by combining a broad range of structural patterns and heuristics. Second, none of the existing approaches evaluate the detection of unacknowledged ambiguity. We address this gap in our empirical evaluation. Third, the existing automated methods for domain-specific corpus generation from Wikipedia are limited to a pre-defined set of domains. Our approach, in contrast, can generate a corpus based on any given requirements document without knowing the underlying domain in advance. Fourth and finally, industrial evaluations of ambiguity handling in RE are scarce. Our evaluation contributes to addressing this gap by using a large industrial dataset.

B. Ambiguity Handling in the NLP Community

Syntactic ambiguity types, including CA and PAA, have been studied for a long time by the NLP community [54]. In an early work by Goldberg [24], CA is handled using conditional probabilities of word co-occurrences. Pantel and Lin [55] present an unsupervised corpus-based method for handling PAA through a notion of contextual similarity. Agirre et al. [34] improve the accuracy of PAA handling by integrating semantic similarity with syntactic parsing. Calvo and Gelbukh [35] propose querying the web for word co-occurrence frequencies and use these frequencies for more accurate PPA handling. In a similar vein, Nakov and Hearst [26] use structural patterns alongside statistical co-occurrence frequencies gathered from the web for handling CA and PAA.

In the context of ambiguity handling, the use of domain knowledge in NLP is mostly directed at word sense disambiguation (WSD) in specific domains [56]. To this end, Wikipedia is a commonly used source of domain knowledge [57], [58]. Fragolli [59] derives from Wikipedia domain-specific corpora as resources for WSD. Similarly, Gella et al. [60] map manually defined topics in WordNet [61], [62] to Wikipedia for generating domain-specific corpora that can in turn be employed for WSD.

We are not aware of any work in the NLP community that uses domain-specific corpora for handling either CA or PAA. Instead, in the existing NLP technologies, e.g., syntax parsing [63], the handling of syntactic ambiguity – CA and PAA included – is tuned over generic texts such as news articles. These technologies therefore do not provide accurate results for CA and PAA in a domain-specific context. As we show in Section IV, our approach, which incorporates domain knowledge for handling CA and PAA, provides significant improvements over NLP technologies tuned over generic texts.

III. Approach

Fig. 3 provides an overview of our approach, which is composed of five steps. The input to the approach is an NL requirements document, hereafter, ***RD***. In step 1, we process ***RD*** using an NLP pipeline. In this step, we further identify two subsets of the requirements in ***RD***, namely S_c and S_p. These two subsets contain all the requirements with coordination structures and all the requirements with PP attachments, respectively. In step 2, we match the requirements in S_c and S_p against structural patterns that indicate potential CA and PAA, respectively. In step 3, we generate a domain-specific corpus for ***RD*** by crawling Wikipedia. Step 3 can be bypassed if a representative corpus for ***RD***'s domain already exists (through earlier applications of our approach to other requirements documents in the same domain). In step 4, we apply a set of heuristics to determine likely interpretations for the requirements in S_c and S_p. In step 5, we classify into ambiguous and unambiguous the requirements in S_c and S_p

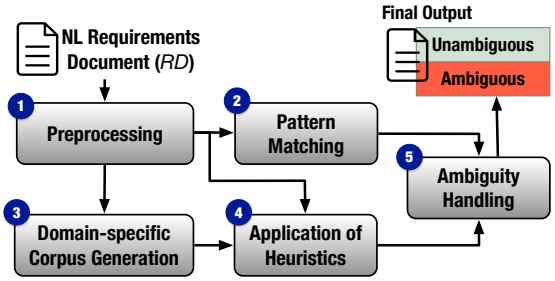

Fig. 3. Approach Overview.

by combining the results of step 2 and step 4. We note that steps 2 and 4 are independent (i.e., the output of neither step is an input to the other). Step 2 is limited to a finite list of CA and PAA structural patterns. As we will explain later in this section, the heuristics in step 4, when compared to the patterns in step 2, cover a wider spectrum of structures that have the potential for CA and PAA. Combining results from both steps leads to better handling of ambiguity. Below, we elaborate each step of our approach. In the rest of this paper, ambiguity refers to CA and PAA exclusively.

A. Preprocessing

The NLP pipeline we use for preprocessing *RD* is depicted in Fig. 4. This pipeline is a sequence of five NLP modules. The first module in the sequence, the *Tokenizer*, divides the input text into tokens, such as words and punctuation marks. The *Sentence Splitter* splits the text into sentences. The *POS Tagger* assigns to tokens part-of-speech (POS) tags, such as noun, verb and adjective. Next is the *Lemmatizer*, which identifies the canonical form (lemma) for each token. For example, the lemma for "bought" is "buy". Finally, the *Constituency Parser* identifies the structural units of sentences, e.g., noun phrases, verb phrases and prepositional phrases. The results of the NLP pipeline are used in the next steps.

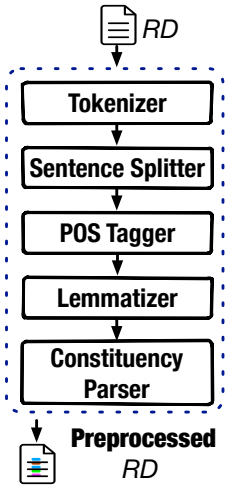

Fig. 4. NLP Pipeline.

In this step (step 1), we further identify the two requirements subsets, S_c and S_p, that should be subject to CA handling and PAA handling, respectively. S_c is comprised of all the requirements in *RD* that contain "or", "and", or both. We note that only these two conjunctions can lead to CA [21], [23]. S_p is comprised of all the requirements in *RD* that contain a PP attachment [20]. Requirements that contain a conjunction of interest (i.e., "and" or "or") as well as a PP attachment are included in both S_c and S_p.

B. Pattern Matching

In this step, we analyze S_c and S_p to identify requirements that are likely to be ambiguous due to their syntactic structure. Table I lists our patterns for CA and PAA. Of these, 23 CA patterns and four PAA patterns come from the literature [3], [24]–[27], [29], [30], [34]. The remaining patterns, shaded blue in the table (i.e., CA patterns #24–29 and PAA patterns #5–10) are novel. The novel patterns were derived by analyzing a subset of the requirements in our dataset, as we will precisely define in Section IV-C. Specifically, we analyzed the ambiguous requirements in the *tuning* portion of our dataset.

TABLE I
PATTERNS FOR AMBIGUITY DETECTION (CA AND PAA).

CA	1	nn $\mathbf{n_1}$ *c* $\mathbf{n_2}$	11	$\mathbf{n_1}$ *c* $\mathbf{n_2}$ *p* dt/adj nn	21	adv $\mathbf{adj_1}$ *c* $\mathbf{adj_2}$
	2	$\mathbf{n_1}$ *c* $\mathbf{n_2}$ nn	12	$\mathbf{v_1}$ *c* $\mathbf{v_2}$ nn	22	$\mathbf{adj_1}$ *c* $\mathbf{adj_2}$ adv
	3	nn *p* $\mathbf{n_1}$ *c* $\mathbf{n_2}$	13	nn *p* $\mathbf{v_1}$ *c* $\mathbf{v_2}$	23	$\mathbf{adj_1}$ *c* $\mathbf{adj_2}$ adj nn
	4	$\mathbf{n_1}$ *c* $\mathbf{n_2}$ *p* nn	14	$\mathbf{v_1}$ *c* $\mathbf{v_2}$ *p* nn	24	nn *dt* $\mathbf{n_1}$ *c dt* $\mathbf{n_2}$
	5	v $\mathbf{n_1}$ *c* $\mathbf{n_2}$	15	v to $\mathbf{v_1}$ *c* $\mathbf{v_2}$	25	nn *p dt* $\mathbf{n_1}$ *c dt* $\mathbf{n_2}$
	6	$\mathbf{n_1}$ *c* $\mathbf{n_2}$ v	16	$\mathbf{v_1}$ *c* $\mathbf{v_2}$ to v	26	*dt* $\mathbf{n_1}$ *c dt* $\mathbf{n_2}$ *p* nn
	7	nn $\mathbf{n_1}$ *c* $\mathbf{n_2}$ nn	17	adv $\mathbf{v_1}$ *c* $\mathbf{v_2}$	27	nn *dt* $\mathbf{n_1}$ *c dt* $\mathbf{n_2}$ nn
	8	adj $\mathbf{n_1}$ *c* $\mathbf{n_2}$	18	$\mathbf{v_1}$ *c* $\mathbf{v_2}$ adv	28	adj *dt* $\mathbf{n_1}$ *c dt* $\mathbf{n_2}$
	9	adj nn $\mathbf{n_1}$ *c* $\mathbf{n_2}$	19	$\mathbf{v_1}$ *c* $\mathbf{v_2}$ *p* dt/adj nn	29	*adj* nn *dt* $\mathbf{n_1}$ *c dt* $\mathbf{n_2}$
	10	adj adj $\mathbf{n_1}$ *c* $\mathbf{n_2}$	20	dt/adj nn *p* $\mathbf{v_1}$ *c* $\mathbf{v_2}$		

PAA	1	**v** $\mathbf{n_1}$ *p* $\mathbf{n_2}$	6	**v** $\mathbf{n_1}$ *p dt adj* $\mathbf{n_2}$
	2	**v** $\mathbf{n_1}$ *p dt/adj* $\mathbf{n_2}$	7	**v** *n* $\mathbf{n_1}$ *p dt/adj* $\mathbf{n_2}$
	3	**v** *dt/adj* $\mathbf{n_1}$ *p* $\mathbf{n_2}$	8	**v** *n* $\mathbf{n_1}$ *p dt adj* $\mathbf{n_2}$
	4	**v** *dt/adj* $\mathbf{n_1}$ *p dt/adj* $\mathbf{n_2}$	9	**v** *dt adj* $\mathbf{n_1}$ *p* $\mathbf{n_2}$
	5	**v** *n* $\mathbf{n_1}$ *p* $\mathbf{n_2}$	10	**v** *dt adj* $\mathbf{n_1}$ *p dt/adj* $\mathbf{n_2}$

n_1, n_2, nn: noun, **v**: verb, **adv**: adverb, **adj**: adjective, **dt**: determiner, **p**: preposition, **/**: or.

For CA: **The two conjuncts are in bold** and the modifier is underlined.
For PAA: **The verb and first noun are in bold**, and the second noun is underlined.

We match the patterns against the requirements in S_c and S_p. For pattern matching, the unit of analysis is a text *segment*, which is the part of a requirement that matches a given structural pattern from Table I. A pattern suggesting CA matches a segment that contains a conjunction (denoted as *c*) linking two conjuncts (marked in bold) with a modifier (underlined). For example, the matching segment in R1 (Fig. 1) corresponds to pattern#1 for CA, where LEO is the modifier and the conjunction *and* joins the two conjuncts **satellites** and **terminals**. We recall from Section I that CA occurs when it is unclear whether a modifier is attached to both conjuncts (*FR*) or only to the closest conjunct (*SR*). A pattern suggesting PAA matches a segment with a verb (**v**) followed by a first noun ($\mathbf{n_1}$) – both marked in bold – followed by a PP which consists of a preposition (denoted as *p*) and a second noun (n_2 – underlined). For example, the matching segment in R2 (Fig. 2), "**categorize outages** *with* discrete tags", corresponds to pattern#2 for PAA. Again, we recall from Section I that PAA occurs when it is unclear whether the PP in question is an adverbial modifier attached to **v** (*VA*) or a noun attribute attached to $\mathbf{n_1}$ (*NA*).

Step 2 identifies the segments (from the requirements in S_c and S_p) that match any of the patterns in Table I. The matched segments are passed on to step 5.

C. Domain-specific Corpus Generation

This step attempts to capture the domain knowledge underlying the input requirements document (*RD*) by crawling Wikipedia. Fig. 5 shows the sub-steps for generating a domain-specific corpus. We elaborate these (sub-)steps next.

Extract Keywords. Step 3.1 builds on an existing automated requirements glossary extraction approach by Arora et al. [64].

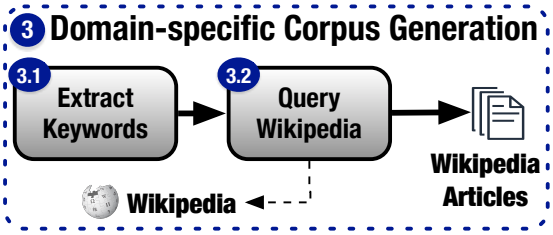

Fig. 5. Domain-specific Corpus Generation.

We begin by (automatically) extracting the list of glossary terms from *RD*, and thereafter select the top-K most frequent keywords from the list. The optimal value of K is tuned in Section IV-D. For example, the keywords extracted from R1 (Fig. 1) include "LEO", "LEO satellites", "satellites", and "terminals". These keywords are used in the next step.

Query Wikipedia. Step 3.2 implements a query engine for identifying Wikipedia articles that are relevant to the keywords resulting from step 3.1. These articles form the basis of our domain-specific corpus. We begin by retrieving matching Wikipedia articles for each keyword. An article is considered a match if the keyword in question contains (or is contained in) the title of the Wikipedia article. For instance, the Wikipedia article titled "satellite navigation" is a match for the keyword "satellite-based navigation". If the above condition is not met, no matching Wikipedia article is retrieved.

Next, we broaden the domain information captured in our corpus by taking advantage of the hierarchical category structure of Wikipedia [57]. In Wikipedia's hierarchy, each category can contain articles and nested sub-categories. For a matching article, we retrieve all the articles in the same category and all the articles in the descendant sub-categories. For example, the "satellite navigation" article, as shown in Fig. 6, is classified under an identically named Wikipedia category (https://en.wikipedia.org/wiki/Category:Satellite_navigation; accessed 17/8/2020). We retrieve all articles in this category and in all its descendants (e.g., one descendant being "Geocaching"). Doing so increases topic coherence [65], meaning that the articles included in the corpus are all indeed relevant to the domain under analysis.

Next, to make our domain-specific corpus applicable to other requirements documents from the same domain, we attempt to increase the coverage of our corpus. In particular, we consider the categories in the Wikipedia category graph that are directly connected to the category of the matching article. For instance, the category "navigation" in Fig. 6 is directly connected to "satellite navigation"; we therefore include articles listed under "navigation" and its descendant sub-categories.

We note that, during the creation of a corpus, we consider only the categories whose number of articles is below a threshold (α); this is both to keep the computation time reasonable and to avoid including large and generic categories in the corpus. We discuss the tuning of α in Section IV-D. The result of this step (step 3.2) is a body of raw text from Wikipedia articles. This extracted body of text is our domain-specific corpus, hereafter denoted as $\mathcal{D}$.

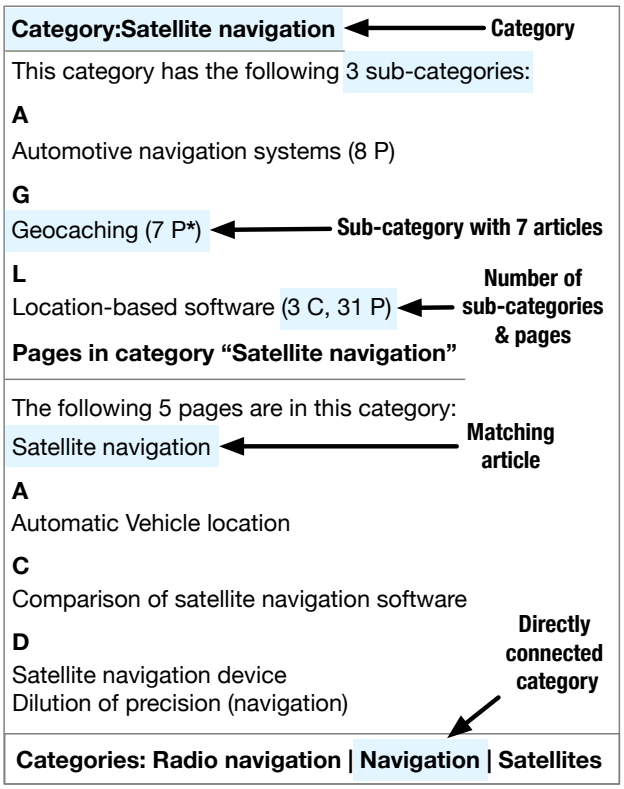

Fig. 6. Example of Category Structure in Wikipedia.

D. Application of Heuristics

Step 4 attempts to provide likely interpretations for the requirements in S_c and S_p. We use six heuristics, denoted as $\mathcal{C}_1$–$\mathcal{C}_6$, for interpreting coordination structures and four heuristics, denoted as $\mathcal{P}_1$–$\mathcal{P}_4$, for interpreting PP-attachment structures. Out of these ten heuristics, eight ($\mathcal{C}_1$–$\mathcal{C}_5$ for CA [23], [31]–[33] and $\mathcal{P}_1$–$\mathcal{P}_3$ for PAA [34], [35]) are borrowed from the literature. The other two ($\mathcal{C}_6$ and $\mathcal{P}_4$) are novel, but based on a very intuitive idea: applying constituency parsing, which has coordination and PP-attachment interpretation built into it.

Similar to step 2 (Section III-B), we operate at a *segment* level. Compared to the patterns in step 2, heuristics cover a wider spectrum of segments that have the potential to be ambiguous. The heuristics are triggered by the presence of any coordination structure (an "and" / "or" conjunction, two conjuncts and a modifier) and any PP-attachment structure (a verb, a noun and a PP). For example, had R2 (Fig. 2) contained an extra adjective "categorize outages with *standard* discrete tags", R2 would not have been detected by the patterns of Table I, but would have been picked up by the heuristics and attempted for interpretation.

Several heuristics in our approach are corpus-based, i.e., require information about the co-occurrence frequencies of the words. We therefore transform the Wikipedia articles from step 3 to an n-grams table with n ranging from 1 to 5. We set the upper limit to 5, motivated by the use of 5-grams in Google's well-known Web1T database [66]; this database is utilized in a wide variety of NLP applications [67]–[69].

Table II shows a (very small) excerpt of a 5-grams table generated for the satellite domain. The frequencies used by the heuristics are the normalized values of the co-occurrence counts listed in the 5-grams table [70]. For example, the co-occurrence frequency of "satellite orbit" is computed as $234/(21013 + 26599) \approx 0.005$.

TABLE II
EXCERPT OF 5-GRAMS TABLE.

	Words	Count
Unigrams	system	360070
	satellite	21013
	navigation	11610
	orbit	26599
	...	
2-grams	navigation system	1284
	satellite orbit	234
	...	
3-grams	satellite navigation system	138
	low earth orbit	724
	...	
4-grams	global navigation satellite system	89
	geosynchronous satellite launch vehicle	27
	...	
5-grams	satellite power system concept development	8
	LEO sun synchronous receiver satellites	8

Heuristics for CA. A segment in S_c contains a conjunction (c), two conjuncts (conjunct$_1$ and conjunct$_2$), and a modifier (mod). CA heuristics attempt to interpret a segment in S_c as either *FR* or *SR*. If a heuristic cannot interpret a segment, it returns a designated value, *not interpretable* (*NI*). As we explain below, four of the CA heuristics ($\mathcal{C}_1$ and $\mathcal{C}_3$–$\mathcal{C}_5$) require pre-defined thresholds, denoted as θ_i. These thresholds come from the existing literature. We tune the thresholds empirically in Section IV-D. To illustrate the heuristics in this section, we already use the tuned θ_i values: $\theta_1 = 0.01$, $\theta_3 = 0.12$, $\theta_4 = 3.45$, and $\theta_5 = 3$.

($\mathcal{C}_1$) *Coordination frequency* computes the co-occurrence frequency of conjunct$_1$ and conjunct$_2$ in our domain-specific corpus ($\mathcal{D}$). We consider the co-occurrence frequency of the conjuncts irrespective of their order. For example, for R1, we consider, among other possible combinations, the co-occurrence frequency of "terminals and satellites" and "satellites or terminals". The intuition is that if the two conjuncts co-occur frequently in $\mathcal{D}$, they can be regarded as one syntactic unit and thus are both modified (by mod), in turn favoring *FR*. $\mathcal{C}_1$ returns *FR* if the resulting frequency is greater than a threshold (θ_1) and *NI* otherwise. In R1, $\mathcal{C}_1$ returns *FR*.

($\mathcal{C}_2$) *Collocation frequency* compares the co-occurrence frequency of conjunct$_1$ and mod against the frequency of conjunct$_2$ and mod. Collocation is a recurrent combination of two consecutive words in a large corpus [71]. For example, the words "red" and "wine" would be considered collocated, while "great" and "wine" would not. The intuition is that a collocation of the mod and the conjunct closer to it is likely to indicate a syntactic unit, thus favoring *SR*. Using collocations, "red wine and cheese" can be interpreted as *SR* while "great wine and cheese" would not be interpretable (*NI*). $\mathcal{C}_2$ returns *SR* if the collocation frequency of mod and the closer conjunct is greater than that of mod and the farther conjunct, and *NI* otherwise. In R1, $\mathcal{C}_2$ returns *SR*.

($\mathcal{C}_3$) *Distributional similarity* measures the contextual similarity of conjunct$_1$ and conjunct$_2$ [72], i.e., how frequently the conjuncts appear in similar contexts. For example, in the context of requirements documents about satellite systems, the terms "satellite" and "navigation" have a higher distributional similarity than "satellite" and "investment". The intuition is that conjuncts with high distributional similarity can be regarded as one unit, thus favoring *FR*. $\mathcal{C}_3$ returns *FR* if the distributional similarity of the conjuncts is greater than θ_3, and *NI* otherwise. In R1, $\mathcal{C}_3$ returns *NI*.

($\mathcal{C}_4$) *Semantic similarity* measures the similarity between conjunct$_1$ and conjunct$_2$ based on their meanings in WordNet. The intuition is that conjuncts with high semantic similarity can be regarded as one unit, thus favoring *FR*. $\mathcal{C}_4$ returns *FR* if the semantic similarity is greater than θ_4, and *NI* otherwise. In R1, $\mathcal{C}_4$ returns *FR*.

($\mathcal{C}_5$) *Suffix matching* examines the number of shared trailing characters (suffixes) of conjunct$_1$ and conjunct$_2$. For example, "installation and configuration" share five trailing characters. Suffixes are used to change the meaning (e.g., "-able" in noticeable) or grammatical property (e.g., "-ed" in closed) of a given word [73]. Hence, matching suffixes provides a cue about how words are semantically or syntactically related [33]. The intuition is that conjuncts with the same number of trailing characters are likely to be a single unit, thus favoring *FR*. $\mathcal{C}_5$ returns *FR* if the conjuncts share trailing characters greater than θ_5, and *NI* otherwise. In R1, $\mathcal{C}_5$ returns *NI*.

($\mathcal{C}_6$) *Coordination syntactic* analysis is based on applying constituency parsing to the requirement in which the (coordination) segment of interest appears and then obtaining (from the parse tree) the interpretation of the parser for the segment. $\mathcal{C}_6$ returns *FR* or *SR* as per the parsing results, and *NI* if the parser fails to parse the requirement. In R1, $\mathcal{C}_6$ returns *FR*.

Heuristics for PAA. A segment in S_p contains a verb (v) and a following noun (n_1), followed by a preposition (p) and another noun (n_2). PAA heuristics attempt to interpret a segment as either *VA* or *NA*, as explained below. If a heuristic cannot interpret a segment, it returns *not interpretable* (*NI*).

($\mathcal{P}_1$) *Preposition co-occurrence frequency* compares the frequency of p occurring with v against p occurring with n_1. The intuition is that, based on the co-occurrence frequency of v (or n_1) and p, PP can be regarded as an adverbial modifier leading to *VA* or a noun attribute leading to *NA*. For example, in the segment "provide user with a valid option", the preposition "with" frequently follows the verb "provide", thus leading to a *VA* interpretation. Precisely, $\mathcal{P}_1$ returns *VA* if the co-occurrence frequency of v and p is strictly larger than that of n_1 and p. $\mathcal{P}_1$ returns *NA* if the converse is true. If there is a tie between the frequencies, e.g., when the frequencies are zero due to v, n_1 or p being absent from the corpus, $\mathcal{P}_1$ returns *NI*. In R2, $\mathcal{P}_1$ returns *NA*.

($\mathcal{P}_2$) *Prepositional-phrase (PP) co-occurrence frequency* has a similar definition and intuition to $\mathcal{P}_1$, the only difference being that we consider the entire PP (i.e., p and n_2) instead of just p. For example, consider the segment "provide [...]" used for illustrating $\mathcal{P}_1$. $\mathcal{P}_2$ would return *VA* because the PP "with a valid option" has a higher co-occurrence frequency with v ("provide") than with n_1 ("user"). $\mathcal{P}_2$'s precise definition is easy to extrapolate from the definition of $\mathcal{P}_1$ and is omitted for space. In R2, $\mathcal{P}_2$ returns *NA*.

($\mathcal{P}_3$) *Semantic-class enrichment* utilizes the semantic classes in WordNet that group words with similar meanings. For example, WordNet puts "scissors" and "knife" under the same semantic class, namely "tool". $\mathcal{P}_3$ is applied after all the segments in S_p have been already processed by $\mathcal{P}_1$ and $\mathcal{P}_2$. Specifically, $\mathcal{P}_3$ attempts to find an interpretation for the segments that have been deemed as *NI* by both $\mathcal{P}_1$ and $\mathcal{P}_2$. For any such segment X, $\mathcal{P}_3$ checks whether there is some segment Y in S_p which has been interpreted as *VA* or *NA* (by either $\mathcal{P}_1$ or $\mathcal{P}_2$) and which shares a semantic class with X. By sharing a semantic class, we mean that X and Y contain nouns or verbs that fall under the same WordNet semantic class. If *Y* has been interpreted as *VA* (respectively, *NA*) and X shares a verb class (respectively, a noun class) with Y, then $\mathcal{P}_3$ interprets X as *VA* (respectively, *NA*).

The intuition is as follows: segments that contain words with similar meanings are likely to have the same interpretation [34]. For instance, a segment *X*: "offer operator with a valid option" is interpreted as *VA* by $\mathcal{P}_3$ if there is a segment *Y*: "provide user with a valid option" already interpreted as *VA* by $\mathcal{P}_2$. This is because the verbs "provide" and "offer" have the same WordNet semantic class: "possession".

($\mathcal{P}_4$) *Attachment syntactic analysis* has the same intuition and definition as $\mathcal{C}_6$, except that it applies to a PP-attachment segment. $\mathcal{P}_4$ returns *VA* or *NA*, as per the parsing results. $\mathcal{P}_4$ returns *NI* if the parser fails. In R2, $\mathcal{P}_2$ returns *VA*.

Combination of Heuristics. To produce a single interpretation for each segment, we combine through voting the results of the heuristics for each ambiguity type ($\mathcal{C}_1$ – $\mathcal{C}_6$ for CA and $\mathcal{P}_1$–$\mathcal{P}_4$ for PAA). We consider two voting methods: *majority voting* and *weighted voting* [74]. In majority voting, all heuristics contribute equally and the resulting interpretation is based on the majority. In weighted voting, the contribution of each heuristic is weighted differently. The weights are tuned in Section IV-D. In R1, majority voting yields *FR*, while weighted voting (using the tuned weights of Section IV-D) yields *SR*. We compare the accuracy of both voting methods in Section IV.

Step 4 partitions S_c and S_p into two subsets each: the first subset contains the *interpretable* segments (*FR* or *SR* for segments in S_c, and *VA* or *NA* for segments in S_p); the second subset contains the segments that are *not interpretable*. These subsets are passed on to step 5 for ambiguity handling.

E. Handling Ambiguity

In this final step, we classify into *ambiguous* and *unambiguous* the segments in S_c and S_p. This classification is based on the results of steps 2 and 4 in our approach (see Fig. 3). A segment X is classified as *ambiguous* if either of the following two conditions is met: (a) X matches some pattern in step 2, or (b) X is deemed as not interpretable (*NI*) in step 4. Any segment that is not classified as ambiguous would be *unambiguous*. We say that a requirement is ambiguous if it has some ambiguous segment; otherwise, we say the requirement is unambiguous. Our empirical evaluation, discussed next, is at a segment level (rather than a requirement level), because each requirement may contain multiple segments that are subject to ambiguity analysis.

IV. Evaluation

In this section, we empirically evaluate our approach.

A. Research Questions (RQs)

Our evaluation addresses four research questions:

RQ1. What configuration of our approach yields the most accurate results for ambiguity handling? Our approach can be configured in a number of alternative ways; the alternatives arise from the choices available for the selection of patterns (Section III-B), the use of default versus optimal thresholds for CA heuristics (Section III-D), and the voting method for combining the heuristics (Section III-D). RQ1 identifies the configuration that produces the best overall results.

RQ2. How effective is our approach at detecting unacknowledged ambiguity? As discussed in Section I, unconscious misunderstandings may occur due to unacknowledged ambiguity. Using the best configuration from RQ1, RQ2 assesses the effectiveness of our approach in automatically detecting unacknowledged ambiguity.

RQ3. How accurate are the interpretations provided by our approach? While the exact interpretation of a segment found by our approach (*FR* or *SR* for segments in S_c, and *VA* or *NA* for segments in S_p) has no bearing on how we tell apart unambiguous cases from ambiguous ones, we want the interpretations to be as correct as possible. A correct interpretation is important both for reducing manual work (in case the analysts choose to vet the automatic interpretations), and also for ensuring that any subsequent automated analysis over the requirements, e.g., automated information extraction, will produce high-quality results. RQ3 examines the accuracy of the interpretations provided by our approach.

RQ4. Does our approach run in practical time? RQ4 studies whether the execution time of our approach is practical.

B. Implementation

We have implemented our approach (Fig. 3) in Java. The implementation has ≈8500 lines of code excluding comments. The NLP pipeline of step 1 is implemented using DKPro [75]. For implementing step 3, we use the English Wikipedia dump[§] timestamped 01/11/2018. We access the data in this dump using the JWPL library [76]. In step 4, we transform the raw text of Wikipedia articles into an n-grams table using the JWEB1T library [77]; this enables us to compute our interpretation heuristics more efficiently. We use Stanford Parser [78] to obtain the parse trees required by heuristics $\mathcal{C}_6$ and $\mathcal{P}_4$. For $\mathcal{C}_4$, we compute semantic similarity using the Resnik measure [79] as implemented by the WS4J library [80]. For implementation availability, please see the footnote on page 2.

C. Data Collection

Our data collection has human experts examine and annotate potential CA and PAA in industrial requirements. We collected

[§]https://dumps.wikimedia.org/backup-index.html

TABLE III
DATA COLLECTION RESULTS.

	Domain	Aerospace	Automative	Defense	Digitalization	Medicine	Satellite	Security	Total
	RDs	5	3	2	3	1	4	2	20
	Total Requirements	1510	284	910	701	189	1420	142	5156
S_c	Requirements	295	49	294	164	28	265	3	1098
	Segments	382	64	416	250	39	352	3	1506
	Acknowledged	78	21	164	36	0	70	1	370
	Unacknowledged	171	15	19	109	19	154	0	487
	Unambiguous	133	28	233	105	20	128	2	649
S_p	Requirements	353	44	238	272	19	333	69	1328
	Segments	388	47	266	312	20	360	81	1474
	Acknowledged	117	8	29	105	5	14	0	278
	Unacknowledged	115	17	131	77	5	174	36	555
	Unambiguous	156	22	106	130	10	172	45	641

our data from 20 requirements documents (*RD*s) written in English covering seven different application domains. Data collection was performed by two third-party annotators (non-authors) with expertise in linguistics. The first annotator, Anna (pseudonym), has a Masters degree in Multilingualism. Anna had previously completed a three-month internship in RE. The second annotator, Nora (pseudonym), has a Masters degree in IT Quality Management. Nora has a professional certificate in English translation. Both annotators underwent a half-day training on ambiguity in RE. The two annotators produced their annotations over a six-month span, during which they declared a total of ≈130 and ≈165 hours, respectively.

The annotators were then tasked with independently labeling with *FR* or *SR* all the ("and"/"or") coordination segments in S_c and labeling with *VA* or *NA* all the PP-attachment segments in S_p. The annotators were specifically instructed to ascribe an interpretation to a segment only when they were confident about their interpretation. When in doubt, the annotators labeled the segment in question as ambiguous. An "agreement" between annotators is observed for segment X, when both of them either find X ambiguous or interpret X the same way. Any other situation is a "disagreement". Using Cohen's kappa metric (κ) [81], we obtain an inter-rater agreement of 0.37, suggesting "fair agreement". To examine the sources of disagreement, we further analyze the cases where X is deemed ambiguous (i.e., acknowledged ambiguity). For these cases, we obtain $\kappa = 0.78$ ("substantial agreement"), indicating that most disagreements are due to different interpretations (i.e., unacknowledged ambiguity). As stated earlier in the paper, unacknowledged ambiguity is believed to be prevalent in requirements [3], [23]. The analysis, discussed above, provides empirical evidence for this belief.

We constructed our ground truth as follows: (i) any segment labeled as ambiguous by at least one annotator is a case of *acknowledged ambiguity*, (ii) any segment labeled with different interpretations by the annotators is a case of *unacknowledged ambiguity*, and (iii) any segment labeled with the same interpretation by both annotators is *unambiguous*. We motivate our definitions of acknowledged and unacknowledged ambiguity by considering what might happen during a manual inspection where a team would typically be involved. If a segment is ambiguous enough for someone (not necessarily everyone) to raise a concern, then this segment is likely to be further discussed by the team (acknowledged). The situation is different for unacknowledged ambiguity. In reality and under time pressure, the analysts are unlikely to spell out their interpretations when they feel there is no ambiguity. Consequently, the disagreement remains hidden (unacknowledged).

Table III provides overall statistics about our data collection, showing for each domain, the number of *RD*s, the total number of requirements, and the number of requirements and segments in S_c and S_p. The table further lists the number of ambiguous segments (grouped into acknowledged and unacknowledged) and the number of unambiguous segments. We observe from Table III that out of the total of 2980 segments analyzed by the annotators, 57% are ambiguous and the remaining 43% are unambiguous. In the ambiguous segments, the proportion of segments with unacknowledged ambiguity (1042/1690 ≈ 62%) is significantly higher than the proportion of segments with acknowledged ambiguity (648/1690 ≈ 38%). We note that repeated segments constitute a relatively small fraction of the ground truth: ≈9% (137/1506) for CA and ≈8% (116/1474) for PAA. These repetitions are not disproportionately concentrated in one group. More precisely, in the case of CA, 44 repetitions are unambiguous, 48 are acknowledged, and 45 are unacknowledged. For PAA, 39 repetitions are unambiguous, 35 are acknowledged, and 42 are unacknowledged. Since there is no disproportionate concentration of occurrences, repetitions have no major bearing on our findings.

We set aside ≈20% of our ground truth for parameter tuning, as we will discuss in Section IV-D. We refer to this subset of the ground truth as T. The remaining ≈80% of the ground truth is referred to as E. We use E for answering all the RQs, except RQ4 which is answered over $T \cup E$. The tuning set, T, consists of six *RD*s from six domains with a total of 550 requirements and representing 26% and 21% of the coordination and PP-attachment segments, respectively. We selected one *RD* from each domain; this was done in a way that the selected document would be as close as possible to containing ≈20% of the requirements we had in each domain. We did not select for tuning any documents from the domain of medicine, since we had only one *RD* from this domain.

D. Parameter Tuning

Tuning involves two groups of parameters: parameters for generating a domain-specific corpus (Section III-C) and parameters associated with the heuristics (Section III-D). Both groups of parameters are tuned with the goal of maximizing the overall accuracy of the interpretation heuristics. Note that tuning is performed exclusively over T (see Section IV-C).

Parameters for Corpus Generation. Generating a domain-specific corpus requires tuning the maximum number of keywords (K) to select from an input *RD* and the maximum number of articles (α) in a given category in Wikipedia. For each *RD* in T, we generate a domain-specific corpus. To tune

K, we experiment with five values at regular intervals between 50–250. Values of K outside this range are undesirable as they result in a corpus that is either too small (for $K < 50$) or too large (for $K > 250$). A suitably large corpus is necessary for accurately estimating the co-occurrence frequencies of words in a specific domain [70]. Building and using a corpus that is too large would be time-consuming and, more importantly, would defeat the goal of being domain-specific. Using a corpus that is too small would simply be ineffective. For tuning α, we experiment with values in the range of 50–1000 in intervals of 50. Larger categories (i.e., $\alpha > 1000$) are too generic, and smaller ones (with $\alpha < 50$) are already covered by $\alpha > 50$, as α is the upper bound for the number of articles in a category. For optimizing K and α, we use grid search [82]. The resulting optimal values are $K = 100$ and $\alpha = 250$.

Parameters for Heuristics. Applying the interpretation heuristics requires tuning four thresholds θ_1, θ_3–θ_5 respectively for heuristics $\mathcal{C}_1$, $\mathcal{C}_3$–$\mathcal{C}_5$. For using the weighted voting method, we further need to tune the weights of all the heuristics.

We note that the thresholds for the heuristics have been introduced and tuned in the existing literature, albeit for generic texts [23], [31], [32]. We re-tune these thresholds to better capture co-occurrence frequencies in the context of requirements. The threshold values from the existing literature are hereafter referred to as *default*. We experiment with 1000 regular intervals in the range of 0.01–10 for tuning θ_1, θ_3 and θ_4. To tune θ_5, we investigate suffixes of lengths 1 to 5, e.g., the suffix "-ation" has a length of five. We use random search [82] to optimize the thresholds because the search space is too large for grid search. The resulting *optimal* thresholds are $\theta_1 = 0.01$, $\theta_3 = 0.12$, $\theta_4 = 3.45$, and $\theta_5 = 3$.

For determining the weights of the heuristics, we first apply each heuristic individually on T. The weight of a given heuristic is determined by its success in providing interpretations for the segments. In our experiments, the weights of heuristics in descending order for CA are 0.038 for $\mathcal{C}_5$, 0.019 for $\mathcal{C}_2$, 0.012 for $\mathcal{C}_1$, 0.005 for $\mathcal{C}_4$, 0.005 for $\mathcal{C}_6$ and 0.003 for $\mathcal{C}_3$, and the weights for PAA are 0.08 for $\mathcal{P}_1$, 0.05 for $\mathcal{P}_2$ and 0.03 for $\mathcal{P}_4$. $\mathcal{P}_3$ is not a standalone heuristic and is thus not weighted. These weights reflect the contribution of the heuristics, in weighted voting, to produce a final interpretation for a segment.

E. Evaluation Procedure

We answer our RQs through the following experiments.

EXPI. This experiment answers **RQ1**. We determine the optimal configuration for ambiguity handling by comparing the output of our approach against E. For evaluating the configurations, we define a *true positive (TP)* as a detected ambiguous segment, a *true negative (TN)* as an unambiguous segment marked as such, a *false positive (FP)* as a misclassified unambiguous segment, and a *false negative (FN)* as a misclassified ambiguous segment. We compute *Accuracy* (A) as $(TP + TN)/(TP + TN + FP + FN)$, *Precision* ($P$) as $TP/(TP + FP)$, and *Recall* (R) as $TP/(TP + FN)$.

We consider eight alternative configurations for our approach, denoted as $\texttt{V}_1$–$\texttt{V}_8$. These alternatives are induced by three binary decisions. The first decision is whether to use the *collected* or the *enhanced* patterns in step 2 of our approach (see Table I). The second decision is whether in step 4 we should use for the thresholds the *default* or the *optimal* values (from Section IV-D). And, the third decision is whether the method for combining the heuristics is *majority* or *weighted* voting (see Section III-D). To analyze the impact of using domain-specific corpora, we compare our approach against baselines, denoted as $\texttt{B}_1$–$\texttt{B}_8$, with similar configurations but using a generic corpus: the British National Corpus [83].

To run EXPI, we first need to generate seven corpora, one for each application domain in our ground truth (see Table III). Six of these corpora are reused from Section IV-D. The last one – for the domain of medicine – is generated based on the single *RD* we have in our dataset for this domain. Except for the domain of medicine, EXPI provides an implicit assessment of how reusable a domain-specific corpus is, being generated from one *RD* and reused for other *RD*s from the same domain.

EXPII. This experiment answers **RQ2**. Given the optimal configuration of our approach from EXPI, EXPII assesses how well our approach can detect unacknowledged ambiguity in different domains. In EXPII, we compute *Recall (R)* similar to EXPI, but limiting the evaluation to only the segments with unacknowledged ambiguity in E. The corpora used in EXPII are the same as those in EXPI.

EXPIII. This experiment answers **RQ3**. We evaluate the interpretations provided by our approach for the segments classified as unambiguous (*FR* or *SR* for segments in S_c, and *VA* or *NA* for segments in S_p). Specifically, EXPIII compares the interpretations produced by our approach against the interpretations of unambiguous segments in E, reporting the ratio of the correctly interpreted segments (i.e., *Accuracy*). The corpora used in EXPIII are the same as those in EXPI and EXPII. We further compare our approach against Stanford Parser [78] – one of the commonly used tools for interpreting syntactic ambiguity [84].

EXPIV. This experiment answers **RQ4** by running the best configuration from RQ1 over $T \cup E$. The experiment is done on a laptop with a 2.3 GHz CPU and 16GB of memory.

F. Answers to the RQs

RQ1. Table IV shows the results of EXPI (on E). To determine the optimal configuration of our approach, we investigate among all configurations the factors that cause the most variation in accuracy. We do so by performing regression tree analysis (tree not shown) [85]. The most influential factor for both CA and PAA, as per regression tree analysis, is the choice of domain-specific versus generic corpus. *The configurations that use a domain-specific corpus,* $\texttt{V}_1$–$\texttt{V}_8$*, have an average gain in accuracy of ≈33% over the configurations that use a generic corpus,* $\texttt{B}_1$–$\texttt{B}_8$*. This observation clearly highlights the importance of domain knowledge in ambiguity handling.*

Among $\texttt{V}_1$–$\texttt{V}_8$, using enhanced patterns has a considerable impact on detecting CA. Compared to the configurations with collected patterns ($\texttt{V}_1$–$\texttt{V}_4$), the configurations with enhanced patterns ($\texttt{V}_5$–$\texttt{V}_8$) lead to an average gain of ≈6% in accuracy

TABLE IV
RESULTS OF AMBIGUITY HANDLING (RQ1).

					CA			PAA		
		Patterns	Thresholds	Voting	A (%)	P (%)	R (%)	A (%)	P (%)	R (%)
V_1	Domain-Specific Corpus	collected	default	majority	70.8	76.9	66.6	78.4	77.8	84.0
V_2		collected	default	weighted	71.6	78.0	66.9	78.9	78.5	84.0
V_3		collected	optimal	majority	74.5	81.1	69.5	82.0	81.5	86.4
V_4		collected	optimal	weighted	75.3	82.2	69.9	82.5	82.2	86.4
V_5		enhanced	default	majority	76.7	75.6	84.6	78.6	76.3	87.7
V_6		enhanced	default	weighted	77.5	76.4	84.9	79.1	76.9	87.7
V_7		enhanced	optimal	majority	80.5	78.9	87.6	82.2	79.8	90.1
V_8		enhanced	optimal	weighted	81.3	79.8	87.9	82.7	80.3	90.1
B_1	British National Corpus	collected	default	majority	37.6	42.5	40.6	45.7	49.9	53.5
B_2		collected	default	weighted	38.4	43.2	40.9	46.2	50.2	53.5
B_3		collected	optimal	majority	41.3	46.0	43.6	49.3	53.1	55.9
B_4		collected	optimal	weighted	42.1	46.8	43.9	49.8	53.5	55.9
B_5		enhanced	default	majority	42.9	48.2	60.4	45.8	50.0	57.2
B_6		enhanced	default	weighted	43.7	48.8	60.8	46.2	50.3	57.2
B_7		enhanced	optimal	majority	46.7	50.9	63.3	49.4	52.9	59.6
B_8		enhanced	optimal	weighted	47.5	51.6	63.8	49.9	53.3	59.6

Accuracy (A), Precision (P) and Recall (R) in percentage (%)

TABLE V
UNACKNOWLEDGED AMBIGUITY DETECTION USING V_8 (RQ2).

	Domain	Aerospace	Automative	Defense	Digitalization	Medicine	Satellite	Security	Summary
CA	TP	145	13	17	28	19	46	0	**268**
	FN	25	2	2	4	0	6	0	**39**
	R (%)	84.7	86.6	89.4	87.5	100	88.4	-	**87.3**
PAA	TP	83	15	120	17	4	141	32	**412**
	FN	6	2	11	1	1	12	4	**37**
	R (%)	93.2	88.2	91.6	94.4	80.0	92.1	88.8	**91.8**

TP, FN: number of true positives and false negatives, R: Recall in percentage (%)

and ≈18% in recall for a minor ≈2% drop in precision. Compared to collected patterns, enhanced patterns do not improve the detection of PAA, but do not perform any worse either. Thus, we choose the enhanced patterns over the collected ones.

With respect to the thresholds for the heuristics, the configurations with optimal thresholds (V_7–V_8) outperform those with default thresholds (V_5–V_6) by 3.7% in terms of accuracy. Noting that our parameter tuning used documents from six different application domains, we believe that the optimal thresholds are more suitable in an RE context than the default ones based on generic texts. We note that, overall, the accuracy of ambiguity handling shows little sensitivity to the choice of voting method. However, as highlighted in Table IV, V_8 (weighted voting) is slightly more accurate than V_7 (majority voting). For the subsequent RQs, *we select* V_8 *as the best-performing configuration of our approach with enhanced patterns, optimal thresholds and weighted voting.*

To be able to perform a thorough error analysis (Section IV-G), we run V_8 on the entire dataset ($T \cup E$). This yields a precision and recall of 80.1% and 89.3% for CA, and 81.6% and 90.2% for PAA, respectively. We observe that, for each metric, the overall results are only marginally (≈1%) better than what was reported over E. This provides confidence that our tuning (Section IV-D) did not overfit.

RQ2. The results of EXPII, obtained from running V_8 (the best configuration from RQ1) *on* E are shown in Table V. Overall, *our approach detects unacknowledged ambiguity with an average recall of 87.3% for CA and 91.8% for PAA.*

Our error analysis (Section IV-G) examines missed cases of unacknowledged ambiguity in the entire dataset ($T \cup E$). Over the entire dataset, V_8 detects unacknowledged ambiguity with an average recall of 87.8% for CA and 92.6% for PAA.

RQ3. The interpretations provided by V_8 for the segments in S_c and S_p (when restricted to E) have an average accuracy of 85.2% and 84.4%, respectively. The accuracy of the approach on the entire dataset is marginally higher (by an average of ≈1%). We examine interpretations errors in Section IV-G.

Applying the Stanford Parser to S_c and S_p (when restricted to E) yields interpretations with an average accuracy of 65.7% and 72.6%, respectively. *In an RE context and in comparison to the Stanford Parser, the integration of domain knowledge increases the interpretation accuracy of coordination and PP-attachment structures by an average of ≈16%.*

RQ4. Executing steps 1 and 2 of our approach (Fig. 3) takes ≈0.2 milliseconds per requirement. Step 3 is performed only when a suitable corpus is absent, i.e., when no corpus has been generated before for the domain of a given *RD*, or when the domain of the *RD* is difficult to ascertain. Across the seven corpora we generated for answering RQ1-3, the average execution time was ≈58 minutes (standard deviation: ≈21 minutes). To be able to generate corpora, there is a one-time overhead of ≈3 hours; this is to set up a query engine over Wikipedia (see step 3.2 in Section III-C). Once set up, this query engine does not have to be rebuilt, unless one wants to switch to a different edition of Wikipedia. With a corpus at hand, execution time is dominated by the computation of the frequencies required by the heuristics of step 4. This on average takes ≈6.8 seconds for a requirement in S_c and ≈1.5 seconds for one in S_p. Processing the requirements in S_c takes longer because there are more corpus-based heuristics for CA than PAA. Non-corpus-based heuristics take negligible time.

Excluding corpus generation, the largest document in our dataset took ≈51 minutes to process. This document had 492 requirements with 392 coordination and 245 PP-attachment segments. *Such an execution time is practical for offline (e.g., overnight) processing.* With regard to using our approach interactively, we observe that, at any point in time, an analyst likely works on only a small part of a large document. *For interactive use, ambiguity handling can be localized to the document fraction (e.g., page) that the analyst is reviewing.*

G. Error Analysis

In this section, we analyze the root causes of the errors made by our approach (V_8) on the entire dataset ($T \cup E$).

Errors in RQ1 and RQ2. Out of 1690 segments (Table III), our approach missed 192 ambiguous segments, of which 100 are unacknowledged. These errors can be explained as follows.

1) *Coverage of patterns: 169 segments* do not match any pattern in Table I. For example, the segment "register the

microservice in the operations server" matches no PAA pattern. One can avoid such errors by expanding the pattern set. However, our experiments indicate that doing so comes at the cost of a large number of FPs and is thus not worthwhile.

2) *NLP errors: 23 segments* are missed due to mistakes by the NLP pipeline (Fig. 4). For example, "support" in the segment "[can] support doctors in the ICU" is erroneously tagged as a noun; this results in the segment to not match any of our patterns. Such NLP mistakes are hard to avoid [86].

Errors in RQ3. We found two causes for interpretation errors.

1) Interpretation errors by the heuristics: 74 segments fall under this class of errors, having to do with situations where the combination of heuristics provide a wrong interpretation or return *not interpretable (NI)* where there is indeed an interpretation. For example, for the segment "pulse width and duration" the resulting interpretation is *SR*, although it should be *FR*. One can try to address individual interpretation errors by adjusting the weights of the heuristics. However, doing so will have a negative overall impact by causing other errors.

2) Document-specific abbreviations: 58 segments are misinterpreted due to abbreviations. An abbreviation that is specific to a document can mislead frequency computations if the abbreviation has a homonym or is not found in the corpus at all. For example, "MOC" in "MOC operator and component" stands for "MOnitoring and Control" in one of our *RD*s from the satellite domain. This abbreviation, however, matches "Mars Orbiter Camera" in the corpus that we generate for this domain. Such mismatches can be reduced through abbreviation disambiguation [87]. We leave this for future work.

H. Discussion about Usefulness

As shown by Table III, ambiguity was acknowledged by the annotators in only 38% of the cases. The remaining 62% were unacknowledged. In practice, even if the analysts perform a manual review, under time pressure and outside an evaluation setting, they will likely only examine what at least one analyst finds to be ambiguous and thus miss out on the cases where they unconsciously disagree about the interpretation.

We believe that the main benefit of our automated approach is in bringing ≈90% of the cases of unacknowledged ambiguity to the attention of the analysts (see RQ2). This is achieved while maintaining a high overall precision (≈80%), meaning that the analysts will spend a small fraction of their manual effort over false positives. While user studies remain essential for establishing usefulness, our good accuracy results suggest that our approach has the potential to be helpful in practice.

V. Validity Considerations

Internal Validity. Bias is a potential threat to the internal validity of our evaluation. To mitigate this threat, the authors had no involvement in the annotation activities. Instead, two third-party annotators, who had no knowledge of our technical approach, independently annotated the dataset. Further, we made a strict separation between the data used for defining patterns and tuning, and the data used for assessing effectiveness.

Construct Validity. An individual requirement can potentially have multiple instances of CA or PAA. To ensure that this possibility is properly reflected in our metrics, we defined accuracy, precision and recall at the level of segments rather than whole requirements.

External Validity. Our evaluation builds on 20 industrial requirements documents, covering seven different domains. The promising results obtained across these domains provide a measure of confidence about the generalizability of our approach. This confidence is further strengthened by the fact that our approach can adapt itself to new domains via the (automatic) generation of domain-specific corpora. Due to this characteristic, we are optimistic that our approach will be able to achieve comparable results in other domains. That said, future case studies would help further improve external validity.

VI. Conclusion

In this paper, we proposed an automated approach for improving the handling of coordination ambiguity (CA) and prepositional-phrase attachment ambiguity (PAA). The main novelty of our approach is in automatically extracting domain-specific corpora from Wikipedia and utilizing them for increasing the accuracy of CA and PAA handling in requirements documents. We conducted a large-scale evaluation of our approach using more than 5000 industrial requirements from seven different application domains. Our results indicate that our approach can detect CA and PAA with an average precision of ≈80% and an average recall of ≈89%. The results further indicate that employing domain-specific corpora has a substantial positive impact on the accuracy of CA and PAA handling. Specifically, over our dataset, we observed a ≈33% improvement in accuracy when compared against baselines that use generic corpora. While our work is motivated by improving the quality of systems and software requirements, our technical solution is also novel from an NLP standpoint. Our solution thus has the potential to be useful over other types of textual documents within and beyond software engineering.

In future work, we would like to integrate our ambiguity handling approach with automated techniques for extracting structured information from requirements specifications. The motivation for doing so is to increase the quality of information extraction by more accurately interpreting coordination and prepositional-phrase structures. Another direction we would like to explore in the future is to use deep learning to complement or as an alternative to our current approach.

Acknowledgement. This work has received funding from Luxembourg's National Research Fund (FNR) under the grant BRIDGES18/IS/12632261 and from NSERC of Canada under the Discovery, Discovery Accelerator and CRC programs. We are grateful to the research and development team at QRA Corp. (Canada) for very valuable insights and assistance.

REFERENCES

[1] F. de Bruijn and H. Dekkers, "Ambiguity in natural language software requirements: A case study," in *Proceedings of the 16th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'10)*, 2010.

[2] K. Pohl, *Requirements Engineering*, 1st ed. Springer, 2010.

[3] D. Berry, E. Kamsties, and M. Krieger, "From contract drafting to software specification: Linguistic sources of ambiguity, a handbook," 2003. [Online]. Available: http://se.uwaterloo.ca/~dberry/handbook/ambiguityHandbook.pdf

[4] S. Piantadosi, H. Tily, and E. Gibson, "The communicative function of ambiguity in language," *Cognition*, vol. 122, no. 3, 2012.

[5] K. Pohl and C. Rupp, *Requirements Engineering Fundamentals*, 1st ed. Rocky Nook, 2011.

[6] A. Ferrari and A. Esuli, "An NLP approach for cross-domain ambiguity detection in requirements engineering," *Automated Software Engineering*, vol. 26, no. 3, 2019.

[7] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis, "Identifying nocuous ambiguities in natural language requirements," in *Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, 2006.

[8] V. Gervasi, A. Ferrari, D. Zowghi, and P. Spoletini, "Ambiguity in requirements engineering: Towards a unifying framework," in *From Software Engineering to Formal Methods and Tools, and Back*. Springer, 2019.

[9] E. Kamsties, D. Berry, and B. Paech, "Detecting ambiguities in requirements documents using inspections," in *Proceedings of the 1st Workshop on Inspection in Software Engineering (WISE'01)*, 2001.

[10] N. Kiyavitskaya, N. Zeni, L. Mich, and D. Berry, "Requirements for tools for ambiguity identification and measurement in natural language requirements specifications," *Requirements Engineering*, vol. 13, no. 3, 2008.

[11] F. Dalpiaz, I. Schalk, and G. Lucassen, "Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP," in *Proceedings of the 24th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'18)*, 2018.

[12] P. Spoletini, A. Ferrari, M. Bano, D. Zowghi, and S. Gnesi, "Interview review: An empirical study on detecting ambiguities in requirements elicitation interviews," in *Proceedings of the 24th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'18)*, 2018.

[13] H. Yang, A. de Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Analysing anaphoric ambiguity in natural language requirements," *Requirements Engineering*, vol. 16, no. 3, 2011.

[14] F. Dalpiaz, D. Dell'Anna, F. Aydemir, and S. Cevikol, "Requirements classification with interpretable machine learning and dependency parsing," in *Proceedings of the 27th IEEE International Requirements Engineering Conference (RE'19)*, 2019.

[15] S. Mishra and A. Sharma, "On the use of word embeddings for identifying domain specific ambiguities in requirements," in *Proceedings of the 27th IEEE International Requirements Engineering Conference Workshops (REW'19)*, 2019.

[16] D. Toews and L. Van Holland, "Determining domain-specific differences of polysemous words using context information." in *Proceedings of the 25th Working Conference on Requirements Engineering: Foundation and Software Quality Workshops (REFSQW'19)*, 2019.

[17] V. Jain, R. Malhotra, S. Jain, and N. Tanwar, "Cross-domain ambiguity detection using linear transformation of word embedding spaces," in *Proceedings of the 26th Working Conference on Requirements Engineering: Foundation and Software Quality Workshops (REFSQW'20)*, 2020.

[18] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Extracting domain models from natural-language requirements: approach and industrial evaluation," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS'16)*, 2016.

[19] A. Sleimi, N. Sannier, M. Sabetzadeh, L. Briand, and J. Dann, "Automated extraction of semantic legal metadata using natural language processing," in *Proceedings of the 26th IEEE International Requirements Engineering Conference (RE'18)*, 2018.

[20] C. Schütze, "PP attachment and argumenthood," *MIT working papers in linguistics*, vol. 26, no. 95, 1995.

[21] P. Engelhardt and F. Ferreira, "Processing coordination ambiguity," *Language and Speech*, vol. 53, no. 4, 2010.

[22] B. Strang, *Modern English Structure*, 2nd ed. Edward Arnold, 1968.

[23] F. Chantree, A. Kilgarriff, A. De Roeck, and A. Willis, "Disambiguating coordinations using word distribution information," in *Proceedings of the 5th International Conference on Recent Advances in Natural Language Processing (RANLP'05)*, 2005.

[24] M. Goldberg, "An unsupervised model for statistically determining coordinate phrase attachment," in *Proceedings of the 37th annual meeting of the Association for Computational Linguistics (ACL'99)*, 1999.

[25] P. Resnik, "Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language," *Journal of Artificial Intelligence Research*, vol. 11, no. 1, 1999.

[26] P. Nakov and M. Hearst, "Using the web as an implicit training set: application to structural ambiguity resolution," in *Proceedings of the 5th conference on Human Language Technology and Empirical Methods in Natural Language Processing (HLT'05)*, 2005.

[27] A. De Roeck, "Detecting dangerous coordination ambiguities using word distribution," in *Proceedings of the 6th International Conference on Recent Advances in Natural Language Processing (RANLP'07)*, 2007.

[28] S. Tjong and D. Berry, "Can rules of inferences resolve coordination ambiguity in natural language requirements specification?" in *Proceedings of the 13th Workshop on Requirements Engineering (WER'08)*, 2008.

[29] H. Yang, A. Willis, A. De Roeck, and B. Nuseibeh, "Automatic detection of nocuous coordination ambiguities in natural language requirements," in *Proceedings of the 10th IEEE/ACM international conference on Automated software engineering (ASE'10)*, 2010.

[30] S. Tjong and D. Berry, "The design of SREE—a prototype potential ambiguity finder for requirements specifications and lessons learned," in *Proceedings of the 19th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'13)*, 2013.

[31] A. Kilgarriff, "Thesauruses for natural language processing," in *Proceedings of the 1st International Conference on Natural Language Processing and Knowledge Engineering (NLPKE'03)*, 2003.

[32] H. Yang, A. De Roeck, A. Willis, and B. Nuseibeh, "A methodology for automatic identification of nocuous ambiguity," in *Proceedings of the 23rd International Conference on Computational Linguistics (COLING'10)*, 2010.

[33] A. Okumura and K. Muraki, "Symmetric pattern matching analysis for English coordinate structures," in *Proceedings of the 4th Conference on Applied Natural Language Processing (ANLP'94)*, 1994.

[34] E. Agirre, T. Baldwin, and D. Martínez, "Improving parsing and PP attachment performance with sense information," in *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL'08)*, 2008.

[35] H. Calvo and A. Gelbukh, "Improving prepositional phrase attachment disambiguation using the web as corpus," in *Proceedings of the 8th Iberoamerican Congress on Progress in Pattern Recognition, Speech and Image Analysis (CIARP'03)*, 2003.

[36] M. B. Hosseini, R. Slavin, T. Breaux, X. Wang, and J. Niu, "Disambiguating requirements through syntax-driven semantic analysis of information types," in *Proceedings of the 26th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'20)*, 2020.

[37] U. Shah and D. Jinwala, "Resolving ambiguities in natural language software requirements: A comprehensive survey," *SIGSOFT Software Engineering Notes*, vol. 40, no. 5, 2015.

[38] C. Ribeiro and D. Berry, "The prevalence and severity of persistent ambiguity in software requirements specifications: Is a special effort needed to find them?" *Science of Computer Programming*, vol. 195, 2020.

[39] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "The linguistic approach to the natural language requirements quality: Benefit of the use of an automatic tool," in *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop (SEW'01)*, 2001.

[40] E. Kamsties and B. Peach, "Taming ambiguity in natural language requirements," in *Proceedings of the 13th International Conference on Software and Systems Engineering and Applications (ICSSEA'00)*, 2000.

[41] A. Massey, R. Rutledge, A. Anton, and P. Swire, "Identifying and classifying ambiguity for regulatory requirements," in *Proceedings of the 22nd IEEE International Requirements Engineering Conference (RE'14)*, 2014.

[42] L. Mich, "NL-OOPS: From natural language to object oriented requirements using the natural language processing system LOLITA," *Natural Language Engineering*, vol. 2, no. 2, 1996.

[43] V. Ambriola and V. Gervasi, "On the systematic analysis of natural language requirements with CIRCE," *Automated Software Engineering*, vol. 13, no. 1, 2006.

[44] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak, "Easy approach to requirements syntax (EARS)," in *Proceedings of the 17th IEEE International Requirements Engineering Conference (RE'09)*, 2009.

[45] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated checking of conformance to requirements templates using natural language processing," *IEEE Transactions on Software Engineering*, vol. 41, no. 10, 2015.

[46] D. Rodriguez, D. Carver, and A. Mahmoud, "An efficient wikipedia-based approach for better understanding of natural language text related to user requirements," in *Proceedings of the 39th IEEE Aerospace Conference (AeroConf'18)*, 2018.

[47] B. Gleich, O. Creighton, and L. Kof, "Ambiguity detection: Towards a tool explaining ambiguity sources," in *Proceedings of the 16th Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'10)*, 2010.

[48] H. Femmer, D. Méndez Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *Journal of Systems and Software*, vol. 123, 2017.

[49] B. Rosadini, A. Ferrari, G. Gori, A. Fantechi, S. Gnesi, I. Trotta, and S. Bacherini, "Using NLP to detect requirements defects: An industrial experience in the railway domain," in *Proceedings of the 23rd Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'17)*, 2017.

[50] A. Ferrari, G. Gori, B. Rosadini, I. Trotta, S. Bacherini, A. Fantechi, and S. Gnesi, "Detecting requirements defects with NLP patterns: An industrial experience in the railway domain," *Empirical Software Engineering*, vol. 23, no. 6, 2018.

[51] G. Lami, M. Fusani, and G. Trentanni, "QuARS: A pioneer tool for NL requirement analysis," in *From Software Engineering to Formal Methods and Tools, and Back*. Springer, 2019.

[52] F. Dalpiaz, I. van der Schalk, S. Brinkkemper, F. Aydemir, and G. Lucassen, "Detecting terminological ambiguity in user stories: Tool and experimentation," *Information and Software Technology*, vol. 110, 2019.

[53] A. Willis, F. Chantree, and A. De Roeck, "Automatic identification of nocuous ambiguity," *Research on Language and Computation*, vol. 6, no. 3-4, 2008.

[54] K. Church and R. Patil, *Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table*, 1st ed. MIT Press, 1982.

[55] P. Pantel and D. Lin, "An unsupervised approach to prepositional phrase attachment using contextually similar words," in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics (ACL'00)*, 2000.

[56] E. Agirre, O. de Lacalle, C. Fellbaum, A. Marchetti, A. Toral, and P. Vossen, "SemEval-2010 task 17: all-words word sense disambiguation on a specific domain," in *Proceedings of the 5th Workshop on Semantic Evaluations: Recent Achievements and Future Directions (SEW'10)*, 2010.

[57] M. Strube and S. Ponzetto, "WikiRelate! computing semantic relatedness using Wikipedia," in *Proceedings of the 21st national conference on Artificial intelligence (AAAI'06)*, 2006.

[58] E. Gabrilovich, S. Markovitch *et al.*, "Computing semantic relatedness using wikipedia-based explicit semantic analysis." in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, 2007.

[59] A. Fogarolli, "Word sense disambiguation based on Wikipedia link structure," in *Proceedings of the 3rd IEEE International Conference on Semantic Computing (ICSC'09)*, 2009.

[60] S. Gella, C. Strapparava, and V. Nastase, "Mapping WordNet domains, WordNet topics and Wikipedia categories to generate multilingual domain specific resources," in *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC'14)*, 2014.

[61] G. Miller, "WordNet: A lexical database for English," *Communications of the ACM*, vol. 38, no. 11, 1995.

[62] C. Fellbaum, *WordNet: An Electronic Lexical Database*, 1st ed. The MIT Press, 1998.

[63] D. Chen and C. Manning, "A fast and accurate dependency parser using neural networks," in *Proceedings of the 18th Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*, 2014.

[64] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, "Automated extraction and clustering of requirements glossary terms," *IEEE Transactions on Software Engineering*, vol. 43, no. 10, 2017.

[65] D. Newman, J. Lau, K. Grieser, and T. Baldwin, "Automatic evaluation of topic coherence," in *Proceedings of the 8th annual conference of the North American chapter of the association for computational linguistics: Human language technologies (NAACL-HLT'10)*, 2010.

[66] S. Evert, "Google web 1T 5-grams made easy (but not for the computer)," in *Proceedings of the 8th annual conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT'10) and the 6th Web as Corpus Workshop (WAC'10)*, 2010.

[67] C. Biemann, F. Bildhauer, S. Evert, D. Goldhahn, U. Quasthoff, R. Schäfer, J. Simon, L. Swiezinski, and T. Zesch, "Scalable construction of high-quality web corpora." *Journal for Language Technology and Computational Linguistics*, vol. 28, no. 2, 2013.

[68] T. Yen, J. Wu, J. Chang, J. Boisson, and J. Chang, "WriteAhead: Mining grammar patterns in corpora for assisted writing," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Proceedings of System Demonstrations (ACL-IJCNLP'15)*, 2015.

[69] T. Hawker, "USYD: WSD and lexical substitution using the Web1T corpus," in *Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval'07)*, 2007.

[70] D. Jurafsky and J. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed. Prentice Hall, 2009.

[71] C. Manning and H. Schütze, *Foundations of statistical natural language processing*, 1st ed. MIT press, 1999.

[72] G. Dinu and M. Lapata, "Measuring distributional similarity in context," in *Proceedings of the 14th Conference on Empirical Methods in Natural Language Processing (EMNLP'10)*, 2010.

[73] L. J. Brinton, *The structure of modern English: A linguistic introduction*. John Benjamins Publishing, 2000.

[74] I. Witten, E. Frank, M. Hall, and C. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. Elsevier, 2011.

[75] R. Eckart de Castilho and I. Gurevych, "A broad-coverage collection of portable NLP components for building shareable analysis pipelines," in *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT (OIAF4HLT'14)*, 2014.

[76] T. Zesch, C. Müller, and I. Gurevych, "Extracting lexical semantic knowledge from Wikipedia and Wiktionary," in *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC'08)*, 2008.

[77] C. Giuliano, "jWeb1T: A library for searching the web 1T 5-gram corpus," last accessed: August 2020. [Online]. Available: http://hlt.fbk.eu/en/technology/jWeb1t

[78] M. Zhu, Y. Zhang, W. Chen, M. Zhang, and J. Zhu, "Fast and accurate shift-reduce constituent parsing," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL'13)*, 2013.

[79] P. Resnik, "Using information content to evaluate semantic similarity in a taxonomy," in *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, 1995.

[80] H. Shima, "WS4J WordNet similarity for java," last accessed: August 2020. [Online]. Available: https://code.google.com/archive/p/ws4j/

[81] J. R. Landis and G. G. Koch, "An application of hierarchical kappa-type statistics in the assessment of majority agreement among multiple observers," *Biometrics*, vol. 33, no. 2, 1977.

[82] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 1, 2012.

[83] G. Leech, "100 million words of English," *English Today*, vol. 9, no. 1, 1993.

[84] J. Hirschberg and C. Manning, "Advances in natural language processing," *Science*, vol. 349, no. 6245, 2015.

[85] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification And Regression Trees*, 1st ed. Routledge, 1984.

[86] Y. Tian and D. Lo, "A comparative study on the effectiveness of part-of-speech tagging techniques on bug reports," in *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)*, 2015.

[87] J. Charbonnier and C. Wartena, "Using word embeddings for unsupervised acronym disambiguation," in *Proceedings of the 27th International Conference on Computational Linguistics (COLING'18)*, 2018.