2022 IEEE 30th International Requirements Engineering Conference (RE) | 978-1-6654-7000-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/RE54965.2022.00012

Automatic Terminology Extraction and Ranking for Feature Modeling

Jianzhang Zhang*, Sisi Chen*, Jinping Hua*, Nan Niu†, Chuang Liu*
*Alibaba Business School, Hangzhou Normal University, Hangzhou, China
†Department of Electrical Engineering & Computer Science, University of Cincinnati, USA
jianzhang.zhang@foxmail.com, {chensisi1, huajinping}@stu.hznu.edu.cn, nan.niu@uc.edu, liuchuang@hznu.edu.cn

***Abstract*—Requirements terminology defines and unifies key specialized and/or technical concepts of the software system, which is significant for understanding the application domain in requirements engineering (RE). However, manual terminology extraction from natural language requirements is laborious and expensive, especially with large scale requirements specifications. In this paper, we aim to employ natural language processing (NLP) techniques and machine learning (ML) algorithms to automatically extract and rank the requirements terms to support high-level feature modeling. To this end, we propose an automatic framework composed of noun phrase identification technique for requirements terms extraction and TextRank combined with semantic similarity for terms ranking. The final ranked terms are organized as a hierarchy, which can be used to help name elements when performing feature modeling. In the quantitative evaluation, our extraction method performs better than three baseline methods in recall with comparable precision. Moreover, our adapted TextRank algorithm can rank more relevant terms at the top positions in terms of average precision compared with most baselines. An illustrative example on the smart home domain further shows the usefulness of our framework in aiding elements naming during feature modeling. The research results suggest that proper adoption and adaption of NLP and ML techniques according to the characteristics of specific RE task could provide automation support for problem domain understanding.**

***Index Terms*—terminology extraction, TextRank, text mining, requirements analysis, feature modeling**

I. Introduction

Terminology refers to a group of single-word and/or multi-word expressions that are given specific meanings within a specialized domain. The first step to understand and model a knowledge domain is to build a vocabulary of domain-relevant terms which are the linguistic surface manifestation of domain concepts. Requirements terminology helps the stakeholders share a common understanding of the key concepts within the problem domain, thus mitigating misinterpretation and promoting communication among them [1]. Once constructed, requirements terminology can be further used to enhance many other requirements elaboration activities, such as feature request extraction [2; 3], abstraction identification [4], and feature modeling [5].

This paper is supported by Hangzhou Normal University Scientific Research Staring Foundation (Grant No.4135C50220204073), the Open Project of Zhejiang Key Laboratory of Film and Television Media Technology Research (Grant No.CM2021003), and the National Natural Science Foundation of China (Grant No.61873080).

Natural language has been the most common notation for expressing requirements in the industrial practice [6]. Improvements and advances in natural language processing (NLP) technologies have motivated researchers to explore a range of NLP techniques for various requirements engineering (RE) tasks [7], which leads to the emerging and growing of the research area natural language processing for requirements engineering (NLP4RE). Requirements terms indicating significant domain concepts have proved to be fundamental for informing analysts' extracting requirements-related knowledge [8] and understanding the problem domain [9].

Terminology extraction aims to automatically extract relevant terms from a given corpus to support a variety of applications relying on document meaning. Though widely studied, existing terminology extraction methods are mostly designed and evaluated based on scientific documents corpus [10], e.g., research papers. These methods, when directly applied to software requirements, may yield poor results due to the fact that they are not tailored to requirements specifications. The best practice in RE [11] recommends determining requirements terms when the system requirements are being elicited. However, from a cost-effectiveness standpoint in industrial projects, requirements terms may be extracted *after the fact*, i.e., when the requirements have sufficiently stabilized [12].

Requirements terminology extraction is a time-consuming and expensive task, especially for large-scale requirements specifications. The existing requirements terminology extraction work mainly focuses on extracting a flat list of expressions from the requirements documents. When the results list is large, manually filtering noise is cost-ineffective. Furthermore, The flat term list alone without any explicit structure can only provide fragmented information on the problem domain, limiting the potential of requirements terminology as an aid to many specific problem domain understanding tasks, e.g., feature modeling.

Our goal is to accurately extract requirements terms from textual requirements specifications, and ranking more relevant and also more abstract ones in the top positions to support easily selecting suitable elements' names when performing feature modeling.

To this end, we propose an automatic framework for extracting and ranking requirements terms based on noun phrase identification and an adaptation of TextRank separately. The output is a ranked list of requirements terms with a hierarchical

978-1-6654-7000-1/22/$31.00 ©2022 IEEE
DOI 10.1109/RE54965.2022.00012

structure which can inform analysts in feature modeling.

To evaluate our proposed framework, we specifically investigate three research questions:

- **RQ1**: *How effectively does our noun phrase identification method extract requirements terms compared with baselines for requirements terminology extraction and general terminology extraction?*
- **RQ2**: *How effectively does our adaption of TextRank rank more relevant terms in the top positions compared with other TextRank variants?*
- **RQ3**: *How to use the ranked term list produced by our adaptation of TextRank to help the construction of feature models?*

Through RQ1 and RQ2, we want to compare the performance of our extraction and ranking methods with the existing requirements terminology extraction methods and some popular terminology ranking methods based on TextRank. Through RQ3, we attempt to identify the potential usage of the ranked term list produce by our method in feature modeling.

In summary, we make the following contributions:

1) Design a noun phrase identification based method for extracting requirements terms, which performs better than three baselines in recall while not sacrificing too much precision.
2) Adapt the unsupervised graph-based algorithm TextRank with terms as nodes and semantic similarity as edge weights for ranking the extracted requirements terms to make more relevant ones in the top position, which performs better than most baselines (except for one) in terms of average precision.
3) Construct an illustrative feature model in smart home domain to demonstrate that the ranked terms with an implicit hierarchy structure produced by our adaption of TextRank can provide assistance in feature modeling, e.g., helping easily select suitable names for elements from the ranked term list.

The rest of this article is structured as follows: Section II presents the overview of our framework followed by detailing the methods employed in requirements terminology extraction and ranking modules. Section III describes the experimental settings including dataset, ground truth, baselines, and evaluation measures. Section IV analyzes and discusses the quantitative and qualitative experiment results to answer the research questions. Section V discusses threats to validity. Section VI compares our study with related work and Section VII concludes the paper with several valuable future extension avenues.

II. Automatic Requirements Terminology Extraction and Ranking Framework

In this section, we start with briefly introducing the whole framework followed by elaborating two automated components in detail, i.e., terminology extraction and ranking.

Figure 1 displays the overall pipeline of our proposed feature modeling oriented requirements terminology extraction and ranking framework.

- **Input**: the input is textural requirements with each requirement statement expressed in a single sentence. Three example input requirements sentences with terms highlighted in bold blue font are shown in Figure 2, which will act as running examples in the rest of this section.
- **Terminology extraction**: this module is responsible for identifying continuous text fragments as requirements terms after the requirements sentences are syntactically processed. The output of terminology extraction module is a flat list of identified requirements terms that can be compared with ground truth, if existing, or manually investigated by human analysts to estimate the extraction performance.
- **Terms ranking**: this module implements our adapted TextRank algorithm to rank the relevant and more abstract terms at the top positions so as to reduce the human noise filtering efforts and aid the feature modeling procedure.
- **Feature modeling**: The ranked requirements term list output by the previous module combined with the original requirements statements could inform the analysts an overview of the problem domain in general. Furthermore, analysts can employ feature model or its variants to build high-level models for the system under consideration with top ranked terms denoting various model elements, e.g., software features, system context.

A. Requirements Terminology Extraction Based on Noun Phrase Identification

A noun phrase (NP) is a phrase that has a noun or pronoun as its head or performs the same grammatical function as a noun [13]. The most common NP in English are compounds (e.g., "pet owner") and adjective noun phrases (e.g., "smart home"). In NLP, automatic term extraction approaches typically make use of linguistic processors (e.g., part of speech tagging, n-gram sequences) to extract syntactically plausible noun phrases as terminological candidates [10].

NP has been proved to be accounting for 99% of the technical terms [14]. In RE, NP also plays a significant role in identifying important artifacts, such as software features [2; 15], abstractions [4], input/output variables [16], and various domain model elements [17]. Therefore, we focus only on NPs in the requirements sentences for extracting requirements terms, which include single-word nouns and multi-word noun phrases.

For identifying NPs, we rely on constituency parsing, which aims to extract a constituency-based parse tree from a sentence that represents its syntactic structure according to a phrase structure grammar. Compared with lexical analysis, e.g., POS tagging and n-gram sequences, syntactic parsing considers more contextual information and the whole sentence structure, making it have the potential to identify NPs more accurately. In addition, NLP community has made considerable strides in English constituency parsing recently[1]. There are several well known NLP tools implementing the state of the art

[1]http://nlpprogress.com/english/constituency_parsing.html

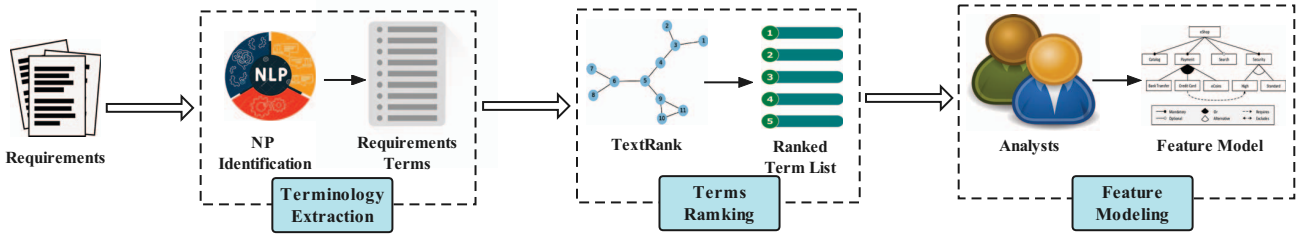

Fig. 1. Feature modeling oriented requirements terminology extraction and ranking framework

Req1: *As a* **pet owner**, *I want my* **smart home** *to let me know when the* **dog** *uses the* **doggy door**, *so that I can keep track of the* **pets whereabouts**.
Req2: *As a* **home owner**, *I want my* **smart home** *to turn on* **yard lights** *when* **motion** *is detected so that* **break-ins** *can be avoided.*
Req3: *As a* **home occupant**, *I want my* **smart home** *to learn my lighting habits and continue to use them when I am away so that* **intruders** *are deterred.*

Fig. 2. Three example input requirements

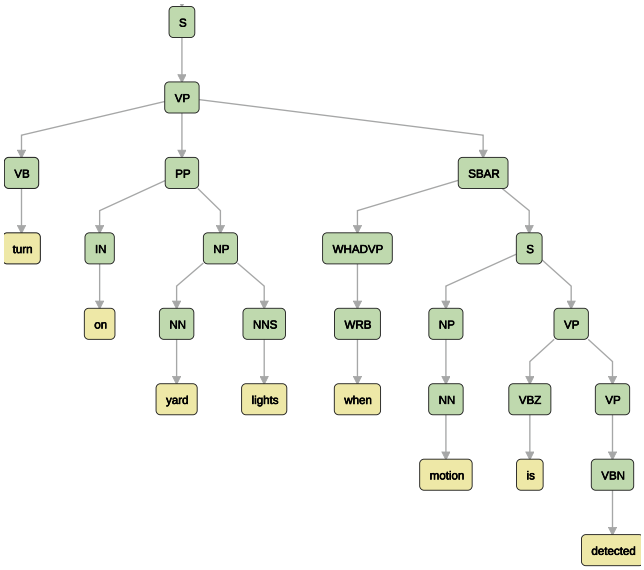

Fig. 3. Constituency parsing tree of Req2 fragment in Figure 2

constituency parsing algorithms, from which we select the widely used and free open-source library spaCy[2]. Figure 3 shows the constituency parsing tree of a fragment of Req2 in Figure 2 due to the limitation of space.

Algorithm 1 details the procedure of identifying NP combined with post-processing heuristic rules for extracting requirements terms. The main idea of Algorithm 1 is parsing requirements statements sentence-by-sentence syntactically and then selecting the continuous text segments as candidate requirements terms which are then processed by two types of heuristic rules to obtain the final requirements terms list.

Algorithm 1 Requirements Terms Extraction Based on NP identification

Input: requirements statements S_r
Output: A flat list of requirements terms L

Initialize $L = []$
for each $r_{raw} \in S_r$ **do**
  $r = Spell\_Check(r_{raw})$
  parsing tree $T_r = Constituency\_Parsing(r)$
  for each $NP \in T_r$ **do**
    if $word\_length(NP) < 2$ **then**
      continue
    else
      $w\_list = []$
      for each $word \in NP$ **do**
        if $word \in STOPLIST$ **then**
          continue
        else
          w_list *include* $word$
        end if
      end for
      if $length(w\_list) \neq 0$ **then**
        $term$ = concatenate $word \in w\_list$
        L *include* $Lemmatization(term)$
      end if
    end if
  end for
end for

Each requirements statement is checked with respect to spelling error and corrected (if has) with pyspellchecker tool[3] to ensure the constituency parsing performance. In order to reduce noisy terms, we apply two types of heuristic rules to filtering the candidate terms. The first rule is excluding NP containing less than 2 characters. For example, in Req2 of Figure 2, "I" is filtered while "motion" is remained. The second one is removing stopwords from the candidate NP and concatenating the remaining words in order. For instance, "a home owner" in Req2 of Figure 2 is transformed to "home

[2]https://spacy.io/

[3]https://github.com/barrust/pyspellchecker

owner" after applying the second rule. The $STOPLIST$ contains not only general English stopwords provided by NLTK[4] but also a few manually selected domain stopwords (e.g., numeral, punctuation, possessive wh-pronoun), which have been proved to be effective in cleaning textual requirements [2; 3]. After post-processing, lemmatization is applied to each word in the candidate requirements terms to reduce redundancy.

B. Requirements Terms Ranking with Adapted TextRank

Since the extraction result list is flat and unordered, human efforts are still needed to filter noisy terms and organize the terms to support certain RE tasks. In our research, we surmise that terms with a hierarchical structure may inform the hierarchy of a feature tree and the naming of elements when building feature models.

In this work, we mainly focus on feature modeling with the extracted requirements terms. In order to reduce human noise filtering efforts, we adopt the well-known text rank algorithm TextRank and its prevalent variants to rank more relevant terms in the top positions. We further adapt the TextRank with concept similarity to make more relevant and also more abstract (high-level) requirements terms come near the top of the ranked list so as to inform feature modeling.

TextRank is an unsupervised graph-based ranking model and has been widely used in various NLP applications for ranking text units. We adopt TextRank to rank requirements terms for two considerations. On the one hand, it has been proved to be effective in ranking text units in different granularity across various text domains. On the other hand, TextRank is unsupervised without manually labeled training data, which needs less human supervision and is beneficial to the generality of our proposed framework.

In the remaining of this subsection, we will briefly introduce the basic principle of TextRank followed by how we adapt it to ranking requirements terms according to the characteristics of feature models.

To enable the application of TextRank to natural language text, a graph has to be built for representing the text. Formally, let $G = (V, E)$ be a directed graph with the set of vertices V and set of edges E, where E is a subset of $V \times V$. Each vertex $V_i \in V$ denotes a text unit, e.g., a word or a sentence, and each edge connecting vertices V_i and V_j denotes some type of meaningful relationship between those two text units, e.g., distance, similarity. For a given vertex V_i, let $In(V_i)$ be the set of vertices that point to it, and let $Out(V_i)$ be the set of vertices that V_i points to. The score of vertex V_i is defined as follow:

$$S(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} S(V_j) \quad (1)$$

In equation (1), $S(V_i)$ denotes the score of node V_i, w_{ij} is the edge weight indicating the strength of the connection between two vertices V_i and V_j. d is a damping factor usually set to 0.85 [18], which has the role of integrating into the model the probability of jumping from a given vertex to another random vertex in the graph.

Complying with the steps of applying graph-based ranking algorithms to natural language texts suggested by [19], we adapt TextRank to rank the extracted requirements terms as Algorithm 2:

Algorithm 2 Requirements Terms Ranking Based on TextRank

Input: A flat list of extracted requirements terms L
Output: A ranked list of extracted requirements terms L_r

1: Initialize: vertices set $V = \{\}$, edges set $E = \{\}$, weights set $W = \{\}$
2: Build word set S_w for term list L
3: **for each** $word_i \in S_w$ **do**
4: V include $word_i$
5: **for each** $word_j \in S_w \backslash \{w_i\}$ **do**
6: edge weight $w_{ij} = Similarity(word_i, word_j)$
7: **if** $w_{ij} >$ threshold θ **then**
8: E include $edge_{ij}$
9: W include $weight_{ij}$
10: **end if**
11: **end for**
12: **end for**
13: Build undirected weighted word graph $G = (V, E, W)$
14: Optimize the node socre in equation (1) with random walk algorithm
15: **for each** $term \in L$ **do**
16: $Score(term) =$ average the scores of $word$s in $term$
17: **end for**
18: Return L_r via sorting $term \in L$ by $Score(term)$ in descending order

Algorithm 2 mainly involves four steps:

1) **Determine vertices (lines 3 - 4)**: words composing of requirements terms are identified as the basic text units and are added as vertices in the graph. Taking the extracted terms in Req2 of Figure 2 as an example, the word lemmas "home", "owner", "smart", "yard", "light", "motion", and "break-in" act as nodes in the graph.
2) **Draw edges between vertices (lines 5 - 13)**: semantic similarity is used to connect text units and draw edges between vertices in the graph. Edges are undirected, in which case the out-degree of a vertex is equal to the in-degree of the vertex. Similarity scores are used to indicate edge weights. For the semantic similarity, we employ pretrained embedding [20] based vector similarity and WordNet based concept similarity [21], which take advantage of a priori information from large-scale text corpora as well as lexical and knowledge bases. So far, a weighted and undirected word graph has been built.
3) **Optimize iteratively (line 14)**: starting from arbitrary values assigned to each node in the graph, the compu-

[4]https://www.nltk.org/

tation iterates until convergence using a random walk algorithm derived from PageRank [18]. Notice that the final values obtained after TextRank runs to completion are not affected by the choice of the initial value, only the number of iterations to convergence may be different [19].

4) **Score requirements terms (line 15 - 18)**: requirements terms are scored followed by being sorted in descending order with respect to their scores. The score of each requirements term is computed by averaging the scores of words in that term.

In the above procedure, step 2 and step 4 are our innovative designs to achieve our goal of making relevant and more abstract requirements terms rank in the top positions.

In what follows, we firstly explain the rationale of using semantic similarity as edge weights to make more relevant terms rank in top positions (step 2). Then we elaborate how WordNet based concept similarity can potentially make words denoting more abstract concepts obtain higher scores (step 2). Finally, we introduce the rationale of scoring multi-word requirements terms by averaging the words' scores (step 4).

In step 2, for the relation connecting word vertices, we employ word semantic similarity other than the commonly used co-occurrence [10]. The rationale behind this selection is that co-occurrence relation prefers scoring frequent words higher [19] while words with low frequency would warrant a precise definition for the requirements terms containing them [12]. Therefore, relying on co-occurrence or frequency may underestimate some important requirements terms with low socres caused by the infrequent but informative words in them, which may lower their ranking and thus need more human efforts to find them.

Furthermore, in step 2, we try WordNet-based conceptual semantic similarity to denote edge weights so as to make abstract requirements terms have a higher ranking than those specific ones. This edge weight design is mainly stemming from the fact that feature model is a hierarchical diagram that visually depicts the features of a solution in groups of increasing levels of detail.

WordNet[5] is a large lexical database of English. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are arranged into hierarchies. The most frequently encoded relation among synsets is the super-subordinate relation (i.e., IS-A relation), which links more general synsets to increasingly specific ones. WordNet based similarity methods usually rely on the structure of the semantic network with the shortest path length between concepts in the hierarchy as an important consideration [22].

Under the setting of using WordNet-based concept similarity to denote edge weights in TextRank, more abstract words tend to obtain higher scores due to their higher similarity with other concrete words and the scoring mechanism of TextRank. For example, in Figure 4, "person" acts like a hyperonymy of other

[5]https://wordnet.princeton.edu/

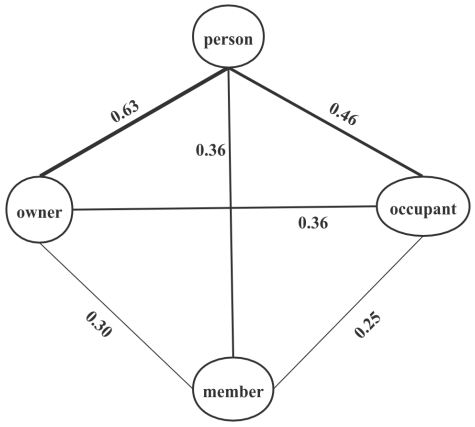

Fig. 4. Example edge weights between word vertices in TextRank

three more concrete related words and has links with higher weights denoted by WordNet-based similarity scores. According to the socring mechanism underlying TextRank [19], when one vertex links to another one, it is basically casting a vote for that other vertex. The link weight reflects the importance of that vote. The higher the number of important votes that are cast for a vertex, the higher the importance of the vertex. Therefore, "person" would obtain a higher score than the other three related words after the TextRank runs to completion. Besides indicating link importance, link weight could also be used to determine if a link should be established between two vertices by being compared with a pre-defined threshold, e.g., θ in Algorithm 2. If the threshold is set to 0.4, there will be only two edges left in the word graph in Figure 4.

In step 4, we score a term by averaging the scores of words in it as follow:

$$Score(T) = \frac{1}{|\{w|w \in T\}|} \sum_{w \in T} \frac{1}{r_w} \times S(w) \qquad (2)$$

where T is a requirements term, w is a word in T, $S(w)$ and r_w are the score and ranking of word w from TextRank seperately. When scoring the requirements terms, we consider the name characteristic of elements in feature models. As levels of details increase, the element nodes in feature models usually are attached with not only more specific but also probably longer names to describe the sub-features more precisely. For terms describing similar domain concepts, the longer requirements terms typically contain some modifiers and concrete words. Those words tend to have lower scores since they usually have fewer higher weights with other word vertices (i.e., fewer important votes from other vertices) in TextRank's word graph. Therefore, after averaging the scores of words in a term as equation (2), longer terms with more modifier and specific words tend to have lower scores. In particular, the word score $S(w)$ is weighted by the reciprocal of its ranking. This design aims to make similar terms with different abstraction levels appear in close positions of the ranking list, which is similar

to clustering. As the link weight threshold increases, the word graph might become a non-complete graph where some word nodes would become isolated vertices. Those isolated vertices would not have a ranking and would be assigned a small score (e.g., the reciprocal of the size of vocabulary, $\frac{1}{|S_{word}|}$) since they do not involve the TextRank's optimization. When scoring terms containing isolated words with formula (2), the scores of those isolated words would not be weighted. Consequently, the scores of similar terms with different abstraction levels would not differ too much.

For instance, in smart home domain, terms "control", "remote control", and "remote control opener" describe hierachical domain concepts, their scores and rankings would be descending, because the modifier and specific words "remote" and "opener" are very likely to obtain lower scores than the more abstract word "control" under the scoring mechanism of TextRank. After averaging word scores, the longer and more concrete term would have a lower score than the shorter and more abstract one. But words "remote" and "opener" might further be isolated nodes whose scores would not be weighted in equation (2), thus those three terms' scores and their rankings would not differ too much. Therefore, those three terms would appear in close positions in the ranking list.

III. Experimental Settings

In this section, we introduce the dataset, ground truth, baselines, evaluation measures, and two types of semantic similarity used in step 2 of the terms ranking module.

For dataset, we select smart home requirements [23; 24] created by 300 workers on Amazon Mechanical Turk platform for two reasons. Firstly, the dataset is publicly available[6]. Secondly, ground truth on requirements terms has been provided by previous work [25; 26] for a subset of the dataset (the first 100 requirements), which facilitates the evaluation and comparison between our method and other baselines. There are 2966 requirements in total with each requirement being expressed as a user story as in Figure 2.

For ground truth, [25] and [26] annotate different requirements terms for the same first 100 requirements of the above dataset. We compare our extraction method and YAKE with Hybrid_1 [25] and Hybrid_2 [26] on the corresponding ground truth separately. The main difference between those two ground truth is that [25] annotated requirements terms in the *role* part of user stories (i.e., the *As a ...,* part in Figure 2) while [26] did not. When evaluating different ranking methods by counting the number of relevant ones in top positions, we select the ground truth provided by [25] due to the fact that the *role* part usually indicates a system's various end users that are essential to understand the problem domain and the human-system interaction process.

For requirements terms extraction baselines, we select two types of methods as follows:

- **Hybrid_1**: an approach combining linguistic processing and statistical filtering for extracting and reducing requirements terms [25].
- **Hybrid_2**: an approach combining text chunking and semantic filtering based on word embeddings for extracting and reducing requirements terms [26].
- **YAKE**: an unsupervised general term extraction method which rests on statistical text features extracted from single documents [27]. We integrate the requirements from which terms are extracted as a single document by concatenation and use it as the input of this method.

The first two hybrid methods are both specially designed to extract requirements terms from large scale requirements. The third method is a state of the art for general terminology extraction.

For requirements terms ranking baselines, we select four terms ranking baselines derived from TextRank as follows:

- **TextRank-original**: the original TextRank which is proposed by [19] and was used to rank keywords or sentences constituting extractive summaries.
- **SingleRank**: a variant of TextRank which incorporates weights to edges with word co-occurrence statistics [28].
- **PositionRank**: a variant of TextRank which tries to capture frequent phrases considering the word-word co-occurrences and their corresponding position in the text [29]. As word position in the full document is an important consideration of this method, we randomly shuffle the test requirements statements to generate 100 documents as input and report the average results of the 100 experiments.
- **MultipartiteRank**: a variant of TextRank which represents term candidates and topics in a multipartite graph and exploits their mutually reinforcing relationship to improve candidate ranking [30].

The last three baseline methods are all extensions of TextRank and are used to rank keywords in information retrieval. They incorporate different types of information to TextRank, such as statistical, positional, word co-occurrence, and topical information.

For evaluation measures, we select the commonly used precision, recall, and F1-score to evaluate the performance of terms extraction. For quantitatively evaluating the ranking quality, we select Average Precision (AP) [31] which takes the ordering of a particular returned list of terms into account. Higher AP means that there are more relevant terms at the top of the returned list. The AP of a returned list is defined as follows:

$$AP = \frac{\sum_{r=1}^{|L|} P(r) \cdot rel(r)}{|L_R|} \tag{3}$$

where $|L|$ is the number of items in the list, $|L_R|$ is the number of relevant items, $P(r)$ is the precision when the returned list is treated as containing only its first r items, and $rel(r)$ equals 1 if the r_{th} item of the list is in the answer set and 0 otherwise.

[6]https://crowdre.github.io/murukannaiah-smarthome-requirements-dataset/

For semantic similarity, we select the pre-trained GloVe word embeddings[7] for vectorizing words and the cosine similarity for measuring word similarity. For computing the WordNet-based similarity, we select the wpath [21] method, which weights the shortest path length between concepts in WordNet using information content computed based on knowledge graphs and outperforms other concept similarity methods on well known word similarity datasets. For the implementation of the wpath method, we employ the sematch toolkit[8] provided by the original paper.

IV. Experimental Results and Analysis

To answer the research questions, we firstly report and discuss the experiment results of requirements terms extraction and ranking. Then we conduct qualitative analysis of the ranked term lists produced by different ranking methods. Finally, we build an illustrative feature model in the smart home domain to further demonstrate the potential usage of our ranking method in aiding feature modeling. All experimental materials (data and codes) are publicly available[9].

A. Requirements Terminology Extraction

RQ1: *How effectively does our noun phrase identification method extract requirements terms compared with baselines for requirements terminology extraction and general terminology extraction?*

Hybrid_1 [25] and Hybrid_2 [26] separately manually labeled 250 oracle requirements terms for the same first 100 requirements in the smart home dataset. Table I and Table II show the comparison results between our NP method and other baselines on these two different oracles. Main results drawn from the tables and explanations are summarized as follows:

- Our NP method performs better than YAKE consistently with respect to the two oracles. For either of the two oracles, YAKE yields the worst results in terms of the three accuracy measures. This undesirable result can be attributed to the fact that general-purpose NLP tools are not suitable for directly applying on RE tasks without being adapted according to the goal and characteristic of the underlying task [7; 12].
- Compared with other baselines, our method extracts the most number of relevant requirements terms, achieving the highest recalls, 82.8% and 80.40% on the two oracles respectively. Nearly 20 extra relevant terms are returned by our method than the two hybrid methods due to our fewer filtering heuristics. Our heuristics only filter the obviously inappropriate noun phrases that are too short or only include stopwords. YAKE employs n-grams as candidate terms without further lexical and syntactic analysis leading to the lowest recalls, less than 50% on the two oracles.
- The cost of our method's high recall is sacrificing partial precision compared with two hybrid baselines since our method returns the most number of terms (see the Total column). Though there is a little loss on oracle_1, larger precision loss occurs on oracle_2 where Hybrid_2 only returns 218 (<250) terms limiting its recall upper bound being 87.2% (218/250). However, for requirements terms extraction, recall is more important than precision. Finding the omitted terms from raw requirements documents needs much more efforts than excluding some noisy terms from the returned list.
- Taking both precision and recall into account, our method outperforms Hybrid_1 and YAKE while is comparable to Hybrid_2 in F1 score. Moreover, our method is more lightweight than those two hybrid methods. In addition to NLP processing, Hybrid_1 and Hybrid_2 also need domain-specific reference corpus (the whole requirements dataset including the unlabeled ones and Wikipedia crawling corpus separately) to construct complex statistical and semantic filters. In contrast, our extraction method only relies on constituency parsing and simple filtering heuristics, increasing its generality and portability potentially.

TABLE I
Requirements Terms Extraction Results on Oracle_1

Method	Total	Relevant	Precision	Recall	F1
NP (Ours)	288	**207**	71.88%	**82.80%**	**76.95%**
Hybrid_1	255	187	**73.33%**	74.80%	74.06%
YAKE	250	107	42.80%	42.80%	42.80%

TABLE II
Requirements Terms Extraction Results on Oracle_2

Method	Total	Relevant	Precision	Recall	F1
NP (Ours)	271	**201**	74.17%	**80.40%**	77.16%
Hybrid_2	218	183	**83.94%**	73.20%	**78.21%**
YAKE	250	102	40.80%	40.80%	40.80%

Answer to RQ1: *based on our initial experimentation, our method outperforms the general terminology extraction baselines in precision and recall. Compared with other requirements terms extraction baselines, our method achieves a higher recall without sacrificing much precision.*

B. Requirements Terms Ranking

RQ2: *How effectively does our adaption of TextRank rank more relevant terms in the top positions compared with other TextRank variants?*

With a flat list of extracted requirements terms at hand, we go a step further to rank them. Ranking relevant terms at the top of the list would substantially reduce the analysts' noise filtering efforts. We thus quantitatively evaluate the ranking quality of our adaption of TextRank and other TextRank variants.

The input of terms ranking methods is the returned requirements terms list by our NP identification method. To evaluate the ranking quality, we select oracle terms provided by [25] as ground truth (i.e., Oracle_1), which includes requirements terms in the *role* part of user stories. As there are totally

[7]https://nlp.stanford.edu/projects/glove/
[8]https://gsi-upm.github.io/sematch/
[9]https://figshare.com/s/9002ed3093971325a438

TABLE III
TERMS RANKING COMPARISON RESULTS BETWEEN ORIGINAL TEXTRANK AND OUR ADAPTED TEXTRANK

Method	Relevant	Recall	AP
TextRank-original	166	66.40%	65.75%
TextRank-embedding (Ours)	178	71.20%	59.13%
TextRank-wpath (Ours)	**185**	**74.00%**	**70.23%**

Note: AP denotes average precision defined by formula (3).

250 requirements terms in $Oracle_1$, we compute the recall and average precision for the first 250 terms returned by each rank method.

TextRank-embedding and TextRank-wpath denote our adaption of TextRank as shown in Algorithm 2. The only difference of these two methods is the edge weights in the word graph, where word embedding cosine similarity and WordNet based concept similarity are employed separately.

Table III shows the terms ranking comparison results between original TextRank and our two adapted TextRank methods. Generally, TextRank-wpath outperforms the original TextRank and TextRank-embedding in terms of recall and average precision. Specifically, TextRank-wpath achieves a recall of 74% (7.6% more than TextRank-original) and an average precision of 70% (4.25% more than TextRank-original) demonstrating that our adaption of TextRank is effective in ranking more relevant terms in the front positions. Though obtaining a recall increase (4.6%) compared with the original TextRank, TextRank-embedding loses more (6.75%) in average precision, causing its lower ranking quality.

It should be noticed that the results of our two methods in Table III are produced with complete graphs. In other words, there is an edge between every two distinct words in the graph built in Algorithm 2. As noted in Subsection II-B, pre-defined similarity threshold can be selected to determine whether drawing an edge between two words so as to build an incomplete graph. For example, if the threshold is set to 0.5, only the similarity score is more than 0.5 between two words, an edge is drawn to connect those two words with the similarity score being the edge weight.

For both of our two ranking methods, we experimentally select the best similarity threshold with the changing step as 0.1. The recall and AP of TextRank-embedding and TextRank-wpath along with different similarity thresholds are shown in Figure 5 and Figure 6 respectively. When selecting the best threshold, we prefer to ensure a high recall firstly and then pursue an AP as high as possible. As can be seen from Figure 5 and Figure 6, the recalls of both methods are relatively stable. With the threshold increasing, AP of both methods shows an upward trend in general. Consequently, 0.6 is selected for both TextRank-embedding and TextRank-wpath to balance the recall and AP.

After fine-tuning the similarity thresholds, we further compare our methods with other TextRank variants which use the best settings stated in the original papers. The comparison results are shown in Table IV. Our TextRank-wpath achieves the highest recall of 74% with the similarity threshold being 0.0

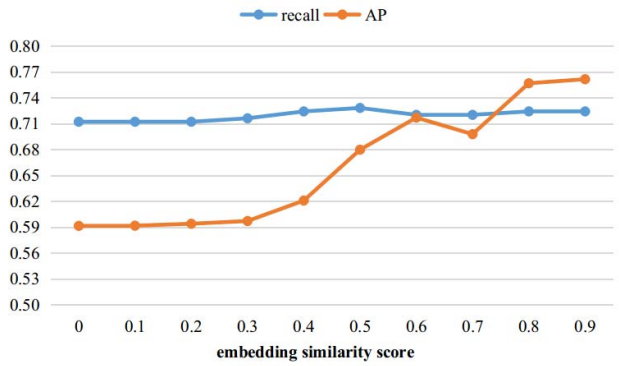

Fig. 5. Ranking quality of TextRank-embedding along with different similarity thresholds

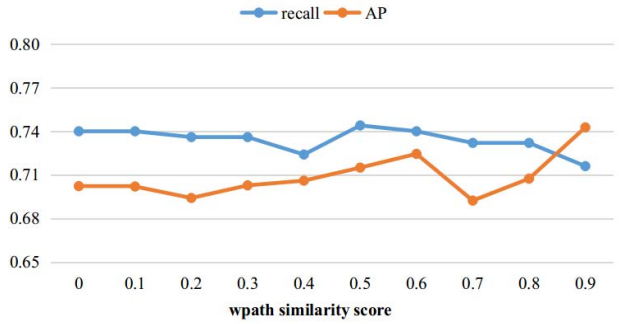

Fig. 6. Ranking quality of TextRank-wpath along with different similarity thresholds

or 0.6. It is worth noting that TextRank-wpath with complete graph setting (i.e., threshold = 0.0) is one of four methods achieving an AP higher than 0.7. This means that TextRank-wpath can produce competitive ranking results even without any fine-tuning. Compared with other TextRank variants, both of our methods achieve a relatively higher recall. The recalls of other variants are all less than 70% while both our two methods with or without fine-tuning obtain a recall higher than 70%. Among the four methods with AP exceeding 0.7, MultipartiteRank achieves the highest AP of 75.85% while the other three methods are all our adaptions of TextRank. Although MultipartiteRank gains 3.4% improvement than TextRank-wpath in AP, it loses more (4.8%) in recall.

Answer to RQ2: *our TextRank-wapth method outperforms most TextRank variants in ranking more relevant terms in the top positions except for MultipartiteRank. Our TextRank-embedding can also return competitive ranking results with a proper similarity threshold.*

C. Qualitative Analysis of Different Term Rank Lists

RQ3: *How to use the ranked term list produced by our adaptation of TextRank to help the construction of feature models?*

TABLE IV
TERMS RANKING COMPARISON RESULTS BETWEEN THREE TEXTRANK VARIANTS AND OUR ADAPTED TEXTRANK

Method	Relevant	Recall	AP
TextRank-original	166	66.40%	65.75%
TextRank-embedding$_{0.0}$ (Ours)	178	71.20%	59.13%
TextRank-wpath$_{0.0}$ (Ours)	**185**	**74.00%**	70.23%
TextRank-embedding$_{0.6}$ (Ours)	180	72.00%	71.68%
TextRank-wpath$_{0.6}$ (Ours)	**185**	**74.00%**	72.45%
MultipartiteRank	173	69.20%	**75.85%**
SingleRank	169	67.60%	67.68%
PositionRank	168	67.20%	66.48%

Note: (1) the number subscripts in our methods' names denote the similarity thresholds used in building word graphs; (2) AP denotes average precision defined by formula (3).

To qualitatively analyze ranking results, Table V lists the top 20 terms returned by different ranking methods with bold ones being in the Oracle. The results in Table V are basically consistent with those in Table IV when only considering the number of relevant terms.

When analyzing the content characteristics of those different term lists further, some meaningful insights can be drawn. The terms returned by TextRank-original and SingleRank are too homogeneous with most terms including the highly frequent words "smart" and "home". These results are attributed to the fact that co-occurrence and frequency are essential factors in constructing edge weights in those two methods. Most terms returned by TextRank-embedding and TextRank-wpath are short due to the average operation in the term scoring equation. For TextRank-embedding, the number of relevant terms in top 20 is the least which is in accord with its lower AP in Table IV. Though considering topic information when building the word graph, the terms returned by MultipartiteRank are dispersed without obvious structure.

In contrast, for similar domain concepts, TextRank-wpath could rank more abstract terms in the front of those more concrete ones, which is similar to the hierarchy structure of feature models. As can be seen in the TextRank-wpath column of Table V, the top 4 terms summarize several fundamental elements of the smart home requirements. The first one is system user with different roles, such as *home owner*, *pet owner*, *music fan* etc. The term "person" is a generalization of those roles in concept. As most functions of smart home focus on intelligent control via various smart devices, e.g., *smart window cleaner to clean windows automatically*, *smart coffee machine controllable from mobile to prepare coffee in the morning*. The term "control" summarizes the key point of various smart features while the term "device" generalizes various specific tools implementing those smart features. More specific terms related to the above ones are all ranked behind, e.g., "health conscious person", "smart device", "remote control", "remote control opener". The more concrete term "room" is also ranked behind the more general one "place". Furthermore, most of the relevant terms appearing in the TextRank-original column and SingleRank column are more specific instances of "person" and "smart device". TextRank-wpath does not generate very specific terms in the top 20 list due to its application of conceptual similarity and the average operation in the term scoring equation.

In summary, the qualitative comparison of different term rank lists shows that the content characteristics stem from different settings of ranking methods. Requirements analysts could pick the suitable ranking method in line with the task at hand. For instance, the result of MultipartiteRank could be further clustered for building requirements glossary due to its higher average precision. The result of TextRank-wpath could provide some useful insights on feature modeling due to its implicit hierarchy structure.

For further exploring the practical usefulness of the TextRank-wpath ranked list in feature modeling, we build an illustrative feature model for the smart home domain.

Specifically, we apply the extended context-aware feature modeling (eCFM) [32] to model the smart home domain. Because a smart home usually employs dynamically adaptable software (DAS) that uses sensor devices to identify context changes in the home and can activate or deactivate home automation features such as temperature control. eCFM is chosen for its ability to create a correct and expressive model for developing DAS.

The third author is responsible for building the eCFM of smart home application via analyzing the first 100 requirements and being provided with the ranked term list returned by TextRank-wpath. The third author is a postgraduate majoring in software engineering and not involved in the terms extraction and ranking work. After feature modeling, we interview the third author and query if the provided term list was helpful in feature modeling and how it facilitated the process. The feedback is that the top ranked terms gave some insights on determining the high-level elements of the feature diagram and the names of hierarchical features could be easily selected from the ranked list.

Figure 7 displays the eCFM built for smart home application, where the highlighted nodes are not present in the top 250 term list returned by TextRank-wpath. There are five main parts in this eCFM: functional requirements (left part), variability information specified as feature relationships (e.g., mandatory, optional), context information (right part), adaptation rules specified as dependencies among features and context (e.g., require, exclude), and users interacting with the system and context. As can be seen from Figure 7, functional features, context, and system users are all expressed in a hierarchical tree structure with increasing detailed description from roots to leaf nodes, which is similar to the implicit hierarchy structure contained in the TextRank-wpath result term list.

Answer to RQ3: *the term list of our TextRank-wpath has an implicit hierarchy structure that is similar to the hierarchy structure of feature models and could potentially facilitate the name selection of hierarchical elements during feature modeling.*

TABLE V
TOP 20 REQUIREMENTS TERMS RETURNED BY DIFFERENT RANKING ALGORITHMS

Rank	TextRank-original	TextRank-embedding	TextRank-wpath	MultipartiteRank	SingleRank
1	**smart home**	need	**person**	**smart home**	**smart home**
2	smart carbon monoxide detector	**heating**	**activity**	**parent**	**home occupant**
3	**smart window cleaner**	thing	**device**	**pet owner**	**home owner**
4	**smart breakfast appliance**	**kitchen item**	**control**	**night**	full home sport experience
5	**smart touchscreen tv**	**smartphone**	**health conscious person**	**room**	home door
6	**smart coffee machine**	**heater**	area	**window fan**	**home resident**
7	full home sport experience	**room**	case	**home**	home turn
8	integrate smart electronic	**day**	**work**	**home owner**	**home**
9	**smart appliance**	**sound**	**smart device**	**door**	smart carbon monoxide detector
10	**home owner**	well overall health	issue	**plant**	**smart window cleaner**
11	**smart device**	**kid**	young child movement	**music**	**smart breakfast appliance**
12	**smart feature**	specific sound	**key**	long day	**smart touchscreen tv**
13	smart thermostat	**oven**	**remote control**	**video feed**	**smart coffee machine**
14	smart shower	certain light	**picture**	**time**	**integrate smart electronic**
15	**smart lock**	**child**	**family**	**mood**	**smart device**
16	etc . home resident	well light house	**remote control opener**	**tv fan**	**smart appliance**
17	home door	**door**	**place**	**person**	smart shower
18	**home occupant**	long day	**room**	**comfort**	smart thermostat
19	**automate music system**	good way	add measure	**alarm**	**smart lock**
20	**music fan**	**house**	**light**	**work**	**smart feature**

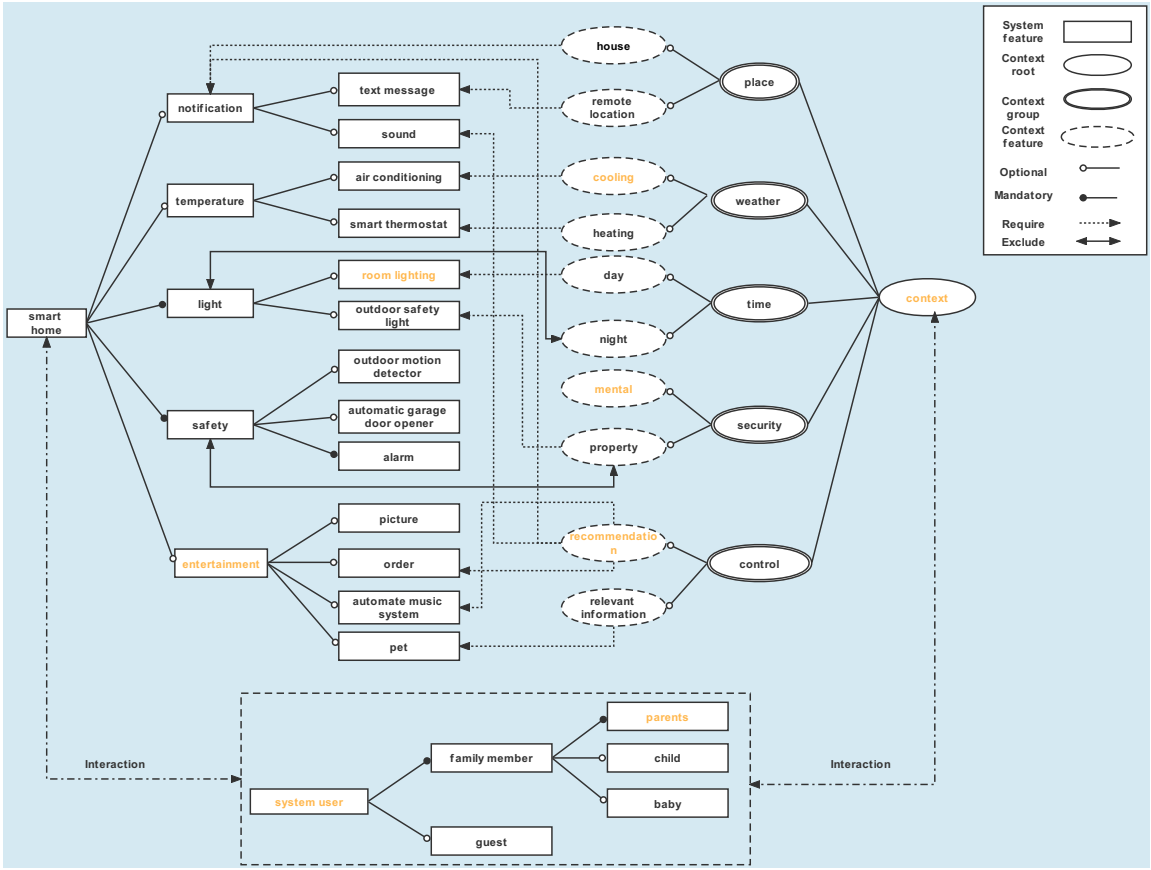

Fig. 7. The eCFM for smart home application

V. Threats to Validity

A threat to construct validity is that there are two requirements terms ground truth sets provided by previous work under different considerations. To reduce the influence on evaluation, we compare our extraction method with other baselines on the two different ground truth separately. Another threat is that our assessment of the helpfulness of ranked terms on feature modeling is mainly qualitative with an illustrative feature model manually built on a relatively small set of crowd-elicited requirements. The crowd-elicited user stories are not coming from real industrial settings, which need to be further refined for creating a more practical feature model. Those intrinsic characteristics of the experiment data might have an impact on the reliability of our conclusions.

We believe the internal validity is high in that the factors potentially affecting the performance of term extraction and ranking are under our direct control. The only hyper parameter is the similarity threshold used to build the word graph and its influence on ranking quality has been analyzed in detail in Section IV-B. Another factor worth noting is that we only use the first 100 requirements other than all to model the smart home domain. The requirements in the crowd RE dataset are independent among each other and the first 100 requirements cover the four main application domains of the dataset (i.e., energy, entertainment, health, and safety). Therefore, the feature model built based on this requirements subset can be reasonably used to investigate the usefulness of our ranked term list.

A threat to external validity is that our evaluation results may not generalize to other subject systems or other domains. It is valuable to explore more software applications beyond smart home in real industry context.

VI. Related Work

NLP techniques and ML algorithms have been widely used in a series of RE activities through mining various software repositories and artifacts [7; 33]. Our work in this paper is related to research topics on both NLP and RE.

The NLP and ML methods used in our framework are firstly related to the terminology extraction research in the NLP community. Inspired by the two-phase pipeline (extraction and ranking) commonly adopted by unsupervised keywords identification methods [10], we design our requirements terminology extraction and ranking modules based on the characteristics of requirements statements. Specifically, we employ constituency parsing based NP identification for terminology extraction motivated by early exploration on linguistic properties of technical terminology [14; 34]. Because TextRank is unsupervised and has been widely proved to be effective in ranking text [35–38], we adapt it to rank requirements terms for supporting problem domain understanding and feature modeling.

For RE task, we select domain feature modeling inspired by [9] where statistical language engineering techniques are employed to analyze ethnographic fieldnotes for supporting early domain understanding with class diagrams. Much research effort has been made to provide automated [39], visual [40; 41], and collaborative [42] support for building various RE models, e.g., variability models [43; 44] and software product line feature models [45; 46]. What differs our work from them is that our extraction and ranking framework can also be easily tuned to support other settings other than feature modeling as discussed in Section IV-C.

In contrast to existing requirements term extraction work by [25] and [26], our work not only extracts a flat list of terms from requirements documents but also goes a step further to rank terms for reducing noise filtering efforts and aiding analysts in feature modeling. Besides, our extraction module does not rely on large scale domain specific corpora like [25] and [26], making it more general and lightweight. What contrasts our work from the work by [12] is that we focus on problem domain understanding and feature modeling instead of constructing a formal glossary that includes definitions, synonyms, related terms, and example usages.

VII. Conclusion

In this paper, we have proposed an automated requirements terminology extraction and ranking framework built on syntactic analysis and an unsupervised graph-based ranking algorithm. Empirical results demonstrate the effectiveness of our automated framework, and an illustrative example on the smart home domain shows the usefulness of our ranked term list in assisting feature modeling.

There are several valuable avenues to extend our work in the future. Hierarchical agglomerative clustering could be integrated into TextRank to generate a more explicit hierarchical structure of terms. Other concept similarity, e.g., knowledge graph based similarity could be explored for improving the terms ranking quality. Finally, other NLP techniques like semantic analysis, relation extraction could be employed to identify the relationships and constraints in feature models.

Acknowledgment

The authors would like to thank the anonymous reviewers for their careful work and thoughtful suggestions.

References

[1] Nan Niu and Steve Easterbrook. So, you think you know others' goals? a repertory grid study. *IEEE software*, 24 (2):53–61, 2007.

[2] Jianzhang Zhang, Yinglin Wang, and Tian Xie. Software feature refinement prioritization based on online user review mining. *Information and Software Technology*, 108:30–34, 2019.

[3] Timo Johann, Christoph Stanik, Walid Maalej, et al. Safe: A simple approach for feature extraction from app descriptions and app reviews. In *2017 IEEE 25th international requirements engineering conference (RE)*, pages 21–30. IEEE, 2017.

[4] Zedong Peng, Prachi Rathod, Nan Niu, Tanmay Bhowmik, Hui Liu, Lin Shi, and Zhi Jin. Environment-driven abstraction identification for requirements-based testing. In *2021 IEEE 29th International Requirements*

Engineering Conference (RE), pages 245–256. IEEE, 2021.

[5] Negar Hariri, Carlos Castro-Herrera, Mehdi Mirakhorli, Jane Cleland-Huang, and Bamshad Mobasher. Supporting domain analysis through mining and recommending features from online product listings. *IEEE Transactions on Software Engineering*, 39(12):1736–1752, 2013.

[6] Alessio Ferrari, Felice Dell'Orletta, Andrea Esuli, Vincenzo Gervasi, and Stefania Gnesi. Natural language requirements processing: A 4d vision. *IEEE Softw.*, 34 (6):28–35, 2017.

[7] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J Letsholo, Muideen A Ajagbe, Erol-Valeriu Chioasca, and Riza T Batista-Navarro. Natural language processing for requirements engineering: A systematic mapping study. *ACM Computing Surveys (CSUR)*, 54(3):1–41, 2021.

[8] Zahra Shakeri Hossein Abad, Vincenzo Gervasi, Didar Zowghi, and Behrouz H Far. Supporting analysts by dynamic extraction and classification of requirements-related knowledge. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 442–453. IEEE, 2019.

[9] Peter Sawyer, Paul Rayson, and Ken Cosh. Shallow knowledge as an aid to deep understanding in early phase requirements engineering. *IEEE Transactions on Software Engineering*, 31(11):969–981, 2005.

[10] Eirini Papagiannopoulou and Grigorios Tsoumakas. A review of keyphrase extraction. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(2): e1339, 2020.

[11] Iso/iec/ieee international standard - systems and software engineering – life cycle processes –requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*, pages 1–94, 2011. doi: 10.1109/IEEESTD.2011.6146379.

[12] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Automated extraction and clustering of requirements glossary terms. *IEEE Transactions on Software Engineering*, 43(10):918–945, 2017.

[13] Noam Chomsky. *The minimalist program*. MIT press, 2014.

[14] John S Justeson and Slava M Katz. Technical terminology: some linguistic properties and an algorithm for identification in text. *Natural language engineering*, 1 (1):9–27, 1995.

[15] Nan Niu, Juha Savolainen, Zhendong Niu, Mingzhou Jin, and Jing-Ru C Cheng. A systems approach to product line requirements reuse. *IEEE Systems Journal*, 8(3): 827–836, 2013.

[16] Zedong Peng, Xuanyi Lin, Sreelekhaa Nagamalli Santhoshkumar, Nan Niu, and Upulee Kanewala. Learning i/o variables from scientific software's user manuals. In *In 2022 22nd International Conference on Computational Science (ICCS)*, (to appear).

[17] Nili Itzik, Iris Reinhartz-Berger, and Yair Wand. Variability analysis of requirements: Considering behavioral differences and reflecting stakeholders' perspectives. *IEEE Transactions on Software Engineering*, 42(7):687–706, 2015.

[18] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[19] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411, 2004.

[20] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai, and Xuanjing Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897, 2020.

[21] Ganggao Zhu and Carlos A Iglesias. Computing semantic similarity of concepts in knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 29 (1):72–85, 2016.

[22] Lingling Meng, Runqing Huang, and Junzhong Gu. A review of semantic similarity measures in wordnet. *International Journal of Hybrid Information Technology*, 6 (1):1–12, 2013.

[23] Pradeep K Murukannaiah, Nirav Ajmeri, and Munindar P Singh. Acquiring creative requirements from the crowd: Understanding the influences of personality and creative potential in crowd re. In *2016 IEEE 24th International Requirements Engineering Conference (RE)*, pages 176–185. IEEE, 2016.

[24] Pradeep K Murukannaiah, Nirav Ajmeri, and Munindar P Singh. Toward automating crowd re. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 512–515. IEEE, 2017.

[25] Tim Gemkow, Miro Conzelmann, Kerstin Hartig, and Andreas Vogelsang. Automatic glossary term extraction from large-scale requirements specifications. In *2018 IEEE 26th International Requirements Engineering Conference (RE)*, pages 412–417. IEEE, 2018.

[26] Siba Mishra and Arpit Sharma. Automatic word embeddings-based glossary term extraction from large-sized software requirements. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 203–218. Springer, 2020.

[27] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Jorge, Célia Nunes, and Adam Jatowt. Yake! keyword extraction from single documents using multiple local features. *Information Sciences*, 509:257–289, 2020.

[28] Xiaojun Wan and Jianguo Xiao. Single document keyphrase extraction using neighborhood knowledge. In *AAAI*, volume 8, pages 855–860, 2008.

[29] Corina Florescu and Cornelia Caragea. Positionrank: An unsupervised approach to keyphrase extraction from scholarly documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1105–1115, 2017.

[30] Florian Boudin. Unsupervised keyphrase extraction with multipartite graphs. In *Proceedings of the 2018 Conference of the North American Chapter of the Association*

for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), pages 667–672, 2018.

[31] Ling Liu and M Tamer Özsu. *Encyclopedia of database systems*, volume 6. Springer, 2009.

[32] Ismayle de Sousa Santos, Magno Lua de Jesus Souza, Michelle Larissa Luciano Carvalho, Thalisson Alves Oliveira, Eduardo Santana de Almeida, and Rossana Maria de Castro Andrade. Dynamically adaptable software is all about modeling contextual variability and avoiding failures. *IEEE Software*, 34(6): 72–77, 2017.

[33] M Vidoni. A systematic process for mining software repositories: Results from a systematic literature review. *Information and Software Technology*, page 106791, 2021.

[34] Ken Barker and Nadia Cornacchia. Using noun phrase heads to extract document keyphrases. In *conference of the canadian society for computational studies of intelligence*, pages 40–52. Springer, 2000.

[35] Ambedkar Kanapala, Sukomal Pal, and Rajendra Pamula. Text summarization from legal documents: a survey. *Artificial Intelligence Review*, 51(3):371–402, 2019.

[36] Muhammad Abulaish, Md Aslam Parwez, et al. Disease: A biomedical text analytics system for disease symptom extraction and characterization. *Journal of Biomedical Informatics*, 100:103324, 2019.

[37] Zhaoxin Huang and Zhenping Xie. A patent keywords extraction method using textrank model with prior public knowledge. *Complex & Intelligent Systems*, pages 1–12, 2021.

[38] Wafaa S El-Kassas, Cherif R Salama, Ahmed A Rafea, and Hoda K Mohamed. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*, 165:113679, 2021.

[39] Rijul Saini, Gunter Mussbacher, Jin LC Guo, and Jörg Kienzle. Domobot: An ai-empowered bot for automated and interactive domain modelling. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 595–599. IEEE, 2021.

[40] Nan Niu, Sandeep Reddivari, and Zhangji Chen. Keeping requirements on track via visual analytics. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 205–214. IEEE, 2013.

[41] Garm Lucassen, Marcel Robeer, Fabiano Dalpiaz, Jan Martijn EM Van Der Werf, and Sjaak Brinkkemper. Extracting conceptual models from user stories with visual narrator. *Requirements Engineering*, 22(3):339–358, 2017.

[42] Fatma Başak Aydemir and Fabiano Dalpiaz. Supporting collaborative modeling via natural language processing. In *International Conference on Conceptual Modeling*, pages 223–238. Springer, 2020.

[43] Nan Niu and Steve Easterbrook. Extracting and modeling product line functional requirements. In *2008 16th IEEE International Requirements Engineering Conference*, pages 155–164. IEEE, 2008.

[44] Eleonora Arganese, Alessandro Fantechi, Stefania Gnesi, and Laura Semini. Nuts and bolts of extracting variability models from natural language requirements documents. In *Integrating Research and Practice in Software Engineering*, pages 125–143. Springer, 2020.

[45] Jianmei Guo, Yinglin Wang, Zheying Zhang, Jyrki Nummenmaa, and Nan Niu. Model-driven approach to developing domain functional requirements in software product lines. *IET software*, 6(4):391–401, 2012.

[46] Anjali Sree-Kumar, Elena Planas, and Robert Clarisó. Extracting software product line feature models from natural language specifications. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*, pages 43–53, 2018.