OpenReq-DD: A Requirements Dependency Detection Tool

Quim Motger
ESSI Dept.
jmotger@essi.upc.edu

Ricard Borrull
ESSI Dept.
rborrull@essi.upc.edu

Cristina Palomares
ESSI Dept.
cpalomares@essi.upc.edu

Jordi Marco
CS Dept.
jmarco@cs.upc.edu

Universitat Politecnica de Catalunya

Abstract

Requirements Engineering (RE) is one of the most critical phases in software development. Analyzing requirements data is a laborious task performed by expert stakeholders using manual processes, as there are no standard automatic tools to handle this issue in a more efficient way. The purpose of this paper is to summarize the approach of the OpenReq-DD dependency detection tool developed at the OpenReq project, which allows an automatic requirement dependency detection approach. The core of this proposal is based on an ontology which defines dependency relations between specific terminologies related to the domain of the requirements. Using this information, it is possible to apply Natural Language Processing techniques to extract meaning from these requirements and relations, and Machine Learning techniques to apply conceptual clustering, with the major purpose of classifying these requirements into the defined ontology.

1 Introduction

One of the most critical branches in software engineering is *Requirements Engineering* (RE) [1]. There are many different problems related to this area, one of them being *requirements traceability* (RT). This is known as the "ability to describe and follow the life of a requirement, in both forwards and backwards direction" [2]. Traceability allows to identify dependencies between these relations, and how changes in the requirements may affect others that are related to them. Identifying dependencies between requirements, usually specified in Natural Language (NL) is considered a difficult, expensive and long process [3] [4].

In order to provide an automated solution for the dependency detection step, we introduce OpenReq-DD, a requirements Dependency-Detection tool addressed to apply NL and Machine Learning (ML) techniques to analyze requirements and extract dependencies between them. This tool has been developed inside OpenReq [5], an EU Horizon 2020 project that aims to provide advanced innovative tools for community-driven RE.

In this context we present a use case based on data provided by Siemens (Austria), one of the industrial partners of OpenReq. This data is a set of several documents called *Request for Proposals* (RFP) in the railway systems domain, which comprise NL requirements and can be several hundred pages long. Furthermore, stakeholders do not keep track of current dependencies. Therefore, it would be a large task to extract these dependencies in a manual way. For a better comprehension of the tool action, an example of these requirements is used for the description of the steps carried out by the tool.

2 OpenReq-DD approach

OpenReq-DD architecture is composed by a RESTful service as the main component, which exposes an API to provide the required data and perform the dependency detection algorithm. This is intended to match a microservice architecture and define an isolated, decoupled component that can be used in different contexts. This component exports and integrates as internal dependencies all required toolkits and frameworks for the different algorithm steps (see Fig. 1). For demonstrations, a simple GUI is provided (see Sec. 3). The OpenReq-DD project is available at GitHub (https://github.com/OpenReqEU/dependency-detection), including a README file with all required information to deploy and run the tool.

Figure 1 shows the sequence of steps that OpenReq-DD performs to extract requirement dependencies. In the following, we describe the needed input data (Sec. 2.1), and a a brief description of each stage (Sec. 2.2).

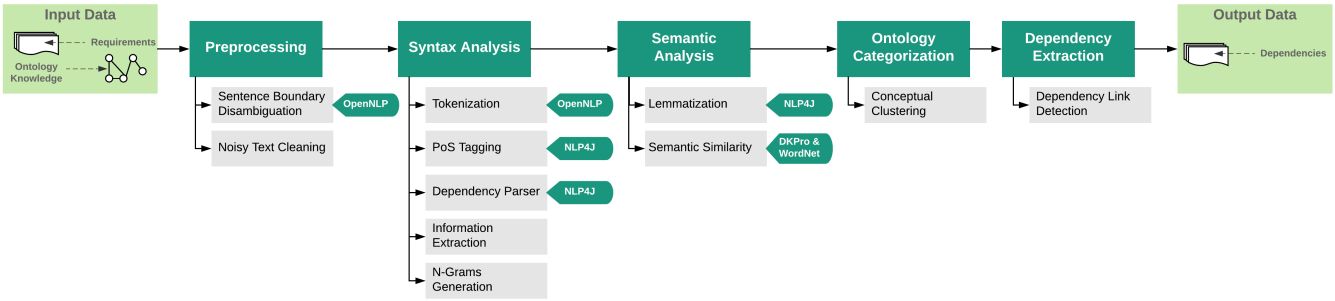


Figure 1: Dependency detection overview

2.1 I/O Data

The dependency detection algorithm designed and developed in OpenReq-DD requires two types of data to perform the dependency extraction. The format of this data has been defined and discussed with the stakeholders of the OpenReq project.

- **Requirements list**. A list of requirements from which to extract dependencies is provided. The JSON Schema used for input and output response is available at https://goo.gl/Gx8vpJ
- **Ontology**. The ontology provides the tool the required knowledge about the general patterns and dependency types that are potential dependencies between requirements. This is the result of an study performed by stakeholders of the project. The ontology knowledge is structured in a dependency relations tree, where each node is a topic and each edge a dependency relation type (see Fig. 2). An example of this ontology is available at https://goo.gl/Hx6GS2

The output of the RESTful service is a JSON response using the same format that the input data, but with the set of detected dependencies included.

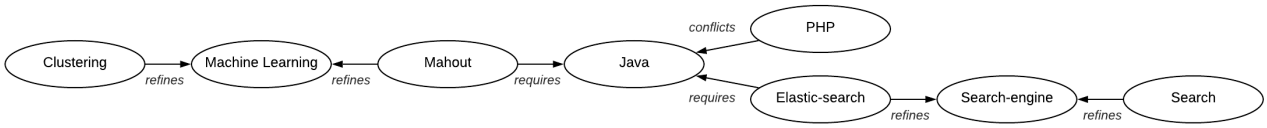


Figure 2: Ontology structure example

2.2 Dependency detection process

Given the input data, OpenReq-DD initiates the dependency extraction by following the next steps (which are transparent to the user).

2.2.1 Preprocessing

In order to reduce deficiencies in the data set, two preprocessing methods are applied to the requirements.

- **Sentence Boundary Disambiguation** (SBD). Sentence detection is applied to extract isolated sentences from each requirement, by deciding where are the beginning and the end of each sentence. The Apache toolkit OpenNLP [6] is used for this purpose.

- **Noisy Text Cleaning**. After SBD, a total of 14 rules are applied to clean the text of each sentence. The cleaning includes, between others: removal of character, numeric and roman numerals list pointers; removal of acronyms that may appear at the beginning of a requirement; removal of escape sequence characters; addition of white spaces to prevent PoS tagger faults (e.g., between parenthesis or question marks).

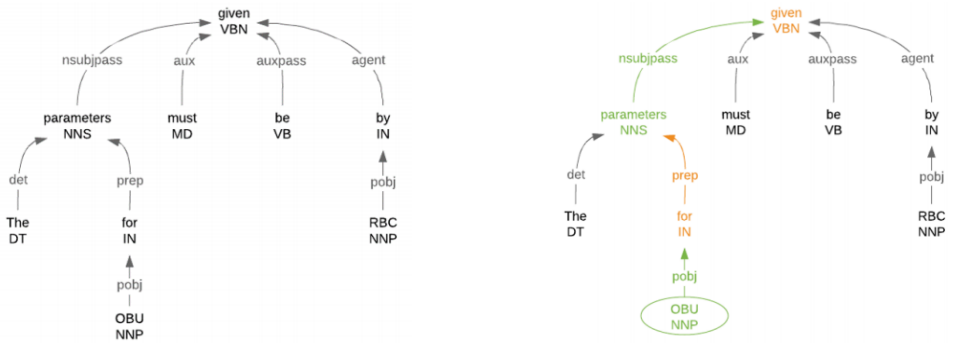
2.2.2 Syntax Analysis

Given the preprocessed requirements, a syntax analysis is executed in order to extract words that can be potential candidates of a match with concepts of the ontology.

- **Tokenization**. The input sentence is split into single words using the OpenNLP toolkit.
  Sentence: "The parameters for OBU must be given by RBC."
  Tokens: "The", "parameters", "for", "OBU", "must", "be", "given", "by", "RBC", "."
- **PoS tagging**. Each token of the sentence is marked with a part-of-speech tag using the NLP4J toolkit [7].

The	parameters	for	OBU	must	be	given	by	RBC	.
DT	*NNS*	*IN*	*NP*	*MD*	*VB*	*VBN*	*IN*	*NNP*	*.*

- **Dependency Parser**. Using the NLP4J toolkit, a dependency tree is generated where each node is a token of the input sentence and edges are the relations between parent words and child words.
- **Information Extraction**. In this step, the words to categorize each requirement into the ontology are detected. For doing so, patterns that take into account the position in the sentence and PoS tag of the word are used. To extract these patterns, a study was performed with existing datasets: the most representative words for each requirement were detected and afterwards patterns were extracted according to the position in the sentences of these representative words.



(a) Dependency parser result

(b) Deepest path for term extraction

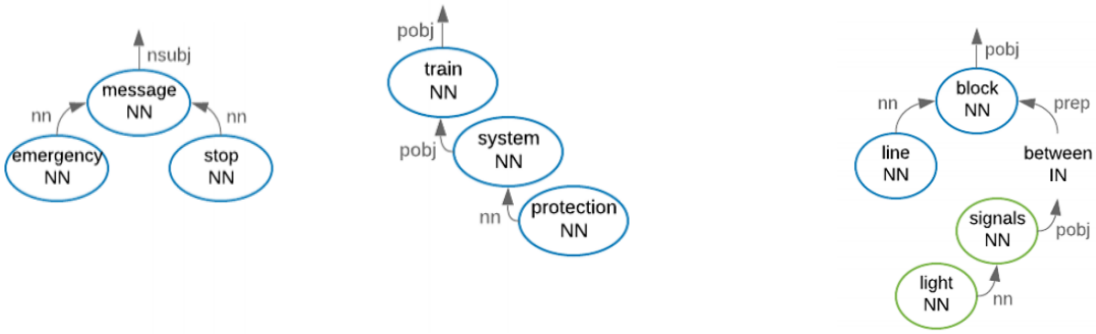
Figure 3: Dependency tree

- **N-Grams Generation**. The matched patterns inside the dependency tree are analyzed in order to generate n-grams of nodes directly connected within the tree that are composed by a set of keywords encapsulating a big concept, a general idea superior to the individual meaning of each keyword. See Fig. 4 for an example.

2.2.3 Semantic Analysis

Semantic analysis is the process that interprets the language meaning (i.e., the topic concept) of the whole text. The main goal is to obtain the meaning of each word of the n-gram, and join them to get a unique meaning that can be matched with the concepts of the ontology.

- **Lemmatization**. The morphological analyzer included in the NLP4J framework is used to apply several rules (based on a large dictionary and several advanced heuristics) to extract the lemmas of each token. These lemmas allow us to compare different words with the same lexeme.
- **Semantic similarity**. The DKPro-Similarity framework [8] is used as a word pair similarity detection in order to improve the lemmatization process, by identifying those tokens with a high similarity score that are not identified as part of the same lexeme. This step is potentially interesting for synonyms and similar meanings, which are analyzed using the lexical database WordNet [9].



(a) A parent with two direct term children to merge in a unique set of words.

(b) A parent with one direct term child that has its own set of words to be merged in a unique set of words.

(c) A parent with one direct term child to merge in a set of words, and another separated set of words from the other not relevant child.

Figure 4: N-gram generation

2.2.4 Ontology Categorization

OpenReq-DD uses conceptual clustering to classify requirements into the different concepts of the input ontology by similar features. For each n-gram obtained in the semantic analysis, the following rules are applied.

- First, find equal matches between words combinations of the n-gram and the n-grams of the ontology.
- If there is no match, find equal matches between combinations of the extracted lemmas from the n-gram and the extracted lemmas of the input ontology.
- If there is no match, calculate the semantic relatedness between lemmas from the n-gram and the input ontology. The requirement is matched if the value is greater than a provided threshold.
- If none of the previous conditions are satisfied, the requirement is discarded as an individual in the ontology.

The result of this step is the ontology filled by requirement individuals.

2.2.5 Dependency Extraction

Finally, each pair of classes in the ontology that are linked with a dependency relation are analyzed extracting their instances (i.e., the requirements individuals) to find the existing dependencies.

3 Demo plan

In this section we provide details about a demo plan of the dependency detection analysis using OpenReq-DD.

3.1 Environment configuration

For a demo execution, it is necessary to run both the OpenReq-DD web service and the Java GUI application.

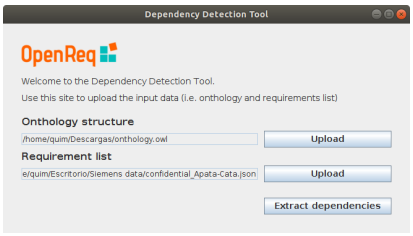
The GUI component is a presentation layer that simplifies the communication with the Dependency-Detection RESTful service. It allows the user to execute a dependency detection analysis in a simplified way, decoupling HTTP communication requirements and output data interpretation on the client side.

As input data we need a list of requirements and an ontology. Examples of both files are referenced in Sec. 2.1.

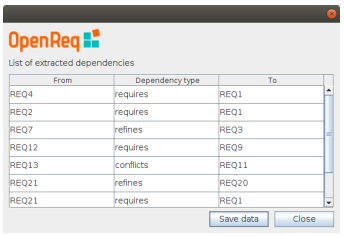
3.2 Dependency detection execution

Figures 5a and 5b show the main views of the OpenReq-DD GUI application. The user is asked to upload two files: the ontology structure file (*.owl file) and the requirements list file (*.json file). Once these files have been uploaded, the user can initiate the dependency analysis by clicking on the "Extract dependencies" button.

After this interaction, the whole process of dependency detection starts: the *back-end* of the tool applies the steps explained in Sec. 2.2 . The RESTful service returns a JSON format response including the requirements and the list of dependencies detected. Through the GUI component, the list of dependencies is shown in a table including, for each dependency, three items: the source of the dependency; the dependency type; and the target of the dependency.



(a) Input Data View



(b) Results View

Figure 5: OpenReq-DD GUI

3.3 Siemens use case results

We use the 6 RFP documents of the Siemens' use case to validate OpenReq-DD and the achievement of objectives in terms of efficiency and efficacy of the requirements' dependencies detection.

Tables 1 and 2 present a summary of both the automatic generated results of the OpenReq-DD tool and the stakeholder's manual validation. On the left side, we introduce the RFPs used for the Siemens use case, the number of total requirements of each RFP, and the number of dependencies extracted by our tool. These results were manually analyzed by stakeholders' experts, using three statistic measures: the precision of true positive detected dependencies, the precision with a refinement possibility (Precision-R) of true positive detected dependencies, and the imprecision of the detected dependencies, which is related to false positives outcomes. The results are good, but there is still room for improvement. We consider as future work exploring dependencies beyond semantic similarity, using other natural language criteria, and the extraction of relevant words from requirements using ML techniques (e.g., topic modelling).

Document	Requirements	Dependencies
RPF1	1209	4453
RPF2	6880	12242
RPF3	1209	182
RPF4	1209	167
RPF5	1209	39
RPF6	1209	1261
Total	**14714**	**18344**

Table 1: Detected dependencies within provided RFPs

Precision	89.2%
Precision-R	7.7%
Imprecision	3.1%

Table 2: Dataset results

Acknowledgments

The work presented in this paper has been conducted within the scope of the Horizon 2020 project OpenReq, which is supported by the European Union under the Grant Nr. 732463.

References

[1] K. Pohl, The three dimensions of requirements engineering: A framework and its applications, *Information Systems*, 19(3):243–258, 1994.

[2] O. C. Z. Gotel and C. W. Finkelstein, An analysis of the requirements traceability problem, In *Proc. of IEEE International Conference on Requirements Engineering*, pages 94–101, 1994.

[3] A. P. Nikora and G. Balcom, Automated identification of ltl patterns in natural language requirements, In *20th International Symposium on Software Re-liability Engineering*, pages 185–194, 2009.

[4] Fabiano Dalpiaz, Alessio Ferrari, Xavier Franch, Cristina Palomares: Natural Language Processing for Requirements Engineering: The Best Is Yet to Come. *IEEE Software* 35(5): 115-119 (2018)

[5] OpenReq, Project, Accessed: 2019-01-11. [Online]. Available: https://openreq.eu/.

[6] Apache OpenNLP Toolkit, Accessed: 2019-01-11. [Online]. Available:http://opennlp.apache.org

[7] NLP Toolkit for JVM Languages (NLP4J), Part-of-speech Tagging, Accessed: 2019-01-11. [Online]. Available: https://emorynlp.github.io/nlp4j/.

[8] DKPro Similarity, Accessed: 2019-01-11. [Online]. Available: https://dkpro.github.io/dkpro-similarity

[9] WordNet - A Lexical Database for English, Accessed: 2019-01-11. [Online]. Available: https://wordnet.princeton.edu/.