# 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, a solution based on an automatic requirement dependency detection approach. The core of this proposal is based on an ontology which defines dependency relations between specific terminologies related with the domain of the requirements. Using this information, it is possible to apply Natural Language Processing (NLP) techniques to extract meaning from this requirements and relations, and Machine Learning techniques in order to apply conceptual clustering, with the major purpose of classifying this 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, such as *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]. This traceability allows us to identify dependencies between these relations, and how changes in these requirements may affect others that are related to them. Identifying this dependencies between requirements, usually specified in Natural Language (NL) [3] is considered a difficult, expensive and long process.

In order to provide an automated solution for this 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, a EU Horizon 2020 project [4] that aims to provide advanced an innovative tools for community-driven RE.

## 2 OpenReq-DD approach

As a component of the general OpenReq architecture system, OpenReq-DD is exposed as a RESTful service with an API to provide the required data and perform the dependency detection algorithm. This is intended to match a microservices architecture and define an isolated, decoupled component that can be reused in different contexts. For demonstrations, a simple GUI is provided (see Fig. 3 in Sec. 3).



In: A. Editor, B. Coeditor (eds.): Proceedings of the NLP4RE Workshop, Essen, Germany, 18-21 March, 2019, published at http://ceur-ws.org

Figure 1 shows the sequence of steps that OpenReq-DD performs to extract requirement dependencies. In Sec. 2.2, a brief description of each stage is introduced in order to understand the general purpose, the techniques and the tools used in each step.

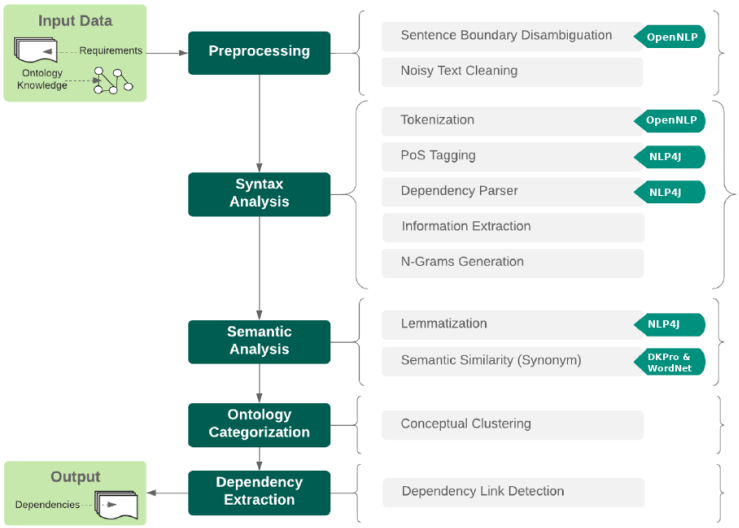



Figure 1: Dependency detection overview

### 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 these data has been defined and discussed alongside stakeholders in the OpenReq project, due to different criterion based on a general perspective of the project.

- **Requirements list**. Using a general JSON OpenReq format, a list of requirements is needed. The JSON Schema used for input and output response is available here.
- **Ontology**. The ontology-based approach pretends to provide to the tool the required knowledge about the general patterns and dependency types that are potential dependencies between requirements. This is the result of an extended 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 here.

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.

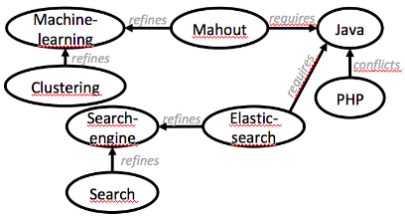



Figure 2: Ontology structure example

### 2.2 Dependency detection process

Given a data set, OpenReq-DD initiates the dependency extraction by following the next steps.

#### 2.2.1 Preprocessing

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

- **Sentence Boundary Disambiguation** (SBD). Apply sentence detection to extract isolated sentences from each requirement, by deciding where is the beginning and the end of each sentence. The Apache toolkit OpenNLP [5] is used for this purpose.
- **Noisy Text Cleaning**. After SBD, a total of 14 rules are applied to clean input text of each sentence. These include: removal of character, numeric and roman numerals list pointers; removal of acronyms that may appear at the beginning of a requirement; removal of scape sequence characters; addition of white spaces to prevent PoS tagger faults (e.g. between parenthesis or question marks); between others.

#### 2.2.2 Syntax Analysis

Given the preprocessed requirements, and therefore a list of cleaned segmented requirements, a syntax analysis is applied 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.
- **PoS tagging**. Each token of the sentence is marked with a part-of-speech tag using the NLP4J toolkit [6].
- **Dependency Parser**. Generates a dependency tree where each node is a token of the input sentence and edges are the relations between parent words and child words. This is done using the NLP4J toolkit.
- **Information Extraction**. Once the data is in an optimal form to be analyzed, the keywords to categorize each requirement into the ontology can be extracted. For this keyword extraction, a set of potential keywords is assumed given the results of manual requirements analysis performed by stakeholders from the project. Given these keywords, a set of patterns were extracted according to the position of these topics.
- **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.

#### 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 keyword 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 [7] 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 [8].

#### 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, it tries to 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 that are linked with a dependency relation are analyzed extracting their instances (i.e. the requirements individuals) to find the dependencies between each requirement.

## 3 Demo plan

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

### 3.1 Environment configuration

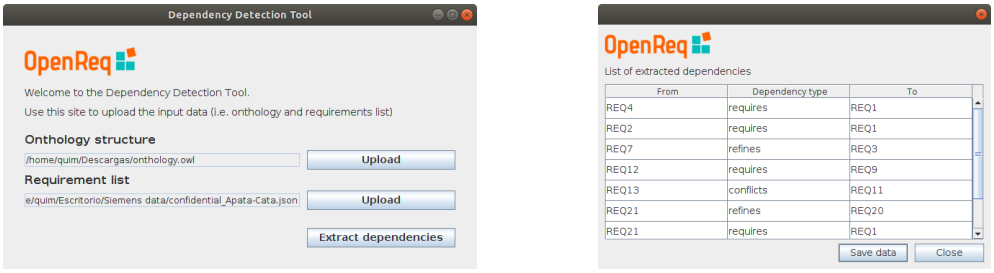
For a demo execution, it is necessary to run both the OpenReq-DD web service and the Java GUI application.

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 structure. Examples of both files are referenced in Sec. 2.1.

### 3.2 Dependency detection execution

Figures 3a and 3b 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 updated, the user can initiate the dependency analysis by clicking on the "Extract dependencies" button.




(a) Input Data View

(b) Results View

Figure 3: OpenReq-DD GUI

After this interaction, the whole process of dependency detection starts: the *back-end* of the tool applies a preprocess to the data, the syntactic analysis, the semantic analysis, the ontology categorization and finally the dependency extraction. 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.

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

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

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

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

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