ORSIM: Integrating existing software components to detect similar natural language requirements

Carlos Adrián Furnari[1], Cristina Palomares[1], Xavier Franch[1]

[1] Universitat Politècnica de Catalunya, Spain

Abstract. [Context & motivation] Requirements Engineering (RE) is considered as one of the most critical phases in software development. Inside RE, interdependency detection and requirements reuse are areas that could be improved and that have been of interest for the research community. **[Problem**] Similarity detection is an activity that emerges in the context of natural language requirements. This activity can be used for interdependency detection and requirements reuse. Although there exist several software components to detect similar texts in English, creating the setup to test them is time-consuming and difficult. **[Principal ideas/results**] In this paper, we present ORSIM (OpenReq-Similarity), a tool which integrates different existing similarity detection components in the same platform. These components are: Cortical, Gensim, ParallelDots, and Semilar. **[Contribution]** ORSIM enables requirements engineers to concentrate on evaluating and choosing the similarity detection component that best suits their user's data rather than worrying about the technical setup of these components.

Keywords: Similarity detection, Paraphrasing detection, Natural language processing, Requirements engineering

1 Introduction

Similarity detection (both from a syntactic and semantic point of view) aims at detecting sentences that express approximately the same idea using different words [1, 2]. In Requirements Engineering (RE), similarity detection can be used for both the identification of interdependencies between requirements and the reuse of knowledge related to requirements.

The relationship between similarity detection and interdependency detection is evident in the case where two requirements have almost the same formulation, since in this case we would have an *OR* interdependency between the requirements. Imagine the requirements *The user interface should use the Arial letter type* and *The user interface should use the Calibri letter type*. It is clear that these two requirements are similar (except for the words *Arial* and *Calibri*) and they cannot be used in the same system (since

it is not possible to use two letter types for the whole user interface), so these requirements are related by an *OR* interdependency (using the terminology proposed by Carlshamre [3]). However, even in other cases not so syntactically similar cases there are commonalities. As an example, if a requirement states that *It shall be possible to filter personal data by name and address* and another one states that *The system shall be able to filter personal data by age*, it would probably be wise to treat these two requirements at the same time to save development resources. This example can be considered as an *ICOST* interdependency (i.e., a requirement affects the implementation of another requirement), again using the terminology proposed by Carlshamre [3].

In requirements reuse, similarity detection could be used to retrieve similar requirements from previous projects. From the identified similar requirements, knowledge about the metadata attributes could be reused (such as effort, risk or priority). In addition, it is possible to know if the retrieved requirements caused problems in previous projects, avoiding their occurrence again in the current one. Similarity detection could also be used to retrieve similar requirements from a reusable requirements database—which contains relationships among the reusable requirements—and proposing requirements to the user that are related to the retrieved one.

The OpenReq project [4] aims at developing, evaluating, and transferring highly innovative methods, algorithms, and tools for community-driven RE in large and distributed software-intensive projects. Inside OpenReq, the detection of similar requirements is a cornerstone, since it is the basis for the dependency detection and the requirements reuse functionalities that it will provide. Some of these functionalities are: discard duplicate requirements, identify related requirements in previous projects to save effort and reuse knowledge, and detect incompatibilities and prevent errors.

Specifically, these functionalities are of special importance for two of the use cases of OpenReq. One use case deals with bid projects of thousands of requirements. They want to identify similar and dependent requirements both in the same bid and in previous bids, and to reuse requirements knowledge of previous bids in the bid at hand. It is expected that this will reduce the cost of the bid phase and the requirements analysis phase by 10%. The other use case has a database with almost a hundred thousand requests (containing requirements and bugs). They have not that much interest in the saving effort and reuse functionality, but in the identification of similar and dependent requests. It is estimated that at least 1000 requests (annually) could improve their management or be avoided. Of course, these are just estimations and they need to be validated during the project.

As there are several well-known components already developed to detect similar texts in English, the goal of OpenReq is not to reinvent the wheel, but to use the most adequate component in every context of use and improve the results by applying pre processing on the input and/or post processing on the output of these components. However, creating the setup to test these different components is time consuming and difficult, since most of them need different settings to work: they use different programming languages (such as Python and Java), some of them are available just as APIs while others are available as coding libraries, some of them need specific databases to store specific data (such as the pre-processing of texts) to speed up their computing process while others can connect to any database, etc.

Therefore, the first goal of ORSIM (standing for OpenReq-SIMilarity) is to integrate different existing similarity detection components in the same environment, so requirement engineers do not have to worry about the different setups and may concentrate on evaluating and choosing the components that best suits their data. The engineering behind ORSIM is a prerequisite for further systematic evaluations. With the ability of doing systematic evaluations, ORSIM can aim to help stakeholders in a more challenging way in the future. The long-term goal of ORSIM is that the tool learns what component and parameterization behaves better for data with specific characteristics (for instance, long versus shorts requirements, requirements containing lots of not so common domain terms, as the ones found in avionics or rail systems, etc.) to not only provide stakeholders with a setup to evaluate the components, but to assist stakeholders in the parameterization and choice of the components that behaves the best for the stakeholder's data.

ORSIM is not only of interest to OpenReq, but to any other stakeholder of the RE community who needs to evaluate different similarity components or choose the one that provides better results for their specific data.

In the following, we present an overview of the ORSIM tool (Section 2). Section 3 presents an initial evaluation of ORSIM and the integrated similarity detection components. Finally, we conclude the paper in Section 4.

2 ORSIM Overview

In this version of ORSIM, four similarity components have been included. These components have been chosen after a literature review done by the authors, choosing the components that have been used in the works reviewed. These four components are:

- *Cortical* [5]. It provides an API with a method that calculates the similarity between two texts using different algorithms, e.g. using Jaccard [2] and Cosine [2] distances.
- *Gensim* [6]. It provides a Python library with methods that allow to load a corpus (similar to a dictionary) and applying the TF-IDF [7], LSA [8], LDA [9] and RP [10] algorithms to obtain the similarity between a text and a corpus.
- *ParallelDots* [11]. It provides a semantic analysis API that uses the cosine similarity to compute the similarity between two given texts.
- *Semilar* [12]. It provides implementations (developed in Java) of a series of algorithms to evaluate the semantic similarity between two texts, through an application and a library. In OpenReq, we will use the library, in order to encapsulate it in an API accessed by ORSIM. Semilar library comes with various similarity methods such as LSA, LDA, BLEU, Meteor, Pointwise Mutual Information, and optimized methods based on Quadratic Assignment (a description of these methods can be found in [13]). Semilar also includes features for text pre-processing, such as tokenizer, tagger, stemmer and parser, being able to choose between different options (e.g., StanfordNLP [14] and OpenNLP [15]).

Each one of these components is encapsulated in an individual API. This allows to update a component without affecting other components.

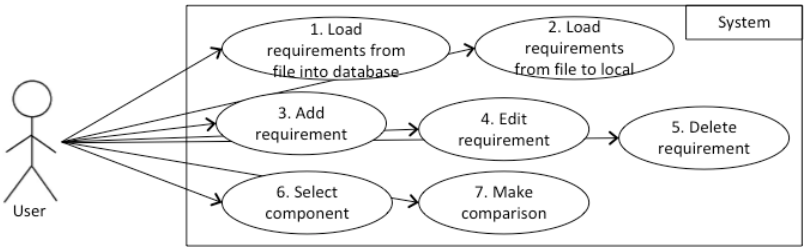

Fig. 1. ORSIM's Use Case (UC) diagram

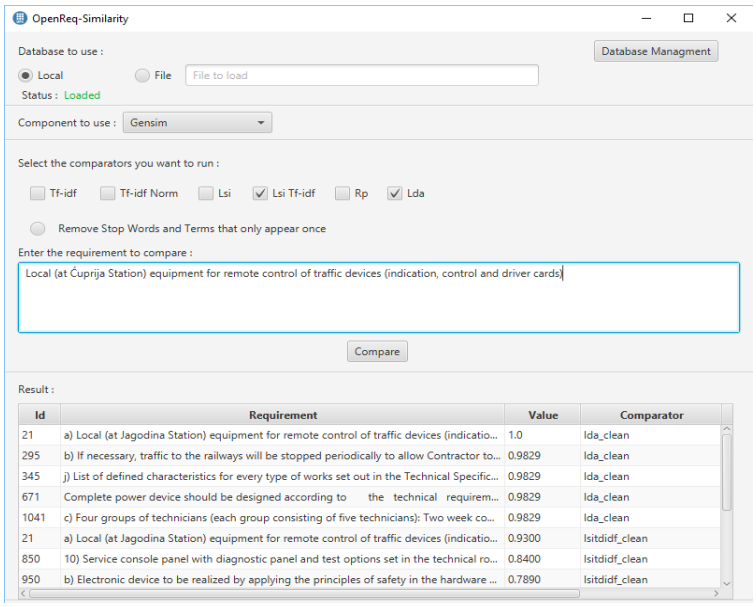

Fig. 2. ORSIM's Usage screen example – Gensim component

To use ORSIM, the requirements engineer must upload a file with all the requirements to fill the database (UC 1 of Figure 1), or to use a file containing requirements (instead of the database) (UC 2 of Figure 1). The tool also allows to edit the requirements in the database (UCs 3, 4 and 5 of Figure 1). After that, from the main window, the requirements engineer can select one of the available components (UC 6 of Figure 1). When a component is chosen, the interface will change to show the parameters that are necessary for the component and the requirement that is to be compared (Figure 2 shows such screen for the Gensim component). Next, the requirements engineer enters the desired value for the parameters, enters the requirement to be compared, and sends the comparison to the ORSIM's server (UC 7 of Figure 1). When the server replies, the results are shown in the bottom of the screen, in a decreasing order of similarity score (see lowest part of Figure 2). As can be seen in the example of Figure 2 (with the database containing requirements of one of the trials of OpenReq), the requirements engineer has entered the requirement "*Local (at Ćuprija Station) equip-*

ment for remote control of traffic devices (indication, control and driver cards)". The component brings the requirement "*Local (at Jagodina Station) equipment for remote control of traffic devices (indication, control and driver cards)*" as the most similar to the one that the requirements engineer has entered.

3 Initial Evaluations

Initial tests were done during the study of the components and the development of ORSIM. In the following, we show the main results of these tests. Although more tests were carried out, we present here only the ones that are representative. Table 1 contains the data of the tests, i.e., the requirement to be analysed (i.e., the one entered by the requirements engineer) and the requirement in the database that the domain expert manually selected as to be the most similar one. Table 2 shows the results of the tests (in the case of the components that are parameterizable, we do not show all the parameterizations tested, but only the one that returned the best result). The column *Pos* in Table 2 refers to the position of the requirement identified manually as the most similar requirement in the list of results identified with the components (this list is ordered using a decreasing order of similarity score).

The used database, which contains 1137 requirements, belongs to one of the trials of OpenReq. Test 1 is between very similar requirements in the database (only a word differs) and the results are very good with all the components. Test 2 to 4 are with requirements that have same meaning of one identified in the database, but the expression (i.e., syntax) is different. In Test 2, the results of Cortical decreased a lot compared to the first test, while the rest gave good results, with ParallelDots standing out as the best. In Test 3, in the case of Semilar, what is supposed to be the best result has position 2, meaning that it identifies one requirement as more similar than the original one, which is: *"The new telephone of level crossing, which is in working condition, must be placed at level crossing PBE3"*. As it can be seen, this requirement is not more similar than the one identified manually, so this is considered a bad result of Semilar. Cortical and Gensim also decreased the similarity scores they returned, while ParallelDots continues behaving well. Finally, in Test 4, Gensim gave the best result, closely followed by ParallelDots, while the rest gave notoriously lower results.

Table 1. Test data

Test	Requirement entered by the user	Most similar requirement in database (identified manually)
1	Local (at Ćuprija Station) equipment for remote control of traffic devices (indication, control and driver cards).	Local (at Jagodina Station) equipment for remote control of traffic devices (indication, control and driver cards).
2	The contractor must provide the spare parts that the employer requested. The prices are specified in the spare parts list.	The Employer have the possibility to request, at the prices specified in the list of spare parts, and the contractor's obligation to deliver, a different number of spare parts specified in the list.
3	To prevent accidents on the rail level crossing, a security camera system is required to be installed.	For the purpose of railway traffic safety increase, it is necessary to install video supervision system at the existing level crossings.
4	The interlocking requirements includes a small maintenance costs, great availability, high reliability and long life duration.	Request of the Employer for interlocking is: high availability, high reliability and low maintenance costs. Long life of products through the use of modern technology.

Table 2. Test results
AWM = Abstract Word Metric, NA = Not Apply, BF = Base Form, WWT to Word Weight Type
** Result obtained for cosine distance*

Test	Component	Parameters	Similarity	Position
1	Cortical	NA	99,99%*	1
1	Gensim	LDA, with stop words	100%	1
1	ParallelDots	NA	100%	1
1	Semilar	Lexical (BF=true)	91,89%	1
2	Cortical	NA	59,85%*	1
2	Gensim	LSI TF-IDF, without stop words	91,30%	1
2	ParallelDots	NA	97,82%	1
2	Semilar	Optimum (AWM=LCH, WWT=MIN/TEXTA)	86,81%	1
3	Cortical	NA	43,08%*	1
3	Gensim	LSI TF-IDF, without stop words	75,28%	1
3	ParallelDots	NA	97,80%	1
3	Semilar	Corley	69,06%	2
4	Cortical	NA	59,75%*	1
4	Gensim	LDA, with stop words	99,03%	1
4	ParallelDots	NA	98,20%	1
4	Semilar	Greedy (BF=false, AWM=LCH)	59,45%	1

4 Conclusions & Future Work

ORSIM allows the comparison of different similarity detection components (Cortical, Gensim, ParallelDots, and Semilar) in a quick and easy way, to know which one suits best the requirements at hand. To test the tool and the components, we performed a series of initial tests with a real requirements database. In our tests, ParallelDots was the component that brought the best results, but this could change depending on the database the requirements engineer is dealing with.

As future work, we want to identify what similarity detection component and parameterization behaves the best in different situations. On the one hand, we want to identify the component and parameterization that provides the best results no matter the set of requirements loaded into the component. With that goal, we will conduct more tests using different datasets and we will explore more the parameters offered by Semilar and Gensim to determine if a different configuration of the ones tested until now behaves better. On the other hand, taking into account that the final goal of ORSIM is assisting stakeholders in the choice of the similarity component and parameterization that will work better for the stakeholders' requirements, we want the tool to be able to learn what component and parameterization behaves better for data with specific characteristics.

Additionally, we aim at integrating other similarity components like DKPro [16], SenseClusters [17] or Scikit-Learn [18], to extend the possibilities of finding better results. In a longer term, we would like to make the ORSIM tool easy to extend with new components.

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. Lee, M.C., et al.: A Grammar-Based Semantic Similarity Algorithm for Natural Language Sentences. The Scientific World Journal (2014).
2. och Dag, J.N., et al.: Evaluating Automated Support for Requirements Similarity Analysis in Market-Driven Development. REFSQ (2001).
3. Carlshamre, P., et al.: An industrial survey of requirements interdependencies in software product release planning. IEEE International Symposium on Requirements Engineering (2001).
4. Intelligent Recommendation and Decision Technologies for Community-Driven Requirements Engineering (Horizon 2020 Project, https://www.openreq.org).
5. Cortical: http://www.cortical.io/ . Last visited: January 22nd, 2018.
6. Gensim: https://radimrehurek.com/Gensim/index.html. Last visited: January 22nd, 2018.
7. Yang, Y., Pederson, J.O.: A Comparative Study on Features selection in Text Categorization. ICML (1997).
8. Foltz, et al.: The measurement of textual coherence with latent semantic analysis. Discourse Processes (1998).
9. Chen, Q., et al.: Short text classification based on LDA topic model. (ICALIP, 2016).
10. Lin, J., et al.: Dimensionality reduction by random projection and latent semantic indexing. SDM (2003).
11. ParallelDots: https://www.paralleldots.com/semantic-analysis. Last visited: January 22nd, 2018.
12. Semilar: http://deeptutor2.memphis.edu/Semilar-Web/index.jsp. Last visited: January 22nd, 2018.
13. Rus, V., et al.: SEMILAR: The Semantic Similarity Toolkit. ACL (2013).
14. StanfordNLP: https://nlp.stanford.edu/. Last visited: January 22nd, 2018.
15. OpenNLP: https://opennlp.apache.org/. Last visited: January 22nd, 2018.
16. DKPro: http://dkpro.github.io/. Last visited: January 22nd, 2018.
17. SenseClusters: http://senseclusters.sourceforge.net/. Last visited: January 22nd, 2018.
18. Scikit-Learn: http://scikit-learn.org/stable/. Last visited: January 22nd, 2018.