Traceability Links Recovery in BPMN Models

Raúl Lapeña

SVIT Research Group, Universidad San Jorge, Spain
rlapena@usj.es

Abstract. Traceability Links Recovery has been a topic of interest for many years. However, Traceability Links Recovery in models in general, and BPMN models in particular, has not received enough attention yet. Through my work, I aim to fill this research gap by studying Traceability Links Recovery between requirements and BPMN models. So far, under the tutelage of directors Carlos Cetina and Óscar Pastor, I adapted Traceability Links Recovery code techniques to work over BPMN models. The produced approach was applied to two different case studies, an academic one and an industrial one. The outcomes of the research outperformed the state of the art baseline. Under the light of these novel findings, opportunities for new research unfold.

Keywords: Traceability Links Recovery, BPMN Models, Model Driven Engineering

1 Introduction

Traceability Links Recovery (TLR) is defined as the software engineering task that deals with the identification and comprehension of dependencies and relationships between software artifacts. It has been a subject of investigation for many years within the software engineering community [1, 2]. Research has shown that affordable traceability can be critical to the success of a project [3], and leads to increased maintainability and reliability of software systems [4], also decreasing the expected defect rate in developed software [5]. In recent years, TLR has been attracting more attention [6]. However, most of the works focus on performing TLR tasks in code artifacts [7], while TLR in process models is a topic that has not received enough attention yet. Through my work, I aim to fill this research gap by studying TLR between requirements and process models. So far, under the tutelage of directors Carlos Cetina and Óscar Pastor, I adapted TLR code techniques to work over process models (specifically, BPMN models). More precisely, through the work presented in [8], we studied TLR between requirements and process models through three different approaches, two adapted code techniques and a models-specific baseline. Given a query requirement and a process model, the three approaches used different means to extract a fragment from the model, relevant to the implementation of the query requirement.

The three approaches were evaluated through the Camunda BPMN for Research case study (github.com/camunda/bpmn-for-research) and through a

real-world industrial case study, provided by our industrial partner, CAF (Construcciones y Auxiliar de Ferrocarriles, `www.caf.es/en`), a worldwide provider of railway solutions. One of the adapted code techniques achieved the best results for all the measured performance indicators in both case studies, outperforming the other two techniques. The overall findings of our paper suggested that adapting code techniques that provided good results in code was beneficial for TLR between requirements and BPMN models, since the outcomes outperformed those of a models-specific baseline. Under the light of these findings, a research question arises, unfolding opportunities for novel research: *How can we further improve TLR in BPMN models?*

The rest of the paper is structured as follows: Section 2 describes the Approach that obtained the best results and how to apply it to TLR between requirements and BPMN models. Section 3 details the baseline technique and the designed Evaluation. Section 4 presents the obtained Results. Section 5 formulates the Research Question that arises from our ongoing work. Section 6 discusses potential Future Work. Section 7 mentions the research Methodology in use. Finally, Section 8 reviews the works related to this one.

2 Approach

This section describes the Mutation Search technique, the technique designed in [8] that obtained the best results for TLR between requirements and BPMN models, providing insight on its steps, application, and outcomes.

2.1 Mutation Search

The Mutation Search technique receives a *query* requirement and a BPMN model as input, generates a population of fragments, and ranks said fragments through Latent Semantic Indexing. From the ranking, the first fragment is taken as the proposed solution. In order to generate the fragments population, algorithm 1 is followed. In the algorithm, an empty population and a seed fragment (chosen randomly from the input model) are created. Then, until the algorithm meets a stop condition (for instance, a certain number of iterations), the fragment is mutated and each new mutation is added to the population, avoiding the addition of repeated fragments.

In the algorithm, a mutation in a fragment can be caused by: (1) adding one new event, gateway, or task that is connected to an already present event, gateway, or task, (2) removing an element with only one connection, or (3) adding or removing a lane from the fragment. The performed mutation is chosen randomly on each iteration.

The top part of Fig. 1 shows this process, having the example input BPMN model on the left, and some example fragments on the right, generated through the usage of the algorithm. The generated fragments are represented through the text contained in all their elements. The text of both the input requirement and the generated fragments is then processed through general phrase

Algorithm 1 Mutation Search Algorithm

```
1: P ← []                                          ▷ Initialize the population
2: F ← randomFragment(inputModel)                  ▷ Create an initial seed fragment
3: while !(StopCondition) do                       ▷ While the stop condition is not met
4:     F ← mutateFragment(F)                       ▷ Mutate the fragment
5:     if !(F ∈ P) then                            ▷ If the new fragment is not in the population
6:         P ← P + F                               ▷ Add the new mutation to the population
7:     end if
8: end while
9: return P                                        ▷ Return the population
```

styling techniques (lowercasing and tokenization), Parts-Of-Speech Tagging [9], and Lemmatizing [10].

Finally, the requirement and the fragments are fed into Latent Semantic Indexing, which ranks the fragments according to their similitude to the requirement. Latent Semantic Indexing (LSI) [11] is an automatic mathematical/statistical technique that analyzes relationships between *queries* and *documents* (bodies of text). LSI has been successfully used to retrieve Traceability Links between different kinds of software artifacts in different contexts [7].

To that extent, LSI produces a *term-by-document co-ocurrence matrix*. The bottom left part of Fig. 1 shows an example *term-by-document co-occurrence matrix*, with values associated to an example. Each row in the matrix (*term*) stands for each of the words that appear in the processed text of the requirement and the model elements. Each column in the matrix (*document*) stands for each of the fragments (MF1 to MFn) generated through the algorithm. The final column (*query*), stands for the processed input requirement. Each cell in the matrix contains the frequency of each *term* in each *document*.

Vector representations of the *documents* and the *query* are obtained by normalizing and decomposing the *term-by-document co-occurrence matrix* using a matrix factorization technique called *Singular Value Decomposition* (SVD) [11]. In Fig. 1, a three-dimensional graph of the SVD is provided, on which it is possible to notice the vectorial representations of some of the columns. To measure the similarity degree between vectors, the cosine between the *query* vector and the *documents* vectors is calculated. Cosine values closer to one denote a high degree of similarity, and cosine values closer to minus one denote a low degree of similarity. Through this measurement, the fragments are ordered according to their similarity degree to the requirement, producing the relevancy ranking shown on the bottom right part of Fig. 1. From the ranking, the first fragment is considered as the candidate solution for the requirement, and consequently taken as the final output of the Mutation Search technique.

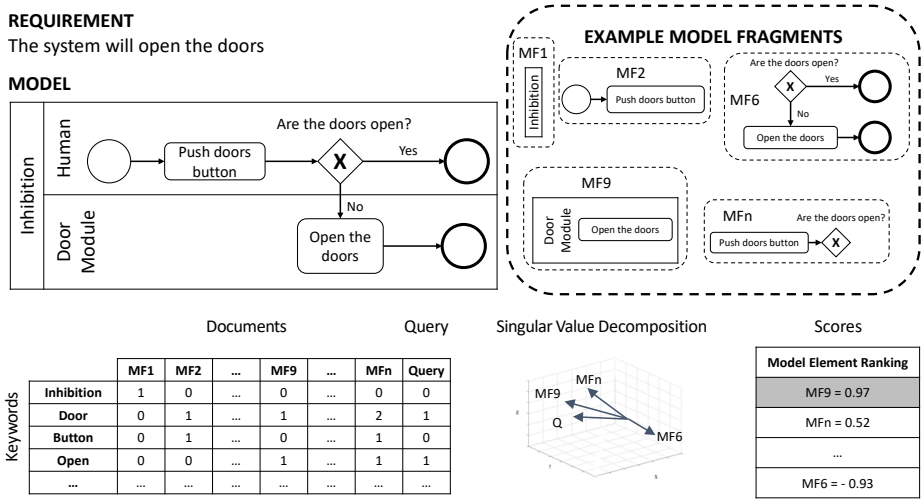

Fig. 1. Mutation Search Technique Example

3 Evaluation

The following paragraphs introduce the baseline, the experimental setup, the case studies, and the oracles used to evaluate the baseline and Mutation Search. This section also details the design of the evaluation.

3.1 Linguistic Rule-Based Baseline

Spanoudakis et al. [12] present a linguistic rule-based approach to support the automatic generation of traceability links between natural language requirements and conceptual models. Specifically, the traceability links between the requirements and the conceptual models are generated through a set of *requirement-to-object-model* (RTOM) rules that specify sequences of terms and grammatical patterns. The technique searches for matching patterns in the requirements and the conceptual models, producing a link per each found match. We worked with a set of rules adapted so that the technique works over BPMN models.

3.2 Experimental Setup

Through [8], TLR between requirements and BPMN models is performed. The results obtained by Mutation Search are compared against those of a models-specific baseline. An overview evaluation can be seen in Fig. 2. The top part shows the inputs, extracted from the documentation provided in the case studies: requirements, BPMN models, and the approved traceability between both. The approved traceability is a document that depicts the correct fragments that

correspond to the requirements. It is provided by software engineers from our industrial partner, and conforms the oracle of the evaluation. For each case study, the linguistic baseline takes the mentioned inputs, and generates a single fragment for each requirement. The generated fragment is compared with the oracle fragment. The Mutation Search technique generates a ranking of fragments per requirement instead. Since the rankings are ordered from best to worst traceability, the first fragment in each ranking is picked for comparison against its corresponding oracle. Once the comparisons are performed, a confusion matrix is calculated both for the baseline and for Mutation Search.

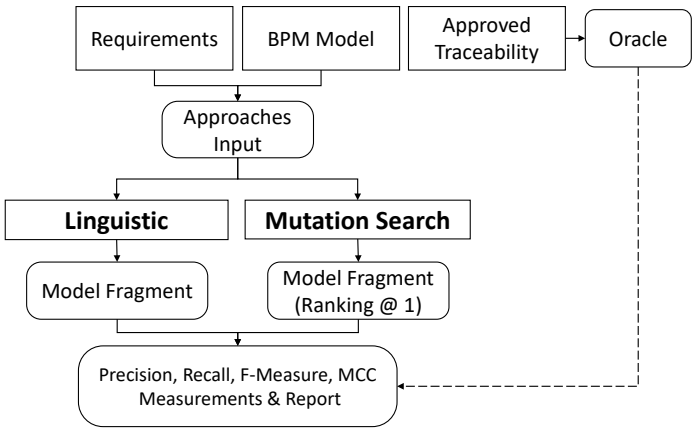

Fig. 2. Experimental Setup

A confusion matrix is a table that is often used to describe the performance of a classification model (in this case, the linguistic baseline and Mutation Search) on a set of test data (the solutions) for which the true values are known (from the oracle). The confusion matrix distinguishes between the predicted values and the real values, classifying them into four categories: (1) true positive; (2) false positive; (3) true negative; and (4) false negative. Then, some performance measurements are derived from the values in the confusion matrix. In particular, a report including four performance measurements (recall, precision, f-measure, and MCC) is created for each of the two case studies, both for the baseline and for Mutation Search.

3.3 Case Study and Oracles

In order to perform the evaluation of the approaches, we relied on two different case studies: (1) the Camunda BPMN for Research academic repository, and (2) a set of BPMN models provided by CAF, our industrial partner. In order to obtain the performance results of the approaches, we relied on the available correct solutions, provided in both case studies.

Camunda BPMN for Research: The Camunda BPMN for Research case study consists of four BPMN modeling exercises. Each exercise contains an associated textual description and the solution model for the provided description. In order to apply the approaches to the Camunda case study, a software engineer derived a set of requirements from the problem descriptions. Each exercise has an associated solution model for the provided description. The same software engineer who derived the requirements from the problem descriptions also generated a set of fragments from the solution model, mapping each fragment to a single requirement. Thus, we were provided with a set of requirements, the fragments that implement them, and the TLR mapping between both artifacts.

CAF: For our evaluation, CAF provided us with the requirements and BPMN models of five railway solutions. They also provided us with their existing documentation on requirements to BPMN models traceability, where each requirement is also mapped to a single fragment.

4 Results

Table 1 outlines the results. Each row shows the obtained precision, recall, f-measure, and MCC values. The Mutation Search technique achieved the best results for all the performance indicators in both case studies, providing a mean precision value of 63%, a mean recall value of 77%, a combined F-measure of 68%, and an MCC value of 0.60 for the Camunda BPMN for Research case study, and a mean precision value of 79%, a mean recall value of 72%, a combined F-measure of 74%, and an MCC value of 0.69 for the CAF case study.

Table 1. Mean Values and Standard Deviations for Precision, Recall and F-Measure

	Precision	Recall	F-Measure	MCC
Linguistic - Camunda	40%±25%	35%±22%	33%±13%	0.25±0.19
Linguistic - CAF	35%±28%	35%±10%	30%±7%	0.18±0.13
Mutation Search - Camunda	63%±21%	77%±22%	68%±19%	0.60±0.24
Mutation Search - CAF	79%±19%	72%±19%	74%±16%	0.69±0.20

5 Research Question

From the results of our work, a Research Question arises: *How can we further improve TLR in BPMN models?* The following section will address this question, briefly mentioning some of the possible future works derived from a close inspection of the results.

6 Future Work

This section presents some ideas and opportunities for future work that arose from the presented Research Question:

1. **(Accepted - CAiSE 2019)** Tacit knowledge in the requirements may have a negative impact on semantic-based techniques. How can we minimize this impact?
2. **(Currently under review - IS CAiSE 2018 special issue)** BPMN models have some particularities that other models lack. Could we take in account these particularities in our techniques in order to lead them to enhanced results?
3. **(Ongoing work)** BMP models have less text that other models. Could we enrich the text of BPMN models to improve TLR techniques based on text search?

7 Methodology

To perform this research, as well as our ongoing work, we have followed the design science methodology guidelines presented in [13].

8 Related Work

Related works focus on the impact and application of linguistic techniques to TLR problem resolution at several levels of abstraction. Works like [14, 15] use linguistic approaches to tackle specific TLR problems. In [16], the authors use linguistic techniques to identify equivalence between requirements. The work presented in [17] uses linguistic techniques to study how changes in requirements impact other requirements in the same specification. Our work is not based or focused on linguistic techniques as a means of TLR analysis, but we rather study novel techniques to perform TLR between requirements and BPMN models.

Other works target the application of LSI to TLR tasks. De Lucia et al. [18] present a tool based on LSI in the context of an artifact management system. [19] takes in consideration the possible configurations of LSI when using the technique for TLR between requirement artifacts. Through our work, we do not study the management of artifacts nor different LSI configurations or how LSI configurations impact the results of TLR, but we rather study TLR between requirements and BPMN models.

References

1. Gotel, O.C., Finkelstein, C.: An Analysis of the Requirements Traceability Problem. In: Proceedings of the First International Conference on Requirements Engineering, IEEE (1994) 94–101

2. Spanoudakis, G., Zisman, A.: Software Traceability: a Roadmap. Handbook of Software Engineering and Knowledge Engineering **3** (2005) 395–428
3. Watkins, R., Neal, M.: Why and How of Requirements Tracing. IEEE Software **11**(4) (1994) 104–106
4. Ghazarian, A.: A Research Agenda for Software Reliability. IEEE Reliability Society 2009 Annual Technology Report (2010)
5. Rempel, P., Mäder, P.: Preventing Defects: the Impact of Requirements Traceability Completeness on Software Quality. IEEE Transactions on Software Engineering **43**(8) (2017) 777–797
6. Parizi, R.M., Lee, S.P., Dabbagh, M.: Achievements and Challenges in State-of-the-Art Software Traceability between Test and Code Artifacts. IEEE Transactions on Reliability **63**(4) (2014) 913–926
7. Rubin, J., Chechik, M.: A Survey of Feature Location Techniques. In: Domain Engineering. Springer (2013) 29–58
8. Lapeña, R., Font, J., Cetina, C., Pastor, O.: Exploring new directions in traceability link recovery in models: The process models case. In: Proceedings of the 30th International Conference on Advanced Information Systems Engineering (CAiSE). (2018)
9. Hulth, A.: Improved Automatic Keyword Extraction given more Linguistic Knowledge. In: Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics (2003) 216–223
10. Plisson, J., Lavrac, N., Mladenic, D., et al.: A Rule Based Approach to Word Lemmatization. In: Proceedings of the 7th International Multi-Conference Information Society. Volume 1., Citeseer (2004) 83–86
11. Landauer, T.K., Foltz, P.W., Laham, D.: An Introduction to Latent Semantic Analysis. Discourse Processes **25**(2-3) (1998) 259–284
12. Spanoudakis, G., Zisman, A., Pérez-Minana, E., Krause, P.: Rule-Based Generation of Requirements Traceability Relations. Journal of Systems and Software **72**(2) (2004) 105–127
13. Wieringa, R.J.: Design science methodology for information systems and software engineering. Springer (2014)
14. Sultanov, H., Hayes, J.H.: Application of Swarm Techniques to Requirements Engineering: Requirements Tracing. In: 18th IEEE International Requirements Engineering Conference. (2010)
15. Sundaram, S.K., Hayes, J.H., Dekhtyar, A., Holbrook, E.A.: Assessing Traceability of Software Engineering Artifacts. Requirements Engineering **15**(3) (2010)
16. Falessi, D., Cantone, G., Canfora, G.: Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques. Transactions on Software Engineering **39**(1) (2013)
17. Arora, C., Sabetzadeh, M., Goknil, A., Briand, L.C., Zimmer, F.: Change Impact Analysis for Natural Language Requirements: An NLP Approach. In: IEEE 23rd International Requirements Engineering Conference. (2015)
18. De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Enhancing an Artefact Management System with Traceability Recovery Features. In: Proceedings of the 20th IEEE International Conference on Software Maintenance, IEEE (2004) 306–315
19. Eder, S., Femmer, H., Hauptmann, B., Junker, M.: Configuring Latent Semantic Indexing for Requirements Tracing. In: Proceedings of the 2nd International Workshop on Requirements Engineering and Testing. (2015)