Exploring New Directions in Traceability Link Recovery in Models: The Process Models Case

Raúl Lapeña[1(✉)], Jaime Font[1], Carlos Cetina[1], and Óscar Pastor[2]

[1] SVIT Research Group, Universidad San Jorge,
Autovía A-23 Zaragoza-Huesca Km. 299, Villanueva de Gállego, Spain
{rlapena,jfont,ccetina}@usj.es
[2] Centro de Investigación en Métodos de Producción de Software,
Universitat Politècnica de València, Valencia, Spain
opastor@pros.upv.es

Abstract. Traceability Links Recovery (TLR) has been a topic of interest for many years. However, TLR in Process Models has not received enough attention yet. Through this work, we study TLR between Natural Language Requirements and Process Models through three different approaches: a Models specific baseline, and two techniques based on Latent Semantic Indexing, used successfully over code. We adapted said code techniques to work for Process Models, and propose them as novel techniques for TLR in Models. The three approaches were evaluated by applying them to an academia set of Process Models, and to a set of Process Models from a real-world industrial case study. Results show that our techniques retrieve better results that the baseline Models technique in both case studies. We also studied why this is the case, and identified Process Models particularities that could potentially lead to improvement opportunities.

Keywords: Traceability Link Recovery · Requirements engineering
Business Process Models

1 Introduction

Traceability Link Recovery (TLR) 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 by making it possible to verify and trace non-reliable parts [4]. Specifically, more complete Traceability decreases the expected defect rate in developed software [5].

In recent years, TLR has been attracting more attention, becoming a subject of both fundamental and applied research [6]. However, most of the works focus on code [7], and the application of Traceability Links Recovery techniques to Process Models is a topic that has not received enough attention yet.

J. Krogstie and H. A. Reijers (Eds.): CAiSE 2018, LNCS 10816, pp. 359–373, 2018.
https://doi.org/10.1007/978-3-319-91563-0_22

Through this work, we study TLR between Natural Language Requirements and Process Models through three different approaches. Given a query Requirement and a Process Model, the three techniques use different means to extract a Model Fragment from the Model, being said Model Fragment relevant to the implementation of the query Requirement. The first technique is a Linguistic technique based on Parts-of-Speech (POS) Tagging and Traceability rules [8]. The technique was designed specifically for TLR in Models, and is used as a baseline against which the proposed techniques are compared. The other two techniques (named 'Aggregation' and 'Mutation Search') are based on Latent Semantic Indexing and Singular Value Decomposition, a well-spread Information Retrieval technique that has been applied previously to TLR in code, obtaining good results in the process [7]. None of the two LSI-based techniques have been applied to extract TLR between Requirements and Process Models previously. Therefore, we adapted them to work for Process Models and propose them as novel techniques in the field.

The three approaches were evaluated through the Camunda BPMN for Research case study (https://github.com/camunda/bpmn-for-research), as well as through a real-world industrial case study, provided by our industrial partner, CAF (Construcciones y Auxiliar de Ferrocarriles, http://www.caf.es/en), a worldwide provider of railway solutions.

Results show that the Mutation Search technique achieves the best results for all the measured 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. In contrast, the Linguistic baseline and the Aggregation technique present worse results in these same measurements in both case studies.

The overall findings of our paper suggest that adapting techniques that have provided good results in code is beneficial for TLR between Requirements and Process Models, since their results outperform those of a technique created specifically with Models in mind. Moreover, studied why this is the case, and identified Process Models particularities that could potentially lead to improvement opportunities.

The rest of the paper is structured as follows: Sect. 2 describes our Approach, that is, our proposed techniques and how to apply them to TLR between Requirements and Process Model fragments. Section 3 details the baseline technique and the designed Evaluation. Section 4 presents the obtained results. Section 5 discusses the outcomes of the paper. Section 6 presents the Threats to Validity of our work. Section 7 reviews the works related to this one. Finally, Sect. 8 concludes the paper.

2 Approach

Through the following paragraphs, we give an introduction on Latent Semantic Indexing, the technique upon which we base the two novel techniques proposed

for TLR between Requirements and Process Models. Afterwards, we describe said techniques, providing insight on their steps, application, and outcomes.

2.1 Latent Semantic Indexing

Latent Semantic Indexing (LSI) [9] 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, specially among Requirements and code [7]. This is due to the fact that code often encodes domain knowledge in the form of domain terms, which are also encoded in the Requirements, hence causing LSI to detect similitude between both.

So far, the technique has not been transported to Process Models. We propose two techniques that use LSI for TLR between Requirements and Process Models. In particular, both techniques use LSI to produce a Model Fragment from the Process Model that serves as a candidate for realizing the Requirement. The following sections give more details on the process.

2.2 Aggregation

The first of the two proposed techniques receives a *query* Requirement and a Process Model as input, and generates a ranking of Model Elements through LSI. From the ranking, a Model Fragment is generated. To that extent, the Process Model is firstly split into Model Elements, represented through the text they contain, which is extracted and used as input for LSI. The top part of Fig. 1 shows this process, having the example input Process Model on the left, and the resulting Model Elements on the right, including: (1) lanes 'Inhibition', 'Human', and 'PLC' (ME1, ME2, ME3); (2) the start and end events (ME4, ME10, ME14); (3) the exclusive gateway 'Are the doors open?' (ME8); (4) the 'Push the doors button' and 'Open the doors' tasks (ME6, ME12); and (5) the sequence flows of the diagram (ME5, ME7, ME9, ME11, ME13).

The text of the Requirement and the Model Elements is then treated through Natural Language Processing techniques. To that extent, general phrase styling techniques, Parts-Of-Speech Tagging [10], and Lemmatizing [11] are applied.

Finally, the Requirement and the Model Elements are fed into LSI, which ranks the Model Elements according to their similitude to the Requirement. The bottom left part of Fig. 1 shows an example *term-by-document co-occurrence matrix*, with values associated to our running example. In the following paragraph, an overview of the elements of the matrix is provided.

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. In Fig. 1, it is possible to notice a subset of said words such as 'Door' or 'Button' as the *terms* of each row. Each column in the matrix (*document*) stands for each of the Model Elements extracted from the input Process Model. In Fig. 1, it is possible to notice identifiers in the columns such as 'ME3' or 'ME12', which stand for the *documents* of those particular Model Elements (namely, the processed text

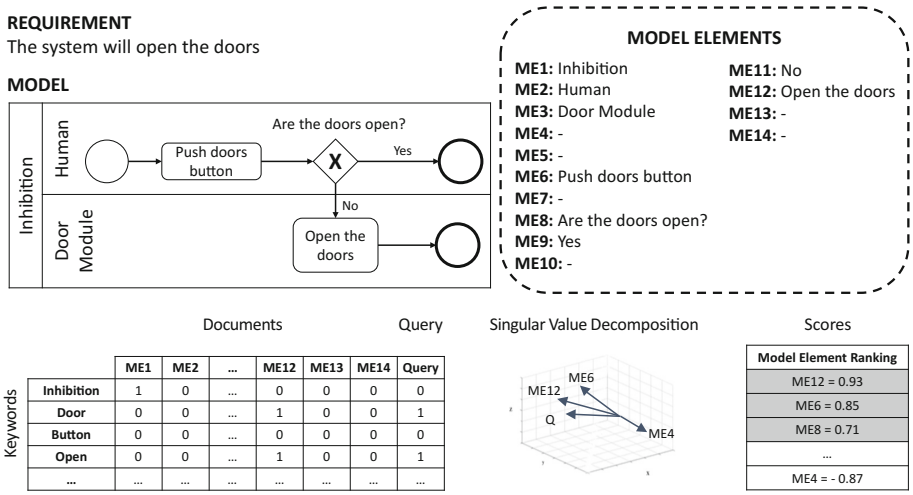

Fig. 1. Aggregation technique example

of 'ME3' and 'ME12'). The final column (*query*), stands for the processed input Requirement. Each cell in the matrix contains the frequency of each *term* in each *document*. For instance, in Fig. 1, the *term* 'Door' appears once in the 'ME12' *document* and once in the *query*.

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) [9]. SVD is a form of factor analysis, or more properly the mathematical generalization of which factor analysis is a special case. In SVD, a rectangular matrix is decomposed into the product of three other matrices. One component matrix describes the original row entities as vectors of derived orthogonal factor values, another describes the original column entities in the same way, and the third is a diagonal matrix containing scaling values such that when the three components are matrix-multiplied, the original matrix is reconstructed.

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. For legibility reasons, only a small set of the columns is represented. 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. Similarity increases as vectors point in the same general direction (as more *terms* are shared between *documents*). Through this measurement, the Model Elements are ordered according to their similarity degree to the Requirement.

The relevancy ranking (which can be seen in Fig. 1) is produced according to the calculated similarity values. In this example, LSI retrieves 'ME12', 'ME6', and 'ME8' in the first, second, and third position of the relevancy ranking due to

their *query-documents* cosines being '0.9343', '0.8524' and '0.7112', implying high similarity between the Model Elements and the Requirement. On the opposite, the 'ME4' Model Element is returned in a latter position of the ranking due to its *query-document* cosine being '−0.8736', implying a low similarity degree.

From the ranking, of all the Model Elements, those that have a similarity measure greater than x must be taken into account. The heuristic that we adopted, and that is used in other works, is $x = 0.7$ [12,13]. This value corresponds to a 45° angle between the corresponding vectors. Nevertheless, the selection of this threshold is an issue still under study, and its proper parametrization has not been tackled in Process Models yet.

Following this principle, the Model Elements with a similarity measure equal or superior to $x = 0.7$ are taken to conform a Model Fragment, candidate for realizing the Requirement. Through the example provided in Fig. 1, 'ME12', 'ME6' and 'ME8' are the Model Elements that conform the Model Fragment for the Requirement, due to their cosine values being superior to the 0.7 threshold. The Model Elements below the threshold, except for 'ME4', are not shown in the ranking for space and understandability reasons. The Model Fragment generated in this manner is the final output of the Aggregation technique.

2.3 Mutation Search

The second of the two proposed techniques receives a *query* Requirement and a Process Model as input, generates a population of Model Fragments, and ranks said Model Fragments through LSI. From the ranking, the first Model Fragment is taken as the proposed solution. In order to generate the Model Fragments population, Algorithm 1 is followed. In the algorithm, an empty population and a seed Fragment (chosen randomly from the input Process Model) are created. Then, until the algorithm mets 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 (the flow that causes the connection is also added to the Fragment), (2) removing an Element with only one connection (and the flow that causes said 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. 2 shows this process, having the example input Process Model on the left, and some example Model Fragments on the right, generated through the usage of the algorithm. The generated Model Fragments are represented through the text contained in all their elements. The text of both the input Requirement and the generated Model Fragments is then processed through general phrase styling techniques, Parts-Of-Speech Tagging, and Lemmatizing.

Finally, the Requirement and the Model Fragments are fed into LSI, which ranks the Model Fragments according to their similitude to the Requirement. The bottom left part of Fig. 2 shows an example *term-by-document co-occurrence matrix*, with values associated to our running example. The technique works

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
```

exactly as it does in the Aggregation technique, except that each column in the matrix (*document*) stands for each of the Model Fragments (MF1 to MFn) generated through the algorithm instead of standing for a single Model Element.

Vector representations of the *documents* and the *query* are obtained by normalizing and decomposing the *term-by-document co-occurrence matrix* using SVD, and the vectorial similarity degrees are calculated through the cosines. The relevancy ranking on Fig. 2 is produced according to the calculated similarity degrees. In this example, LSI retrieves 'MF9' in the first position of the relevancy ranking due to its *query-documents* cosine being '0.9791'. On the opposite, the 'MF6' Model Fragment is returned in the last position of the ranking due to its *query-document* cosine being '−0.9384'.

From the ranking, the first Model Fragment is considered as the candidate solution for the Requirement, and consequently taken as the final output of the Mutation Search technique.

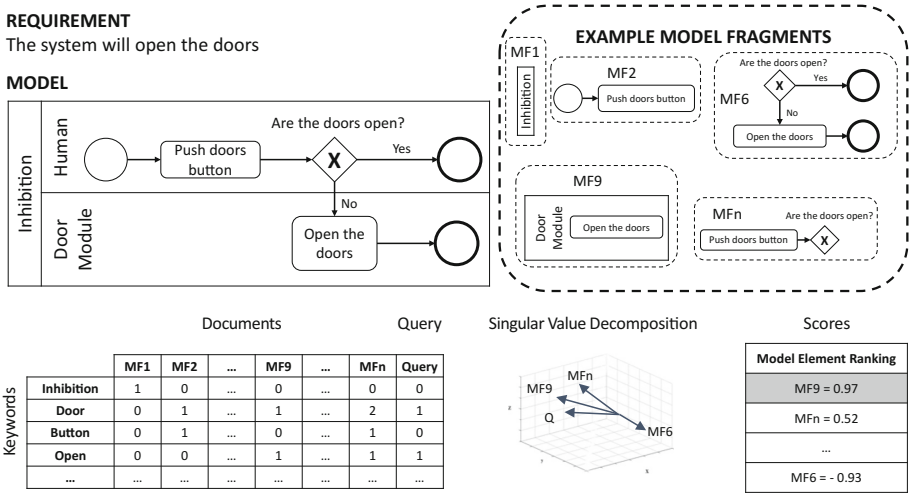

Fig. 2. Mutation search technique example

3 Evaluation

Through the following paragraphs, we introduce the experimental setup and the case studies used to evaluate the baseline and our two proposed approaches, present the oracles used in the evaluation, and detail the design and implementation of said evaluation.

3.1 Experimental Setup

The goal of our work is to perform TLR between Requirements and Process Models through the two proposed techniques, and to compare the results obtained by said techniques against those of a Models specific baseline. Figure 3 shows an overview of the process that was followed to evaluate the Linguistic baseline and our two proposed techniques. The top part shows the inputs, which are extracted from the documentation provided in the case studies: Requirements, Process Models, and approved Traceability between Requirements and Process Models. Each case study comprises a set of Requirements, a Process Model, and an Approved Requirements to Model Fragments Traceability document, which conforms the oracle of our evaluation.

For each case study, the Linguistic baseline and the Aggregation technique take the mentioned inputs, and generate a single Model Fragment for each Requirement. The generated Model Fragments are compared with the oracle Model Fragment. The Mutation Search technique generates a ranking of Model Fragments per Requirement instead. Since the rankings are ordered from best to worst Traceability, the first Model Fragment in each ranking is picked for comparison against its corresponding oracle. Once the comparisons are performed, a confusion matrix is calculated for the baseline and for each technique separately.

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 both of our techniques) on a set of test data (the solutions) for which the true values are known (from the oracle). In our case, each solution outputted by the three techniques is a Model Fragment composed of a subset of the Model Elements that are part of the Process Model. Since the granularity is at the level of Model Elements, the presence or absence of each Model Element is considered as a classification. The confusion matrix distinguishes between the predicted values and the real values, classifying them into four categories: (1) True Positive (TP), values that are predicted as true (in the solution), and are true in the real scenario (the oracle); (2) False Positive (FP), values that are predicted as true (in the solution), but are false in the real scenario (the oracle); (3) True Negative (TN), values that are predicted as false (in the solution), and are false in the real scenario (the oracle); and (4) False Negative (FN), values that are predicted as false (in the solution), but are true in the real scenario (the oracle).

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 Matthews Correlation Coefficient) is created for the case studies, for each of the three techniques.

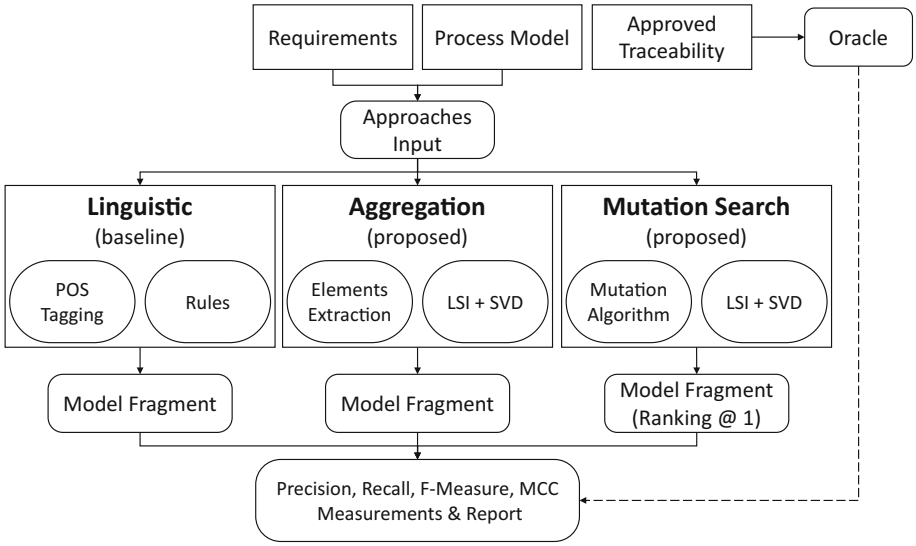

Fig. 3. Experimental setup

Recall measures the number of elements of the solution that are correctly retrieved by the proposed solution. Precision measures the number of elements from the solution that are correct according to the ground truth. F-measure corresponds to the harmonic mean of Precision and Recall [14].

However, none of these previous measures correctly handle negative examples (TN). The **MCC** is a correlation coefficient between the observed and predicted binary classifications that takes into account all the observed values (TP, TN, FP, FN), and is defined as follows:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Recall values can range between 0% (which means that no single model element from the realization of the requirement obtained from the oracle is present in the model fragment of the solution) to 100% (which means that all the model elements from the oracle are present in the solution). Precision values can range between 0% (which means that no single model element from the solution is the oracle) to 100% (which means that all the model elements from the solution are present in the oracle). A value of 100% precision and 100% recall implies that both the solution and the requirement realization from the oracle are the same. MCC values can range between -1 (which means that there is no correlation between the prediction and the solution) to 1 (which means that the prediction is perfect). Moreover, a MCC value of 0 corresponds to a random prediction.

3.2 Linguistic Rule-Based Baseline

Spanoudakis et al. [8] present a linguistic rule-based approach to support the automatic generation of Traceability Links between Natural Language Requirements and Models. Specifically, the Traceability Links are generated following two stages: (1) a Parts-of-Speech (POS) tagging technique [15] is applied on the Requirements that are defined using Natural Language, and (2) the Traceability Links between the Requirements and the Models are generated through a set of *Requirement-to-object-Model* (RTOM) rules.

The RTOM rules are specified by investigating grammatical patterns in Requirements. These rules are specified as sequences of terms, and define relations between Requirements and Model Elements. For instance, a rule may attempt to match a *verb-article-noun* pattern that appears in a Requirement with the text that appears in a Model Element. The rules are atomic: the matching succeeds if the Model Element contains the same words in the same pattern.

In [8], the authors propose 26 rules, applied to a Requirement and a Model in order to retrieve a set of Model Elements from the Model that are related to the Requirement. These Model Elements compose the Model Fragment as a result. We worked with a set of rules adapted to work over Process Models.

3.3 Case Study

In order to perform the evaluation of the three approaches, we rely on two different case studies: (1) the Camunda BPMN for Research academic repository, and (2) a set of Process Models provided by CAF, our industrial partner.

Camunda BPMN for Research: The Camunda BPMN for Research case study consists of four Process Modeling exercises. Each exercise contains an associated textual description and the solution Model for the provided description. In order to apply the three approaches to the Camunda case study, a software engineer (with BPMN expertise, and who is not related to the writing of this paper) derived a set of Natural Language Requirements from the problem descriptions. On average, there are around 15 Requirements per problem, with an approximate average of 25 words per requirement. The Models in the case study contain an approximate average of 25 elements per Model.

CAF: For our evaluation, CAF provided us with Natural Language Requirements and Process Models of five railway solutions from Auckland, Bucharest, Cincinnati, Houston, and Kaohsiung. The functionalities are specified through about 100 Natural Language Requirements each, with an approximate average of 50 words per Requirement. Regarding the Process Models, the distinct functionalities are specified through an average 850 total model elements.

3.4 Oracle

In order to obtain the performance results of the three approaches, their outcomes must be compared against the correct solutions of the two case studies.

Camunda BPMN for Research: In the case of the Camunda BPMN for Research case study, each exercise has an associated solution Model for the provided description. The same software engineer who derived the Natural Language Requirements from the problem descriptions also generated a set of Model Fragments from the solution Model, mapping each Fragment to a single Requirement. Thus, we were provided with a set of Requirements, the Model Fragments that implement them, and the TLR mapping between both artifacts.

CAF: Regarding our industrial partner, CAF provided us with their existing documentation on Requirements to Process Models Traceability, where each requirement is also mapped to a single Model Fragment.

In both cases, we use the existing Traceability as the oracle for evaluating the outcomes of each of the three approaches. To do so, we compare the Model Fragments generated for each Requirement by the three of them against the oracle Model Fragment (ground truth Model Fragment) for said Requirements.

3.5 Implementation Details

We have used three libraries to implement the different approaches taken in account through this work: (1) to load and process the Process Models in both case studies, we used the Camunda BPMN Model API [16], (2) to develop the Natural Language Processing operations in our approaches, we have used the OpenNLP Toolkit for the Processing of Natural Language Text [17], and (3) to perform the LSI and SVD carried out in the Aggregation and Mutation Search techniques, the Efficient Java Matrix Library (EJML) was used [18]. For the evaluation, we used a Lenovo E330 laptop, with a processor Intel(R) Core(TM) i5-3210M@2.5 GHz with 16 GB RAM and Windows 10 64-bit.

4 Results

Table 1 outlines the results of the three studied approaches. Each row shows the Precision, Recall, F-measure, and MCC values obtained through each technique.

The Mutation Search technique achieves 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.

In contrast, both the Linguistic technique and the Aggregation technique present worse results in all the measurements: the Linguistic technique attains a mean precision value of 40%, a mean recall value of 35%, a combined F-measure of 33%, and an MCC value of 0.25 for the Camunda BPMN for Research case study, and a mean precision value of 35%, a mean recall value of 35%, a combined F-measure of 33%, and an MCC value of 0.25 for the CAF case study; and the Aggregation technique attains a mean precision value of 56%, a mean recall value of 72%, a combined F-measure of 61%, and an MCC value of 0.52 for the

Table 1. Mean values and standard deviations for Precision, Recall and F-measure for the three approaches

	Precision	Recall	F-measure	MCC
Linguistic - Camunda	$40\% \pm 25\%$	$35\% \pm 22\%$	$33\% \pm 13\%$	0.25 ± 0.19
Linguistic - CAF	$35\% \pm 28\%$	$35\% \pm 10\%$	$30\% \pm 7\%$	0.18 ± 0.13
Aggregation - Camunda	$56\% \pm 18\%$	$72\% \pm 22\%$	$61\% \pm 17\%$	0.52 ± 0.24
Aggregation - CAF	$69\% \pm 29\%$	$66\% \pm 17\%$	$64\% \pm 17\%$	0.58 ± 0.21
Mutation Search - Camunda	$63\% \pm 21\%$	$77\% \pm 22\%$	$68\% \pm 19\%$	0.60 ± 0.24
Mutation Search - CAF	$79\% \pm 19\%$	$72\% \pm 19\%$	$74\% \pm 16\%$	0.69 ± 0.20

Camunda BPMN for Research case study, and a mean precision value of 69%, a mean recall value of 66%, a combined F-measure of 64%, and an MCC value of 0.58 for the CAF case study.

5 Discussion

The Linguistic technique depends strongly on the language of the Requirements and Models: for a link to be produced between a Requirement a Model Element, exact patterns of words must be atomically matched through the rules. If a single word in a pattern found in a Requirement is different (or missing) in the Model, the rule does not trigger and the link is not produced. On the other hand, in the Aggregation and Mutation Search techniques the atomicity of text patterns is abandoned in favor of the semantic similitude of individual terms. This issue can be illustrated through an example. Consider the Requirement *'The system will open the doors'*, and a Model where the term *'system'* has been swapped for the more technical term *'PLC'*. Due to the vocabulary mismatch, the Linguistic technique would never find the pattern, and thus could never generate the links between Requirement and Model. On the other hand, our techniques would flag the occurrences of the terms *'open'* and *'doors'* in the corresponding Model Elements or Fragments, leading to a potential finding of links.

Moreover, Model Elements with little or no text appear often in Process Models, mainly in the form of flows and sometimes in the form of events. These elements can never be retrieved by the Linguistic technique: since there are no words, there is no pattern that can be matched. They are not retrieved by the Aggregation technique either: they tend to be at the bottom of the ranking produced by LSI since for these elements, all the *term* occurrences are equal to zero and thus, no correlation can be found with the *query* Requirement. However, in the Mutation Search technique, the algorithm does add these Elements to the candidate Fragments. Moreover, the addition of these Elements does not penalize the results technique, since the *term* occurrences are not altered in any way by them. Therefore, the candidate Fragments are more correct and complete, which leads the technique to better Precision and Recall results.

Finally, we also identified certain Process Models particularities that, if leveraged, would improve our Traceability techniques. Some examples of these particularities are: (1) the usage of the term 'if' in a Requirement almost always indicates the presence of an associated gateway in the Process Model, (2) the usage of the terms 'start' or 'end' usually denote events of the same type, (3) questions are often related with gateways in the Models, (4) verbs appear mostly on tasks, or (5) a noun that is often repeated at the start of multiple requirements may be the subject that carries an action (and thus, may appear in the Model as a lane). By studying the patterns of the Process Models language, it could be possible to take in account these particularities in our techniques (by, for instance, weighing the Model Elements accordingly or forcing their appearance), leading them to enhanced Traceability results.

6 Threats to Validity

In this section, we use the classification of threats to validity of [19] to acknowledge the limitations of our approach.

Construct validity: To minimize this risk, our evaluation is performed using four measures: Precision, Recall, F-measure, and MCC. These measures are widely accepted in the software engineering research community.

Internal Validity: The number of requirements and Process Models presented in this work may look small, but they represent a wide scope of different scenarios in an accurate manner.

External Validity: Both Natural Language Descriptions and Business Process Models are frequently leveraged to specify all kinds of different Business Processes. The Camunda Process for Research case study provides different examples from radically different domains. In addition, the real-world CAF Process Models used in our research are a good representative of the railway, automotive, aviation, and general industrial manufacturing domains. Our approach does not rely on the particular conditions of any of those domains. Nevertheless, our results should be replicated with other case studies before assuring their generalization.

Reliability: To reduce this threat, the requirements and Process Models used in our approach were taken from an open-source case study and from an industrial case study. None of the authors of this work was involved in the generation of said data.

7 Related Work

Related works focus on the impact and application of Linguistic techniques to TLR problem resolution at several levels of abstraction. Works like [20,21] or [22], among many others, use Linguistic approaches to tackle specific TLR problems and tasks. In [23], the authors use Linguistic techniques to identify equivalence

between Requirements, also defining and using a series of principles for evaluating their performance when identifying equivalent Requirements. The authors of [23] conclude that, in their field, the performance of Linguistic techniques is determined by the properties of the given dataset over which they are performed. They measure the properties as a factor to adjust the Linguistic techniques accordingly, and then apply their principles to an industrial case study. The work presented in [24] uses Linguistic techniques to study how changes in Requirements impact other Requirements in the same specification. Through the pages of their work, the authors analyze TLR between Requirements, and use Linguistic techniques to determine how changes in requirements must propagate.

Our work differs from [20–22,25] since our approach is not based or focused on Linguistic techniques as a means of TLR analysis, but we rather propose novel techniques to perform TLR between Requirements and Process Models, using a Linguistic technique only as a baseline against which our work is compared. Moreover, we do not study how Linguistic techniques must be tweaked for specific problems as [23] does. In addition, differing from [24], we do not tackle changes in Requirements nor TLR between Requirements, but instead focus our work on TLR between Requirements and Process Models.

Finally, other works target the application of LSI to TLR tasks. De Lucia et al. [26] present a tool based on LSI in the context of an artifact management system. [27] takes in consideration the possible configurations of LSI when using the technique for TLR between Requirements artifacts. In their work, the authors state that the configurations of LSI depend on the datasets used, and they look forward to automatically determining an appropriate configuration for LSI for any given dataset. 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 Process Models.

8 Conclusions

Traceability Links Recovery (TLR) has been a topic of interest for many years, but its study is an issue that has not received enough attention yet in the field of Process Models. Through this paper, we have studied TLR between Natural Language Requirements and Process Models through three different approaches: a Linguistic approach based on rules, specific from Models (which acts as a baseline for our work), and two techniques (Aggregation and Mutation Search) that we proposed and which we based on Latent Semantic Indexing, a technique that has been used successfully over code. The retrieved TLR results can be utilized by software engineers as a starting point for the development of their solutions.

The three approaches were evaluated by applying them to an academia set of Process Models, and to a set of Process Models from a real-world industrial case study with our industrial partner, CAF, a worldwide manufacturer of railway solutions. Results show that our techniques retrieve better results that the baseline Linguistic technique in both case studies. Through this work, we analyzed

why this is the case, and identified some particularities of Process Modeling that could be used in order to improve our techniques in future iterations of our work.

Acknowledgements. This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the project Model-Driven Variability Extraction for Software Product Line Adoption (TIN2015-64397-R). We also thank ITEA3 15010 REVaMP2 Project.

References

1. Gotel, O.C., Finkelstein, C.: An analysis of the requirements traceability problem. In: Proceedings of the First International Conference on Requirements Engineering, pp. 94–101. IEEE (1994)
2. Spanoudakis, G., Zisman, A.: Software traceability: a roadmap. Handb. Softw. Eng. Knowl. Eng. **3**, 395–428 (2005)
3. Watkins, R., Neal, M.: Why and how of requirements tracing. IEEE Softw. **11**(4), 104–106 (1994)
4. Ghazarian, A.: A research agenda for software reliability. In: 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 Trans. Softw. Eng. **43**(8), 777–797 (2017)
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 Trans. Reliab. **63**(4), 913–926 (2014)
7. Rubin, J., Chechik, M.: A survey of feature location techniques. In: Domain Engineering, pp. 29–58. Springer, Heidelberg (2013)
8. Spanoudakis, G., Zisman, A., Pérez-Minana, E., Krause, P.: Rule-based generation of requirements traceability relations. J. Syst. Softw. **72**(2), 105–127 (2004)
9. Landauer, T.K., Foltz, P.W., Laham, D.: An introduction to latent semantic analysis. Discourse Processes **25**(2–3), 259–284 (1998)
10. 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, pp. 216–223 (2003)
11. 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, vol. 1, pp. 83–86. Citeseer (2004)
12. Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J.: An information retrieval approach to concept location in source code. In: Proceedings of the 11th Working Conference on Reverse Engineering, pp. 214–223, November 2004
13. Salman, H.E., Seriai, A., Dony, C.: Feature location in a collection of product variants: combining information retrieval and hierarchical clustering. In: The 26th International Conference on Software Engineering and Knowledge Engineering, pp. 426–430 (2014)
14. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill Inc., New York (1986)
15. Leech, G., Garside, R., Bryant, M.: CLAWS4: the tagging of the British National Corpus. In: Proceedings of the 15th Conference on Computational Linguistics, vol. 1, pp. 622–628. Association for Computational Linguistics (1994)

16. Camunda: Camunda BPMN Model API (2017). https://github.com/camunda/camunda-bpmn-model. Accessed 3 Nov 2017
17. Apache: OpenNLP Toolkit for the Processing of Natural Language Text (2017). https://opennlp.apache.org/. Accessed 12 Nov 2017
18. Abeles, P.: Efficient Java Matrix Library (2017). http://ejml.org/. Accessed 9 Nov 2017
19. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Heidelberg (2012)
20. Sultanov, H., Hayes, J.H.: Application of swarm techniques to requirements engineering: requirements tracing. In: 18th IEEE International Requirements Engineering Conference (2010)
21. Sundaram, S.K., Hayes, J.H., Dekhtyar, A., Holbrook, E.A.: Assessing traceability of software engineering artifacts. Requirements Eng. **15**(3), 313–335 (2010)
22. Duan, C., Cleland-Huang, J.: Clustering support for automated tracing. In: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (2007)
23. Falessi, D., Cantone, G., Canfora, G.: Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. Trans. Softw. Eng. **39**(1), 18–44 (2013)
24. 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)
25. Ryan, K.: The role of natural language in requirements engineering. In: Proceedings of IEEE International Symposium on Requirements Engineering (1993)
26. 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, pp. 306–315. IEEE (2004)
27. 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)