Leveraging Execution Traces to Enhance Traceability Links Recovery in BPMN Models

Raúl Lapeña[a,*], Francisca Pérez[a], Óscar Pastor[b], Carlos Cetina[a]

[a]*SVIT Research Group, Universidad San Jorge*
Autovía A-23 Zaragoza-Huesca Km. 299, 50830, Zaragoza, Spain
[b]*PROS Research Centre, Universitat Politècnica de València*
Camí de Vera, s/n, 46022 València, Spain

Abstract

Context: Traceability Links Recovery has been a topic of interest for many years, resulting in techniques that perform traceability based on the linguistic clues of the software artifacts under study. However, BPMN models tend to present an overall lack of linguistic clues when compared to code-based artifacts or code generation models. Hence, TLR becomes a harder task when performed among requirements and BPMN models.

Objective: This paper proposes a novel approach, called METRA, that leverages the execution traces of BPMN to expand the BPMN models. The expansion of the BPMN models enhances their linguistic clues, bridging the language between BPMN models and other software artifacts, and improving the TLR process between requirements and BPMN models.

Method: The proposed approach is evaluated through a real-world industrial case study, comparing its outcomes against two state-of-the-art baselines, TLR and LORE. The paper also evaluates the combination of METRA with LORE against the rest of the approaches, including standalone METRA. The evaluation process generates a report of measurements (precision, recall, f-measure, and MCC), over which a statistical analysis is conducted.

Results: Results show that approaches based on METRA maintain the excellent precision results obtained by baseline approaches (74.2% for METRA, 78.8% for METRA+LORE), whilst also improving the recall results from the unacceptable values obtained by the baselines to good values (72.4% for METRA, 73.9% for METRA+LORE). Moreover, according to the statistical analysis, the differences in the results obtained by the evaluated approaches are statistically significant.

Conclusions: This paper opens a novel field of work in TLR by analyzing the improvement of the TLR process through the inclusion of linguistic clues present in execution traces, and discusses ideas for further research that can delve into this promising direction explored by our work.

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

*Corresponding author.
Email addresses: rlapena@usj.es (Raúl Lapeña), mfperez@usj.es (Francisca Pérez), opastor@pros.upv.es (Óscar Pastor), ccetina@usj.es (Carlos Cetina)

1. Introduction

Model-Driven Development (MDD) [1] is a software practice where requirements, understood as natural language representations of the specifications of a system [2], are used to build models that are then transformed into source code or interpreted at runtime. Major players in the software engineering field and in the requirements engineering field foresee a broad adoption of MDD [3, 4], since MDD techniques improve the productivity, quality, and performance of software in industrial scenarios that demand more abstract approaches than mere coding [1]. MDD has been applied with success to design novel approaches in model-based engineering [5], model-based Software Product Line (SPL) adoption [6], and feature-oriented engineering [7, 8] in several different domains.

Software engineers from our industrial partner, an international manufacturer in the railway domain, express system requirements in natural language, and use them to design BPMN models through the OMG's BPMN standard, a widespread model standard used to graphically represent processes [9]. The BPMN models are used to describe the interactions that occur between the humans and the trains, and also to design and derive other software artifacts following MDD practices and guidelines. However, in industrial MDD contexts such as the one from our industrial partner, companies tend to have a myriad of products with large and complex models behind, which are created and maintained over long periods of time by different software engineers, who often lack knowledge over the entirety of the product details. Under these conditions, maintenance activities consume high amounts of time and effort without guaranteeing good results.

Traceability Links Recovery (TLR), defined as the software engineering task that deals with the automated identification and comprehension of dependencies and relationships between software artifacts [10], is a key to success in these industrial scenarios. TLR is an important support activity for development, management, and maintenance of software, and is considered as a good practice by numerous major software standards such as CMMI or ISO 15504 [10]. Moreover, affordable TLR can be critical to the success of a project [11], and leads to increased maintainability and reliability of software systems by making it possible to verify and trace non-reliable parts [12], also decreasing the expected defect rate in developed software [13]. However, establishing and maintaining traceability links has proven to be a time consuming, error prone, and person-power intensive task [10, 14]. Therefore, automated TLR has been a subject of investigation for many years within the software engineering community [15, 16]. Moreover, in recent years, it has been attracting more attention, becoming a subject of both fundamental and applied research [17]. Nevertheless, while most of the works in the literature focus on performing TLR between requirements and code [18], the application of TLR techniques to models in general, and BPMN models in particular, is a topic that has not been thoroughly explored so far.

However, state-of-the-art automated TLR techniques rely greatly on the language and the syntactical, lexical, and semantical particularities of the software artifacts under study. For instance, Latent Semantic Indexing (LSI), which is the most popular TLR technique and the one that has yielded the best TLR results so far [18], is based on exploiting term similarities among the requirements and the software artifacts. However, BPMN models tend to present less terms and an overall lack of linguistic clues when compared to code-based artifacts or other models that are designed with code-generation purposes in mind, which tend to contain greater amounts of linguistic clues and natural language within their implementation. Since TLR techniques rely on the textual components of the artifacts under study, TLR becomes an ever harder task when performing TLR directly among requirements and BPMN models. Be that as it may, BPMN models count with an additional source of textual information that can be leveraged to improve the results of the process: the BPMN execution traces.

Through this work, we propose a novel approach called METRA (Model Expansion through TRAces) that minimizes the impact that the diminished amount of linguistic clues in BPMN has on the TLR process. To that extent, natural language processing

(NLP) techniques are used to process the requirements and the BPMN models, and then the BPMN execution traces are used as a means of expanding the linguistic clues of the processed BPMN models, bridging the gap between the language in use in the requirements and the BPMN models. Finally, TLR techniques are applied to analyze the requirements and the expanded BPMN models, in search for model fragments that match the requirements.

From our work, two research questions arise. We ought to study (1) whether METRA improves the results obtained by baseline techniques in a statistically significant manner, and (2) whether it is possible to combine METRA with other expansion-based TLR techniques for enhanced results. We have evaluated these questions through the requirements and BPMN models that comprise a real-world industrial case study, involving the control software of the trains manufactured by one of our industrial partners.

Results show that approaches based on METRA maintain the excellent precision results obtained by baseline approaches (78.8% on average), whilst also improving the recall results from the unacceptable values obtained by the baselines to good values (73.9% on average). Through the analysis of the obtained results, we discuss how leveraging the linguistic clues in execution traces improves the TLR process between requirements and BPMN models, so that further research can delve into this promising direction explored by our work.

Through the following pages, Section 2 presents the background for our work. Section 3 provides details on our approach. Section 4 describes the evaluation of our approach. Sections 5 and 6 present the obtained results and discuss the outcomes of our work. Section 7 presents the threats to the validity of our work. Section 8 reviews works related to this one. Section 9 concludes the paper.

2. Background

Our industrial partner is a worldwide provider of railway solutions. Their trains can be seen all over the world in different forms. Train units are furnished with multiple pieces of equipment that carry out specific tasks for the train. The control software of the train unit is in charge of making all the equipment cooperate to achieve the train functionality while guaranteeing compliance with the specific regulations of each country. Our industrial partner uses BPMN models to describe processes that are carried out between the humans and the main pieces of equipment installed in a train unit, and as part of their MDD practices to design and derive other software artifacts. Lately, our industrial partner has been focusing some efforts on performing Traceability Links Recovery (TLR) between their natural language requirements and their BPMN models. As stated in the introduction section, manual TLR is a nearly infeasible task, and TLR between natural language requirements and BPMN models is no exception.

Figure 1 motivates the difficulty of the task at hand through an excerpt taken from the email vote diagram, extracted from the email voting system example found within the BPMN examples available on the BPMN standard official website[1]. The model represents a process for defining a list of issues on which votes must be taken. This process might be extracted by hand from the model, but the amount of elements shown in the model, along with their positioning and connections cause this to be a quite complex task, even without taking in consideration the full model or the potential loopbacks and implications of the outcomes of this sub-process into the rest of the modeled behaviours. Considering the difficulty of extracting information from the academic model presented in Figure 1, it is possible to better understand the complexity of achieving similar feats over more complex real-world industrial models.

In industrial scenarios as the one from our industrial partner, companies tend to have a myriad of products with large and complex models behind, created and maintained over long periods of time by different software engineers, who often lack knowledge over the entirety of the product details. The complexity of the BPMN models and the number of elements in place render manual TLR virtually impossible to attain. As an example, imagine that we try to manually trace the requirements to the model elements

[1] http://www.bpmn.org/

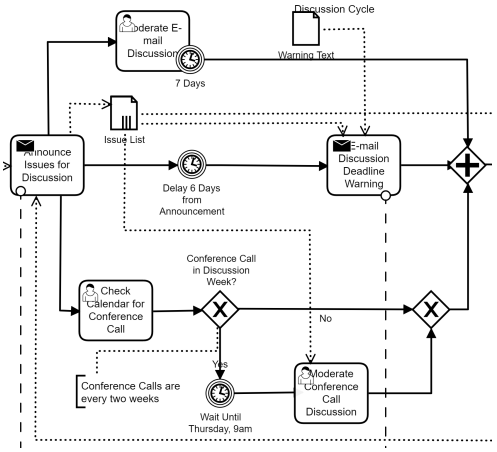

Figure 1: Excerpt taken from the e-mail voting system diagram, an example found within the official BPMN standard website

that comprise the data set provided by our industrial partner for this research. The data set is conformed by 140 test cases, with one requirement per test case and BPMN models that are implemented by an average of 650 model elements. In order to trace a particular requirement to a model fragment, a domain expert would need to examine the full model and decide which elements trace the requirement correctly. Assuming that the domain expert must spend around 5 seconds to take the decision with each element [19], creating the fragment that retrieves the traceability to a requirement would take slightly less than one hour. Thus, tracing the total 140 requirements by hand would take around 140 hours, which translate into 17 full-time working days.

Figure 2 depicts a simplified example of a real-world industrial BPMN model, taken from a real-world train, specified through a BPMN model. The model, which will be used throughout our paper as a running example, has the expressiveness required to describe the interaction between the pieces of equipment installed in a train unit, and also to describe non-functional aspects related to regulation. Specifically, the example of the figure presents the station stop process, where a human driver sets the train in stop mode in preparation for a station stop. The PLC that implements the system controls of the train checks whether the doors are open or closed and, in the event that the doors are closed, opens the passenger doors. The figure also shows an example requirement of the train ("at all the stops, the driver will set the train in stop mode"). The requirement is implemented within the context of the station stop process, as a subset of the BPMN model that depicts the process. Hence, we define a model fragment as the subset of elements of a BPMN model that implement a particular requirement. Thus, a model fragment, which can include any number of elements, can be mapped directly to a single requirement. In that sense, a full BPMN model can implement several requirements, and can thus be divided into several different model fragments that can be mapped to those requirements. The ultimate goal of our research, which is to automate the traceability between the requirements and BPMN models, can also be formulated as the automated retrieval of the mapping between the requirements and the model fragments that implement them. The elements of Figure 2 highlighted in light gray conform such an example model fragment, more precisely, the model fragment that is associated with the requirement shown in the figure. In the case of the example, the model fragment comprises the driver, the start event, and the stop mode action.

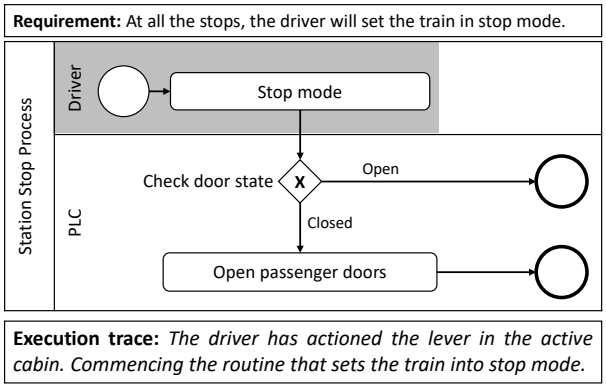

Figure 2: Example of Requirement, Model, and Model Fragment

The figure also shows a simplified example execution trace, associated with the actions represented in

the model fragment. The extraction of the execution traces is performed through an in-house solution implemented by the software engineers of our industrial partner, a common practice in all kinds of software scenarios [20, 21]. During the execution of a BPMN model, their scripts record the different actions that are performed in the trains and the actors that perform those actions in the model. These actions are logged directly in natural language in textual files that are used by technicians and engineers to analyse bugs and instruct newcomers in the internal functioning and architecture of the developed systems. As part of our ongoing research with our industrial partner, they have provided us with the textual files containing the natural language traces associated with the BPMN models of their case study.

In the figure, it is possible to appreciate that the model in Figure 2 contains a very little amount of textual information. To be precise, while the requirement contains 13 words, and the execution trace associated to the model fragment contains 20 words, the model as a whole contains just 15 words (including the name of the process). In comparison, the particular code that implements the requirement comprises around 500 lines of code, including valuable terms in the names of methods and variables and further information in the comments of the developers. The lack of linguistic clues and text in the BPMN model leads to an increase in the difficulty of the TLR process between the requirement and the model.

3. Approach

3.1. Approach Overview

Through the presented approach, named METRA (Model Expansion through TRAces), we tackle the impact that the lack of linguistic clues in BPMN models has on the TLR process between requirements and BPMN models.

To that extent, METRA leverages the linguistic clues in the BPMN models execution traces to expand the BPMN models, bridging the gap between the language used in requirements and BPMN models. The approach runs through three steps:

1 First, we extract model fragments from the processed BPMN model. This step allows METRA to generate a set of candidate solutions for requirement to BPMN model traceability.

2 Secondly, we convert the model fragments into text and use natural language processing (NLP) techniques to process the generated BPMN model fragments, the available input BPMN model execution traces, and the input requirements.

The NLP techniques unify the language of the software artifacts, and facilitate both the expansion process and the traceability process.

3 Afterwards, we leverage the processed BPMN model execution traces in order to expand the processed BPMN model fragments, improving their linguistic clues.

Finally, the processed requirement is used along with the expanded model fragments as an input for Latent Semantic Indexing (LSI) [22], a widely accepted TLR process [3]. Through LSI, the expanded model fragments are assessed in terms of linguistic similarity to the input requirement. Fig. 3 depicts an overview of the steps of the approach. In the figure, squared boxes represent the inputs and outputs of each step, while rounded boxes represent each step of the approach. The highlighted boxes represent the initially available inputs (BPMN model, BPMN model execution traces, and requirement) used for the different steps of our approach and for the TLR process, and the final output (the most relevant expanded model fragment for the requirement).

The following sections, along with figures 4 to 6, deal with the explanation of each of the steps of the METRA approach and their application to the running example presented in figure 2 in a greater level of detail.

3.2. Model Fragments Extraction

In the first step of METRA, model fragments are extracted from the processed BPMN model, generating a set of candidate solutions for requirement to model traceability. As stated in sections 1 and 2, industrial BPMN models are large and complex in

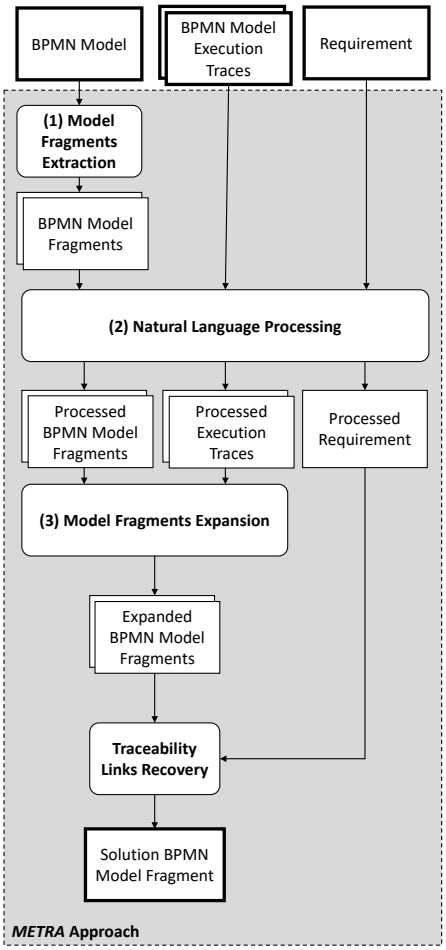

Figure 3: Approach

comparison with their academic counterparts, making extraction of the model fragments by hand unfeasible in practice. While it would be possible to apply Information Retrieval techniques at the level of model elements, such granularity would leave us with a problematic set of relevant model elements for the requirement-to-model traceability process, since (1) elements with no text would never be identified as relevant and (2) the identified elements in the set may not be adjacent in the model, coming from very different parts of the full BPMN. From these problematic elements, it would then be necessary to build an understandable, standard-compliant model fragment to propose as a result for traceability, which would then need manual revision, defeating the purpose of our research. In addition, with industrial BPMN models being large and complex, the search space of the model elements and their potential combinations for building the model fragments becomes also large and complex. Hence, in our approach, we aim to extract the model fragments through a mechanism that provides standard-compliant models, and which also takes into account the possibility of including elements with no text, all while being able to effectively explore the search space. To that extent, we use an algorithm with evolutionary characteristics (mutation of model fragments) that helps us generate populations of standard-compliant, coherent BPMN model fragments that can be automatically evaluated by Information Retrieval techniques afterwards.

METRA extracts model fragments from the BPMN model through Algorithm 1. In the algorithm, an empty population and a seed fragment (chosen randomly from the input BPMN 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 already existing fragments into the population.

Mutations 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), (3) adding a lane to the fragment, or (4) removing a lane from the fragment. The performed mutation is chosen randomly on each iteration. The mutation function is described in Algorithm 2.

Fig. 4 shows the BPMN model fragment extraction process. On the top part of the figure, it is possible to see the example input BPMN. For the sake of legibility, understandability, and due to space reasons, the bottom part of the figure depicts an example subset of generated model fragments, extracted through the usage of the algorithm.

Algorithm 1 Model fragment extraction algorithm

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

Algorithm 2 Mutation function

```
M ← chooseMutation()                                ▷ Randomly choose a mutation
switch M do                                         ▷ Perform action depending on chosen mutation
    case addElement
        F ← addElement(F)                           ▷ Add connected event/gateway/task
    case removeElement
        F ← removeElement(F)                        ▷ Remove element with only one connection
    case addLane
        F ← addLane(F)                              ▷ Add a lane
    case removeLane
        F ← removeLane(F)                           ▷ Remove a lane
return F                                            ▷ Return the modified fragment
```

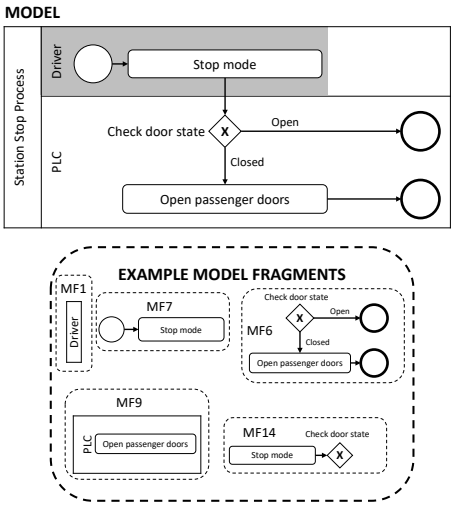

Figure 4: Extraction of BPMN model fragments

3.3. Natural Language Processing

The model fragments in the population generated by the algorithm presented in the first step of METRA are then transformed into their natural language representation through the technique presented by Meziane et. al. [23]. Meziane et. al. propose a set of linguistic rules and a tagging system to build natural language sentences from the texts extracted from the elements that compose UML class models. In order to transform our BPMN model fragments to their natural language representations, we extract the text of the model elements that compose the BPMN model fragments, and then apply the linguistic rules and tagging system proposed by Meziane et. al. to build such natural language representations. We consider lanes as nouns (e.g. the 'PLC' noun would be the actor that is performing an action) and actions as verbal clauses (e.g. the 'open passenger doors' action contains the verb 'open' plus

an associated noun, 'passenger doors'), allowing us to build complete natural language sentences of the form noun + verbal clause following the aforementioned rules and tagging system (e.g. 'The PLC opens the passenger doors').

Afterwards, as the second step of METRA, we process the natural language representations of the extracted BPMN model fragments, the available BPMN model execution traces, and the input requirements through Natural Language Processing (NLP) techniques. This is often seen as a beneficial process in all manners of software engineering tasks [24]. In our approach, the processing step serves as a means of unifying the language of the software artifacts, which in turn facilitates the expansion process and the traceability process alike. The techniques in use are syntactical analysis, root reduction, and human NLP. Fig. 5 is used to illustrate, on two example execution traces (T1 and T2) the whole compendium of NLP techniques used by METRA to process the software artifacts (BPMN model fragments, execution traces, and requirements).

1 **Syntactical Analysis:** Syntactical Analysis (SA) techniques analyze the specific roles of each word in a sentence, determining their grammatical function. These techniques (often referred to in the literature as Parts-Of-Speech tagging, or POS tagging) allow engineers to implement filters for words that fulfill specific grammatical roles in a requirement, usually opting only for nouns [25]. In Fig. 5, it is possible to appreciate the SA process, with the POS tagged tokens associated to each of the example execution traces as outcome.

2 **Root Reduction:** The technique known as lemmatizing reduces words to their semantic roots, also known as lemmas. Thanks to lemmas, the language is unified, avoiding verb tenses, noun plurals, and several other word forms that can interfere negatively with the TLR process. The unification of the language semantics is an evolution over pure syntactical role filtering, allowing for a more advanced filtering of words. In Fig. 5, it is possible to appreciate the root reduction process, with the root-reduced tokens as outcome of the semantic analysis of the POS tags derived from the example execution traces (keeping only nouns).

3 **Human NLP:** The inclusion of domain knowledge through experts and software engineers in the TLR process is regarded as beneficial. Human NLP is often carried out through domain terms extraction or stopwords removal. METRA searches the software artifacts for domain terms provided by software engineers, and adds the found domain terms to the processed artifact. On the other hand, stopwords are filtered out after root reduction, using a list provided by the software engineers. Figure 5 depicts the Human NLP process, where a software engineer provides both lists of terms, which are consequently introduced into the final query, or filtered out of it.

3.4. Model Fragments Expansion

In the last step of METRA, the processed BPMN model fragments are enriched with information from the processed BPMN execution traces. To that extent, METRA uses a technique that is based on Rocchio's method [26], which is perhaps the most commonly used method for query reformulation [27]. Rocchio's method orders the terms in a set of documents based on the importance of each term in the documents, and then uses those terms to expand another artifact.

However, Rocchio's method orders all the terms in all the available documents according to their relevancy in the documents. In other words, if we use the full set of processed BPMN model execution traces to expand the processed BPMN model fragments, we end up including the same terms in all the processed BPMN model fragments, altering the linguistic clues of all the processed BPMN model fragments equally and thus cancelling any potential traceability improvements. Therefore, prior to the expansion of one particular processed BPMN model fragment, it is first required to prune the set of execution traces, leaving only the relevant processed BPMN model execution traces for the expansion of the particular processed BPMN model fragment. To that extent, we

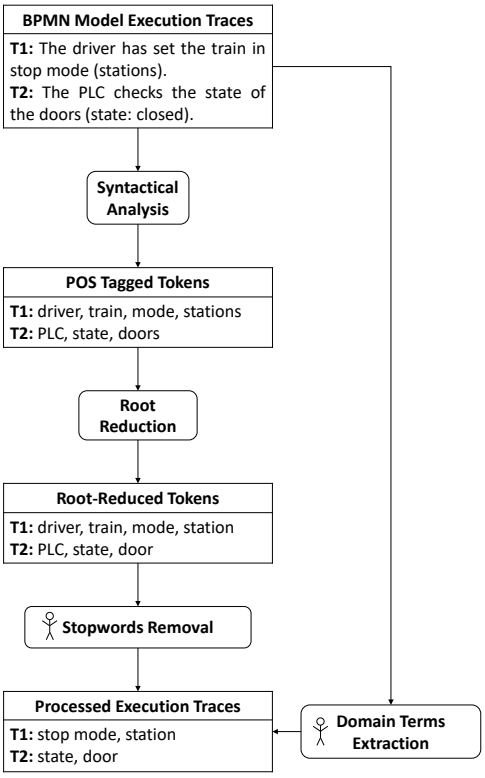

Figure 5: Natural Language Processing Techniques

first search the population of processed BPMN model execution traces for those processed BPMN model execution traces that have term coincidences with the processed BPMN model fragment.

Aftwerwards, we use Rocchio's method for expanding the processed BPMN model fragment. Rocchio's method orders the terms in the execution traces based on the sum of the importance of each term using the following equation:

$$Rocchio = \sum_{d \in R} TF(t,d) \cdot IDF(t,R) \qquad (1)$$

Where R is the set of processed BPMN model execution traces, d is a document in R (that is, one processed BPMN execution trace), and t is one term in d.

The first component of the measure is the Term Frequency (TF), which is the number of times the term appears in a document; it is an indicator of the importance of the term in the document compared to the rest of the terms in that document. The second component is the Inverse Document Frequency (IDF), which is the inverse of the number of documents that contain that term; it indicates the specificity of that term for a document that contains it. The IDF measurement is calculated as:

$$IDF(t,R) = \log \frac{|R|}{|\{d \in R : t \in d\}|} \qquad (2)$$

Where $|R|$ is the number of documents and $|\{d \in R : t \in d\}|$ is the number of documents where the term is present.

Using Rocchio's method, the terms of the processed BPMN model execution traces associated to the processed BPMN model fragment that is being expanded are ordered from highest to lowest sum of importance into a single document of terms. Once ordered, we take in consideration only the first 10 suggested terms and discard the rest, as is recommended in the literature [28]. The terms are concatenated into the processed BPMN model fragment, effectively expanding its linguistic clues. Since the aim of our approach is to enhance the processed BPMN model fragment by expanding it with new linguistic clues, we expand the fragment only with those suggested terms that do not already appear in it. The expansion process is repeated for all the processed BPMN model fragments, until all of them are expanded. The expanded BPMN model fragments are the ultimate goal that METRA seeks to obtain. Along with the processed requirement, they are used as query for the Traceability Links Recovery process.

3.5. Traceability Links Recovery

METRA can be applied to any TLR technique. However, in our work, we utilize Latent Semantic Indexing (LSI), the TLR technique that obtains the best results when performing TLR between requirements and software artifacts [3]. Latent Semantic Indexing (LSI) [22] constructs vector representations of a query and a corpus of text documents by encoding them as a *term-by-document co-occurrence matrix*.

Once the matrix is built, it is normalized and decomposed into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [22]. Finally, the similarity degree between the query and each document is calculated through the cosine between the vectors that represent them. Scores closer to 1 denote a high degree of similarity, and scores closer to -1 denote a low degree of similarity. The document with the score closest to 1 will be the most similar document to the query.

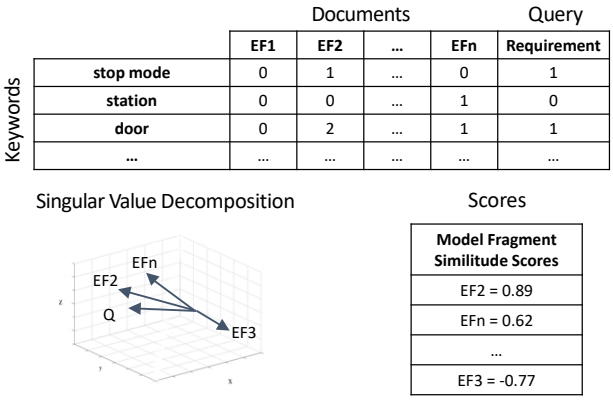

Figure 6: Traceability Link Recovery through Latent Semantic Indexing

Figure 6 shows an example of the LSI process carried out in our approach. The top part of the figure shows the *term-by-document co-occurrence matrix* associated with our running example. For the sake of simplicity, the matrix shows only a subset of the terms and a subset of the documents involved in the creation of the matrix, along with the query. In our approach, *terms* are each of the keywords that compose the processed requirement and expanded BPMN model fragments (e.g. the term 'station'), *documents* are the expanded BPMN model fragments resulting from the process described in section 3.4 (e.g. an expanded model fragment might result in the set of terms ['station', 'door']), and the *query* is the processed requirement for which we want to find the associated model fragment (e.g. a processed requirement might result in the set of terms ['stop mode', 'door']). Each cell in the matrix contains the frequency with which the *term* of its row appears in the *document* denoted by its column. The bottom left part of the figure shows the result of applying the SVD technique to the matrix (i.e. the vectors associated to each of the expanded fragments and query that appear in the figure). The bottom right part of the figure shows the resulting similarity scores associated with each *document*. The first *document* in the ranking is the expanded BPMN model fragment that is most similar to the processed requirement, and is thus proposed as the solution for the TLR process.

4. Evaluation

This section presents the evaluation of our approach, including the research questions that arise from our work, the experimental setup devised to respond to the research questions, a description of the case study where we applied the evaluation, and the implementation details of our approach.

4.1. Research Questions

From our work, two research questions arise:

RQ$_1$ *Does METRA improve the results obtained by baseline techniques in a statistically significant manner?*

RQ$_2$ *Is it possible to combine METRA with other expansion-based TLR techniques to further enhance TLR results?*

According to the principles presented in [29], in order to test that a particular piece of research satisfies its goals, it is necessary to falsify the opposite. In other words, it becomes necessary to formulate the opposite hypothesis (i.e.: the approach is not able to satisfy the intended goal), and to be able to reject the formulated opposite hypothesis. The opposite hypothesis is known as the null hypothesis. According to [29], conclusions about the validity of the research and the satisfaction of the intended goals by the approach can be drawn only after rejecting the null hypothesis. Therefore, towards our research questions, we formulate the following null hypotheses:

H$_1$ *METRA does not improve the results of baseline techniques in a statistically significant manner.*

H_2 *It is not possible to combine METRA with other expansion-based TLR techniques to further enhance TLR results.*

Through our work, we aim to falsify the null hypotheses, in order to test that METRA improves the results of basic baseline techniques and of other expansion-based TLR techniques, and that METRA can be combined with other expansion-based TLR techniques to further improve the outcomes for the TLR process over BPMN models. To that extent, we have based the evaluation of our work through the requirements and BPMN models that comprise a real-world industrial case study, involving the control software of the trains manufactured by our industrial partner. The rest of this section is devoted to the definition of the evaluation process.

4.2. Experimental Setup

The goal of this experiment is to perform TLR between requirements and BPMN models through METRA. We compare the results of METRA against (1) the results obtained by a basic TLR baseline and (2) the results obtained by another expansion-based TLR baseline. The baselines against which we compare our work are (1) plain TLR through LSI without expansion, and (2) LORE [30], a TLR technique that expands the requirements through a domain ontology prior to performing LSI. In addition, we explore the results of combining both METRA with LORE, that is, expanding at the same time both the BPMN models through the linguistic clues in the execution traces and the requirements through a domain ontology.

Fig. 7 shows an overview of the process followed to perform the evaluation. The top part of the figure shows the inputs, as provided by our industrial partner, which are used to build the test cases. The approved traceability is used to build the oracles against which the results of the different approaches are compared.

For each test case, all of the approaches generate one model fragment each. The model fragments generated for each test case are compared against their respective oracles (ground truth), calculating a confusion matrix in the process.

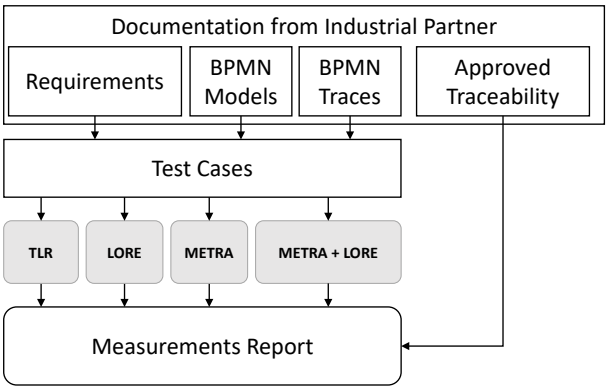

Figure 7: Experimental Setup

A confusion matrix is a table often used to describe the performance of a classification model on a set of test data (the best solutions) for which the true values are known (from the oracle). In our case, each solution obtained is a model fragment composed of a subset of the model elements that are part of the model. Since the granularity is at the level of model elements, each model element presence or absence is considered as a classification. The confusion matrix distinguishes between the predicted values and the real values classifying them into four categories:

- True Positive (TP): values that are predicted as true (in the solution) and are true in the real scenario (the oracle).
- False Positive (FP): values that are predicted as true (in the solution) but are false in the real scenario (the oracle).
- True Negative (TN): values that are predicted as false (in the solution) and are false in the real scenario (the oracle).
- 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, we report four performance measurements

for each baseline and approach: recall, precision, F-measure and the Matthews Correlation Coefficient (MCC) [31, 32].

Recall measures the number of elements of the solution that are correctly retrieved by the proposed solution and is defined as follows:

$$Recall = \frac{TP}{TP + FN}$$

Precision measures the number of elements from the solution that are correct according to the ground truth (the oracle) and is defined as follows:

$$Precision = \frac{TP}{TP + FP}$$

F-measure corresponds to the harmonic mean of precision and recall and is defined as follows:

$$F\text{-}measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

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% (no single model element from the oracle is present in the retrieved fragment) to 100% (all the model elements from the oracle are present in the retrieved fragment). Precision values can range between 0% (no model elements from the retrieved fragment appear in the oracle) to 100% (all the model elements from the retrieved fragment appear in the oracle).

Precision and recall results can be classified in different categories, according to the quality of the obtained results. In particular, the quality in terms of precision and recall can be classified as non-acceptable, acceptable, good and excellent [33, 34]. Table 1 presents the reference values for such a classification.

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, an MCC value of 0 corresponds to a random prediction, where the correlation between the predicted values and the solution has been attained by chance.

Classification	Precision	Recall
Non-acceptable	<20%	<60%
Acceptable	20%-29%	60%-69%
Good	30%-49%	70%-79%
Excellent	50%-100%	80%-100%

Table 1: Precision and recall classification

4.3. Case Study

The case study where we applied our approach was provided by our industrial partner, CAF[2], a worldwide provider of railway solutions. Our evaluation includes 140 test cases, with each test case comprising one requirement, one execution trace, one model, and the approved traceability between the requirement and the model. The requirements have about 25 words on average, the execution traces comprise about 320 words on average, and the models are formed through 650 elements on average.

4.4. Implementation details

A prototype of our approach can be found online along with the results presented in this paper at `https://bitbucket.org/svitusj/lore-metra/`. For the development of the NLP operations used in both our approach and the baseline, we have used the OpenNLP Toolkit [35]. To implement the LSI and SVD techniques, the Efficient Java Matrix Library (EJML) was used [36]. For the evaluation, we used a HP ProBook 440 G5 laptop, with a processor Intel(R) Core(TM) i5-7200U @ 2.5GHz with 8GB RAM and Windows 10 (64-bit).

5. Results

This section of the paper is divided into three main parts: Section 5.1 describes the results, Section 5.2 statistically analyzes the results to test the presented null hypotheses, and Section 5.3 summarizes the outcomes of our research.

[2] `http://www.caf.es/en`

5.1. Descriptive statistics

5.1.1. Results values

Table 2 outlines the results of the TLR baseline, the LORE baseline, our METRA approach, and the combination of our METRA approach with the LORE baseline. Each row shows the precision, recall, F-measure, and MCC values obtained through each of the approaches. The best values of each category are highlighted in the table in light grey.

	Precision	Recall	F-measure	MCC
TLR baseline	59.3%±29.6%	45.5%±34.2%	52.4%±31.9%	0.31±0.13
LORE baseline	79.2%±33.6%	50.2%±30.6%	66.5%±38.6%	0.62±0.32
METRA	74.2%±14.2%	72.4%±15.9%	72.4%±13.4%	0.63±0.15
METRA+LORE	78.8%± 9.7%	73.9%±15.3%	75.6%±11.6%	0.66±0.13

Table 2: Mean Values and Standard Deviations for Precision, Recall, F-measure, and MCC

5.1.2. Results analysis

Fig. 8 shows the distribution of the results for the 140 test cases in each of the baseline approaches, the METRA approach, and the approach that combines METRA with LORE.

The figure depicts, in a graphical manner, the data provided in the table. In the figure, it is possible to appreciate that while standalone LORE achieves better precision results (albeit with a high standard deviation), approaches based on METRA attain improved results in terms of recall, which amounts for the improvements in F-measure and MCC. Specifically, in terms of F-measure, it is possible to appreciate that more than three quarters of the values for F-measure of the METRA approach are above the lower 50% values of standalone LORE, and that all the values of the approach that combines METRA with LORE are above the lower 50% values of standalone LORE.

Table 3 categorizes the values obtained by the approaches according to the classification of precision and recall results provided in table 1. Results show that the approaches that include METRA maintain the excellent precision obtained by baseline approaches, while also improving recall from the unacceptable values obtained by the baselines to good values. In particular, the increase in recall up to 73.9% leads the combination of METRA with LORE to obtain the best average results, with a 75.6% in F-measure and 0.66 in the MCC, being only surpassed in precision by the LORE baseline on a 0.4% margin, before accounting for the high standard deviation presented by the latter. In that sense, it is possible to appreciate from the results that the inclusion of METRA in TLR, either on its own or in combination with LORE, leads to decreases in standard deviation in all measurements in comparison with the baseline approaches.

	Precision	Recall
TLR baseline	Excellent	Non-acceptable
LORE baseline	Excellent	Non-acceptable
METRA	Excellent	Good
METRA+LORE	Excellent	Good

Table 3: Classification of Precision and Recall values for each of the approaches

5.2. Hypothesis testing

The obtained results look promising and indicate that approaches based on METRA improve the results obtained by the baselines in use. However, it is necessary to address whether the improvements are statistically significant. To assess whether there are significant differences in performance between the baselines and approaches based on METRA, their results must be properly compared through statistical methods, following the guidelines presented in [37]. The goals of the statistical analysis are twofold: (1) provide formal evidence that approaches based on METRA do in fact have an impact on the comparison measurements, and (2) show that the differences are significant in practice. To enable statistical analysis, all configurations should be run a large enough number of times independently to collect information on the probability distribution. A statistical test should then be run to assess whether there is enough empirical evidence to claim that there are differences between the configurations.

5.2.1. Statistical methods

Null hypothesis $\mathbf{H}_1$, defined in Section 4 along with the research questions, states that METRA does not improve the results of baseline techniques in a statistically significant manner. The statistical tests aim

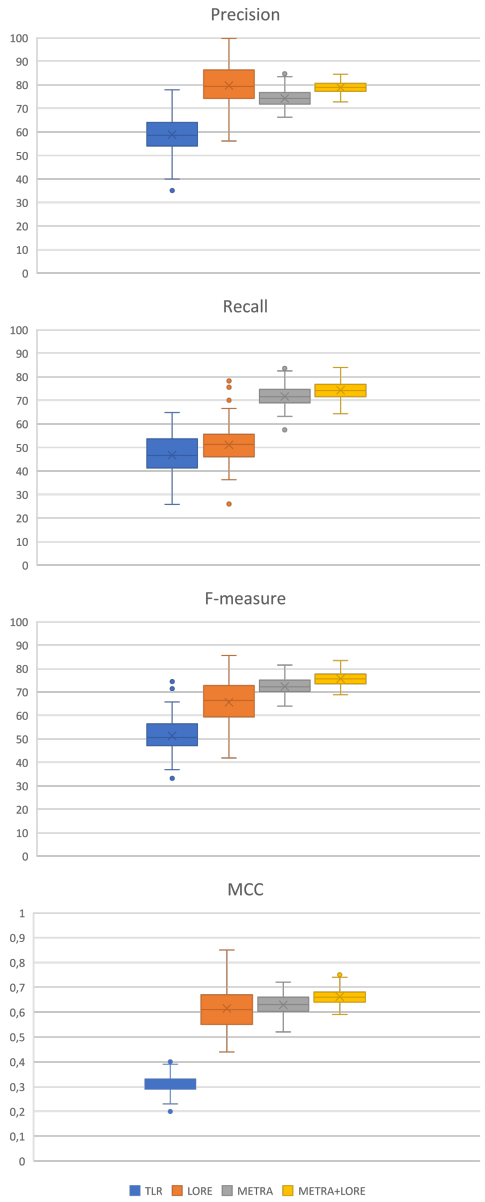

Figure 8: Distribution of precision, recall, F-measure and MCC for the baselines and METRA-based approaches

to falsify this null hypothesis. Statistical tests provide a probability value, the *p-value*, which can range in values from 0 to 1. The lower the *p-value* of a test, the more likely that the null hypothesis can be falsified. It is accepted by the research community that a *p-value* under 0.05 is statistically significant [37] towards falsifying the null hypotheses. The statistical test that must be followed depends on the properties of the data. Since our data does not follow a normal distribution, our analysis requires the usage of non-parametric techniques. Of the several available tests for analyzing this kind of data, the Quade test is more powerful than other tests when working with real data [38], and according to [39], has shown better results than other tests when the number of algorithms is low (no more than 4 or 5 algorithms).

However, statistically significant differences can be obtained even when they are so small as to be of no practical value. *Effect size* measurements are needed to analyze this factor. For a non-parametric effect size measure, we use Vargha and Delaney's $\hat{A}_{12}$ [40]. $\hat{A}_{12}$ measures the probability that running one approach yields higher values than running another approach. With the $\hat{A}_{12}$ statistic, the approaches are compared in pairs. If the $\hat{A}_{12}$ statistic obtains a value greater than 0.5, the comparison will be in favor of the first approach in the pair. If the $\hat{A}_{12}$ statistic obtains a value lesser than 0.5, the comparison will be in favor of the second approach of the pair. If the two approaches are equivalent, then the $\hat{A}_{12}$ statistic will obtain a value of 0.5. This can be better illustrated through a few examples:

- A value of $\hat{A}_{12} = 0.52$ means that on 52% of the runs, the first of the pair of compared approaches would obtain better results than the second approach of the pair.
- A value of $\hat{A}_{12} = 0.24$ means that on 76% of the runs, the second of the pair of approaches would obtain better results than the first approach of the pair.

5.2.2. Statistical analysis results

Regarding statistical significance, the Quade test returns a *p-value* of 0.00204 for precision and a *p-value* of $1.4x10^{-16}$ for recall. Since the values obtained by the Quade test are all below the 0.05 threshold, we can conclude that there are significant differences between the outcomes of the approaches

under evaluation. Hence, we can falsify null hypothesis $\mathbf{H}_1$, and conclude that there are statistically significant differences between the baseline and METRA-based approaches.

Regarding the effect size statistics, table 4 shows the $\hat{A}_{12}$ values for precision and recall. The $\hat{A}_{12}$ values shown in the table reflect the scenario depicted by the results and box-plots of Fig. 8:

- Basic TLR is outperformed by the rest of the approaches in terms of both precision and recall. Remarkably, basic TLR attains near statistical equivalence with standalone LORE in terms of recall.

- Standalone LORE outperforms both METRA and the combination of METRA with LORE in terms of precision. In turn, standalone LORE is outperformed by both of the METRA-based approaches in terms of recall.

- Finally, METRA and the combination of METRA with LORE obtain values close to 0.5 in both precision and recall. Even though $\hat{A}_{12}$ results are slightly in favour of the combination of METRA with LORE, the obtained values reflect the near statistical equivalence of both that can be appreciated in the results table and figure.

Compared approaches	Precision	Recall
TLR vs. LORE	0.2765	0.4620
TLR vs. METRA	0.3145	0.1162
TLR vs. METRA+LORE	0.2901	0.0872
LORE vs. METRA	0.6761	0.1677
LORE vs. METRA+LORE	0.6395	0.1405
METRA vs. METRA+LORE	0.4427	0.4942

Table 4: Vargha and Delaney's $\hat{A}_{12}$ statistic

5.3. Summary of results

From the obtained results and the performed statistical analysis, it is possible to draw conclusions from our research, in the form of responses to the research questions posed in Section 4:

RQ$_1$ *Does METRA improve the results obtained by baseline techniques in a statistically significant manner?*

Yes. Null hypothesis $\mathbf{H}_1$ states that METRA does not improve the results of baseline techniques in a statistically significant manner. However, the results obtained by the METRA approach, along with the statistical tests carried out in Section **??**, falsify this null hypothesis. Hence, we can conclude that METRA improves the results obtained by baseline techniques in a statistically significant manner.

RQ$_2$ *Is it possible to combine METRA with other expansion-based TLR techniques to further enhance TLR results?*

Yes. Null hypothesis $\mathbf{H}_2$ states that it is not possible to combine METRA with other expansion-based TLR techniques to further enhance TLR results. However, as seen on Section 5, the combination of METRA with LORE yields better results than those obtained by the baselines and by standalone METRA. Moreover, the statistical tests carried out in Section **??** show that the obtained improvements are statistically significant. Hence, we can falsify this null hypothesis, and conclude that it is not only possible to combine METRA with other expansion-based techniques, but that the combination of METRA with other expansion-based techniques is able to further enhance the results of TLR over requirements and BPMN models.

Overall, the outcomes of our research attest that expanding the linguistic load and the amount of linguistic clues of the software artifacts in use has a beneficial impact on the results of TLR over requirements and BPMN models.

6. Discussion

By inspecting the results of the approaches, we have researched a series of facts that help explain why leveraging the linguistic clues in execution traces through METRA improves the results of the TLR process between requirements and BPMN models:

(1) Some of the elements in the fragments are relevant for the implementation of a particular requirement, but contain little to no text. Without the expansion of the fragments through execution traces, those particular elements cannot display their relevance. Hence, the fragments containing such elements are often disregarded by TLR and LORE, since LSI determines that query terms do not appear in the fragment. On the other hand, METRA adds the missing linguistic clues to the fragments containing those elements, making them more relevant for LSI.

(2) Some elements in the models are conformed by terms and words that are representative of their functions for human software engineers, but those terms and words are not necessarily aligned with all the underlying actions performed by the element or with the vocabulary in the requirements. The inclusion of execution traces with METRA leads to a reduction of this vocabulary mismatch, which in turn helps identify the true relevancy of the fragments containing those elements.

(3) In TLR and LORE, smaller fragments are often prioritized, since bigger fragments tend to present more elements with the problems depicted in points (1) and (2), which in turn impacts their relevancy score. With smaller solution fragments, precision is maximized, but recall values suffer because many correct elements are left out of the solution. Thanks to the inclusion of execution traces, problems (1) and (2) are mitigated, and through the addition of linguistic clues, METRA is capable of determining the relevancy of bigger fragments with more accuracy, taking them as solutions more often, which in turn leads to an increase in recall.

(4) The addition of this bigger model fragments impacts precision negative in METRA, since incorrect elements are also added to the solution. However, as more elements are present in a fragment, more execution traces are taken in account to expand the fragment. With more documents, Rocchio is capable of providing an improved list of relevant terms for the expansion. The improved list of terms and the diminishing of the vocabulary mismatch between fragments and requirements discussed in point (2) help mitigate the impact that taking bigger fragments has on precision.

(5) TLR and LORE suffer from greater standard deviations than the approaches where METRA is involved. Since both TLR and LORE depend only on the linguistic clues in requirements to obtain the similitude values, a poorly formulated query leads to poor results, whereas a good query leads to strongly improved results. In METRA, the execution traces act as an intermediate artifact that helps unify the language of queries and fragments, bridging the gap between the linguistic clues in both. Therefore, approaches based in METRA are less sensible to the quality of the software artifacts in use, improving the robustness of the results and reducing the difference between the best and worse results. However, this does not make approaches based in METRA immediately immune to problems in the quality of the software artifacts in use, including the quality of execution traces. Thus, studying the quality of the content of requirements and execution traces and how to ensure it remains as future work.

7. Threats to validity

In this section, we use the classification of threats to validity of [29] to acknowledge the limitations of our approach. The threats to the validity of our approach [29] are classified into four categories: conclusion validity, internal validity, construct validity, and external validity.

- **Conclusion validity:** This validity is concerned with the relationship between the treatment and the outcome. In other words, we must ensure that the research is not tailored for the data or vice-versa, also making sure that the relationship of the authors with the dataset is of a purely scientific nature.

  To minimize this threat, we have based our work on research questions and disproving null hypotheses through widespread statistical tests. In addition, the requirements and BPMN models used in our approach were taken from an industrial case study, and none of the authors of this work was involved in the generation of the data. Furthermore, we were not involved with the development of the execution traces nor with the

development of the code that generated those execution traces. As a matter of fact, the execution traces were provided by our industrial partner along with the rest of the software artifacts in use for this particular piece of research.

- **Internal Validity:** If a relationship is observed between the treatment and the outcome, we must make sure that it is a causal relationship, and that it is not a result of a factor of which we have no control or have not measured. In other words, that the treatment causes the outcome (the effect).

  To minimize this threat, we have utilized the same natural language processing techniques as preprocessing over the requirements and BPMN models. Moreover, we have followed the same evaluation process for all the evaluated approaches (the baselines and the approaches based on METRA). In addition, the available case study represents a wide scope of different scenarios in an accurate manner.

- **Construct validity:** This validity is concerned with the relation between theory and observation. If the relationship between cause and effect is causal, we must ensure two things: (1) that the treatment reflects the construct of the cause well, and (2) that the outcome reflects the construct of the effect well.

  To minimize this threat, our evaluation is performed around four widespread measurements: precision, recall, f-measure, and the MCC. These measurements are widely accepted and utilized in the state of the art literature. Moreover, we have used the same kinds of software artifacts for all the approaches (requirements, execution traces, and BPMN models), representing the same scenarios (processes in an industrial case study) so the results can be generalized among constructs.

- **External Validity:** The external validity is concerned with generalization. If there is a causal relationship between the construct of the cause, and the effect, can the result of the study be generalized outside the scope of our study? Is there a relation between the treatment and the outcome?

  All the artifacts in use (requirements, BPMN models, and traces) are frequently leveraged to specify and analyze all kinds of different processes. In addition, while it is true that the traces are generated through an in-house solution by our industrial partner, logging natural language traces is a common practice in all kinds of software development scenarios. The real-world industrial BPMN 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 those domains. In other words, we have not defined METRA based on the case study, but we have rather defined METRA and then applied the approach on the case study. Hence, our approach can potentially work in any scenarios where requirements, natural language traces, and BPMN models are available as inputs. Nevertheless, our results should be replicated with other case studies before ensuring their external generalization. Finally, in the railway domain it is common to need to certify both the software that runs the trains and the trains themselves, which implies providing evidence of requirements traceability. The experimental setup of the paper builds on those needs and serves as a means of providing traceability evidence in practice.

8. Related Work

Works related to this one can be found mainly within two different fields of research in the areas of Traceability Links Recovery, and the study of Software Linguistics and their application to Software Engineering tasks.

8.1. Traceability Links Recovery

The main works related to this one are our previous works on the topic, presented through [41]

and [42]. In [41], we perform TLR between requirements and BPMN models through generalist approaches. In [42], we improve TLR between requirements and models through an ontological expansion of the requirements (LORE). While [42] deals with improving the requirements used as input for TLR, the currently presented work focuses on improving the BPMN linguistic clues through the inclusion of execution traces into the process. The findings from this work and those presented in [42] are complementary, as proven by this paper.

Other works related to our research are mainly found within the knowledge area of Traceability Links Recovery. CERBERUS [43] provides a hybrid technique that combines information retrieval, execution tracing, and prune dependency analysis allowing to perform TLR between requirements and code. Eaddy et al. [44] present a systematic methodology for identifying which code is related to which requirement, and a suite of metrics for quantifying the amount of crosscutting code. Marcus and Maletic [45] use LSI for TLR between code and documentation (manuals, design documentation, requirement documents, test suites). Antoniol et al. [46] propose a method based on information retrieval for TLR between source code and free text documents, such as, requirement specifications, design documents, manual pages, system development journals, error logs, and related maintenance reports. Zisman et al. [47] automate TLR between requirements and object models using heuristic rules. These approaches perform TLR between different kinds of software artifacts, mainly code, but none of them perform TLR between requirements and BPMN models.

Regarding the extraction of BPMN model fragments from a base BPMN model, Piotr Wiśniewski [48] presents an approach towards the automatic decomposition of BPMN models according to a desired number of elements. The main goal of the approach is to retrieve a library of design patterns that can act as reusable components for building models and software systems. In our approach, we use an algorithm to extract a population of BPMN fragments that is not based on a particular desired number of elements, but rather, that is used to explore a large search space of which not all the details are available. In addition, our goal is not to generate reusable components, but to retrieve the closer model fragment to a particular requirement, which can then be used for a variety of software tasks besides reuse.

Other works target the application of LSI to TLR tasks. De Lucia et al. [49] present a TLR method and tool based on LSI in the context of an artifact management system, which includes models.The paper presented in [50] takes in consideration the possible configurations of LSI when using the technique for TLR between requirement artifacts, namely requirements and test cases. In their work, the authors state that the configurations of LSI depend on the used datasets, and they look forward to automatically determining an appropriate configuration for LSI for any given dataset. Through our work, we do not focus on the usage of LSI or its tuning, but rather aim to expand the information of BPMN models to improve TLR between requirements and BPMN models.

In a previous work [51], we explored TLR in MDD models, code generation models that must conform to the MOF standard of the OMG organization. While both this work and the work presented in [51] deal with TLR in models, they do so through different mechanisms and with very different research goals in mind. While the paper in [51] explores novel directions in Evolutionary Algorithms for TLR guided by an approach named Learning to Rank, the goal of this work is to build TLR techniques that cater to requirements and BPMN models. The work presented in [51] does not deal with the characteristics of BPMN models nor with its execution traces. On the other hand, our work abandons the more generalist approach of [51], which considers MDD models in general, in favor of exploring the peculiarities of BPMN models. In that sense, this work explores the unique aspects of BPMN semantics, and exploits the information available in BPMN execution traces with the main goal of mitigating the impact that the lack of inherent semantics in BPMN models has on the TLR process. The fundamental differences in the approaches and in the focus presented by both papers do not mean, however, that the two of them are completely independent and exclusive of each other, since there is a shared context in TLR in models. As a mat-

ter of fact, our research in TLR for BPMN, presented in this paper, can benefit from the novel SBSE TLR techniques introduced in [51], and the study of the linguistic particularities of models provided in this piece of work can be used to enhance and/or adapt the work in [51] for different kinds of models. The study of the potential collaborations between the two research branches and their authors remains as future work.

Finally, our ongoing research in the field has resulted in a novel work [52] that, as this paper does, aims to enhance the results of the TLR process between requirements and BPMN models. The novel research starts from the same premise of this paper, that is, the lack of inherent linguistic clues in BPMN models, and explores ways to mitigate such a lack of linguistics. Even if the goal of both papers is similar, there are also fundamental differences between the approaches of both papers. This research work leverages the linguistic clues available in execution traces, incorporating them to the models in order to enhance the BPMN artifacts themselves. On the other hand, the paper in [52] builds upon the identification of BPMN particularities, and creates a series of rules based on those particularities that are used to enhance the TLR process. Hence, the main fundamental difference between both papers is the focus of the approach: while our approach is focused on tackling the available information in the software artifacts in use, the approach in [52] tackles the TLR process, adapting it to the peculiarities of the artifacts. Once again, the differences in the approaches and in the focus of the research do not mean that the works are independent nor exclusive of each other. As a matter of fact, since one of the papers presents improvements to the process and the other presents improvements to the artifacts in use, we reckon that the combination of this work with the work presented in [52] can and should be explored, since both approaches seem compatible and complementary to each other. We reason that, if we were to combine both approaches, we would be enhancing all the different parts of our problem at once, and thus we might potentially find a more complete solution towards completing our research puzzle.

8.2. Software Linguistics

Other works focus on the impact and application of linguistics to Software Engineering tasks at several levels of abstraction. Works like [53] or [54] use linguistic approaches to tackle specific TLR problems and tasks. In [55], 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 [55] 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 [56] 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 propagate. Our work differs from [53] and [54], since our approach is not based on linguistic techniques as a means of TLR, but we rather expand the BPMN models to enhance TLR between requirements and BPMN models. Moreover, we do not study how linguistic techniques must be tweaked for specific problems as [55] does. In addition, differing from [56], we do not tackle changes in requirements nor TLR between requirements, but instead focus our work on TLR between requirements and BPMN models.

Another field of work puts the focus on aligning process models with textual descriptions, linking process models with textual descriptions, and discovering process models from natural language descriptions. In [57], the authors utilize a tailored linguistic analysis of each description to align the descriptions with the elements of the model, and present a technique that projects knowledge extracted from both process models and textual descriptions into a uniform representation that is amenable for comparison. In [58], Hugo López et. al. present a tool for building declarative processes from natural language texts and then trace changes in the process models back to the text. Other works by López et. al. [59, 60]

build on some of the ideas presented in [58] to propose methods for process discovery from natural language descriptions, and towards linking process models with their textual views. In our paper, we do not present a novel representation of the BPMN models that incorporates knowledge from the textual descriptions as [57] does. In addition, we do not put the focus on building process models from textual descriptions or discovering processes as the works presented in [58, 59, 60] do. Rather, we utilize execution traces to enrich the BPMN models, all with the aim of enhancing the TLR process between already existing software artifacts.

Remaining in the field of work of process and text alignment, Han van der Aa et. al. proposed an approach to identify inconsistencies between process models and their textual descriptions [61] and an approach that used behavioral spaces to capture all the possible interpretations of textual process descriptions in a systematic manner for compliance checking [62]. Our work does not deal with compliance checking and does not consider inconsistencies between the models and the descriptions. Rather, we aim to map textual requirements to the BPMN models.

Other works deal with the extraction of annotations from textual descriptions of process models, and their usage for several software tasks such as process mining. The annotations represent a middle-ground between the unstructured natural language of textual descriptions and the formal characteristics of modeling languages and styles. Quisphi et. al. focused on delivering an approach for extracting annotations from process descriptions [63], which they then improved by considering relationships between nearby sentences [64]. In a later work by Sánchez et. al. [65], the annotations are used as the central piece of a framework that enables process modeling on top of natural language descriptions of processes. This latter work also opens the possibility for new research lines in verification, simulation, and query answering in models on top of textual descriptions of processes. Our work does not deal with the alignment of processes and their textual descriptions, and does not use annotations extracted from the textual descriptions of the processes, but rather uses another software artifact (execution traces) to enhance the text of BPMN models towards improving traceability between the models and textual requirements. However, the annotations proposed in these prior works might be a valuable source of additional information that we might leverage in future works to further enhance the TLR process between requirements and BPMN models.

Finally, other works, derived from the the research of the authors of [66, 67, 68], delve in the area of process model matching, model to text matching, and the identification of language patterns with the aim of transforming BPMN models into natural language requirements, and natural requirements into BPMN models. However, to our knowledge, these papers and their authors have not researched the implications that these connections between natural language and BPMN models may have on Information Retrieval processes such as TLR, as our work does. In any case, in our paper, we do not claim to have identified the entirety of the particularities of BPMN models, nor that the identified particularities provide a complete coverage over the contents of requirements and/or BPMN models. The results of our paper are encouraging, so it is our belief that more work could be carried out in this particular line of research. In that sense, the papers presented in [57, 66, 67, 68] identify ways of aligning text and models that can be used in the future as a starting point to identify novel model particularities and language patterns, which may be used to further refine the TLR process.

9. Conclusions

Through this work, we propose a novel approach (METRA) that minimizes the impact that the lack of linguistic clues in BPMN models has on TLR between requirements and BPMN models. To that extent, the approach leverages the linguistic clues present in the BPMN models execution traces to expand the BPMN models, bridging the gap between the language in use in the requirements and the BPMN models. We evaluated our approach by carrying out METRA between the requirements and BPMN models that comprise a real-world industrial case study. Results show that

approaches based on METRA maintain the excellent precision results obtained by baseline approaches (78.8% on average), whilst also improving the recall results from the unacceptable values obtained by the baselines to good values (73.9% on average). Through the analysis of the obtained results, we discuss how leveraging the linguistic clues in execution traces improves the TLR process between requirements and BPMN models, so that further research can delve into this promising direction explored by our work.

Acknowledgements

This work was supported in part by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds through Project ALPS under Grant RTI2018-096411-B-I00, and in part by the Gobierno de Aragón (Spain) (Research Group S05_20D).

References

[1] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, 1st ed. Morgan & Claypool Publishers, 2012.

[2] L. A. Macaulay, *Requirements engineering*. Springer Science & Business Media, 2012.

[3] S. Winkler and J. Pilgrim, "A Survey of Traceability in Requirements Engineering and Model-Driven Development," *Software and Systems Modeling (SoSyM)*, vol. 9, no. 4, pp. 529–565, 2010.

[4] G. Loniewski, E. Insfran, and S. Abrahão, "A systematic review of the use of requirements engineering techniques in model-driven development," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2010, pp. 213–227.

[5] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "Feature Location in Models Through a Genetic Algorithm Driven by Information Retrieval Techniques," in *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '16. ACM, 2016.

[6] J. Martinez, T. Ziadi, T. F. Bissyande, J. Klein, and Y. Le Traon, "Automating the extraction of model-based software product lines from model variants (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 396–406.

[7] J. Martinez, T. Ziadi, M. Papadakis, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Feature location benchmark for extractive software product line adoption research using realistic and synthetic eclipse variants," *Information and Software Technology*, vol. 104, pp. 46–59, 2018.

[8] J. Krüger, M. Mukelabai, W. Gu, H. Shen, R. Hebig, and T. Berger, "Where is my feature and what is it about? a case study on recovering feature facets," *Journal of Systems and Software*, vol. 152, pp. 239–253, 2019.

[9] M. Chinosi and A. Trombetta, "BPMN: An Introduction to the Standard," *Computer Standards & Interfaces*, vol. 34, no. 1, pp. 124–134, 2012.

[10] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery," in *2010 IEEE 18th International Conference on Program Comprehension*. IEEE, 2010, pp. 68–71.

[11] R. Watkins and M. Neal, "Why and How of Requirements Tracing," *IEEE Software*, vol. 11, no. 4, pp. 104–106, 1994.

[12] A. Ghazarian, "A Research Agenda for Software Reliability," *IEEE Reliability Society 2009 Annual Technology Report*, 2010.

[13] P. Rempel and P. Mäder, "Preventing Defects: the Impact of Requirements Traceability Completeness on Software Quality," *IEEE Transactions on Software Engineering*, vol. 43, no. 8, pp. 777–797, 2017.

[14] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev, "Ontological Approach for the Semantic Recovery of Traceability Links Between Software Artefacts," *IET software*, vol. 2, no. 3, pp. 185–203, 2008.

[15] O. C. Gotel and C. Finkelstein, "An Analysis of the Requirements Traceability Problem," in *Proceedings of the First International Conference on Requirements Engineering*. IEEE, 1994, pp. 94–101.

[16] G. Spanoudakis and A. Zisman, "Software Traceability: a Roadmap," *Handbook of Software Engineering and Knowledge Engineering*, vol. 3, pp. 395–428, 2005.

[17] R. M. Parizi, S. P. Lee, and M. Dabbagh, "Achievements and Challenges in State-of-the-Art Software Traceability between Test and Code Artifacts," *IEEE Transactions on Reliability*, vol. 63, no. 4, pp. 913–926, 2014.

[18] J. Rubin and M. Chechik, "A Survey of Feature Location Techniques," in *Domain Engineering*. Springer, 2013, pp. 29–58.

[19] F. Pérez, J. Font, L. Arcega, and C. Cetina, "Collaborative Feature Location in Models through Automatic Query Expansion," *Automated Software Engineering*, vol. 26, no. 1, pp. 161–202, 2019.

[20] P. He, Z. Chen, S. He, and M. R. Lyu, "Characterizing the natural language descriptions in software logging statements," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 178–189.

[21] D. Yuan, S. Park, and Y. Zhou, "Characterizing logging practices in open-source software," in *34th International Conference on Software Engineering (ICSE)*. IEEE, 2012, pp. 102–112.

[22] T. K. Landauer, P. W. Foltz, and D. Laham, "An Introduction to Latent Semantic Analysis," *Discourse Processes*, vol. 25, no. 2-3, pp. 259–284, 1998.

[23] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating Natural Language Specifications from UML Class Diagrams," *Requirements Engineering*, vol. 13, no. 1, pp. 1–18, 2008.

[24] A. Hulth, "Improved Automatic Keyword Extraction Given more Linguistic Knowledge," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2003, pp. 216–223.

[25] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "On the Role of the Nouns in IR-Based Traceability Recovery," in *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE, 2009, pp. 148–157.

[26] G. Salton, *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall, Inc., 1971.

[27] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 309–318.

[28] C. Carpineto and G. Romano, "A Survey of Automatic Query Expansion in Information Retrieval," *ACM Comput. Surv.*, pp. 1:1–1:50, 2012.

[29] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer Science & Business Media, 2012.

[30] R. Lapeña, F. Pérez, C. Cetina, and Ó. Pastor, "Improving traceability links recovery in process models through an ontological expansion of requirements," in *International Conference on Advanced Information Systems Engineering*. Springer, 2019, pp. 261–275.

[31] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[32] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic, "An Information Retrieval Approach to Concept Location in Source Code," in *Proceedings of the 11th Working Conference on Reverse Engineering*, Nov 2004, pp. 214–223.

[33] T. Vale and E. S. de Almeida, "Experimenting with information retrieval methods in the recovery of feature-code SPL traces," *Empirical Software Engineering*, pp. 1–41, 2018.

[34] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19, 2006.

[35] Apache, "OpenNLP Toolkit for the Processing of Natural Language Text," https://opennlp.apache.org/, 2017, [Online; accessed 12-November-2017].

[36] P. Abeles, "Efficient Java Matrix Library," http://ejml.org/, 2017, [Online; accessed 9-November-2017].

[37] A. Arcuri and L. Briand, "A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering," *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.

[38] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced Nonparametric Tests for Multiple Comparisons in the Design of Experiments in Computational Intelligence and Data Mining: Experimental Analysis of Power," *Information Sciences*, vol. 180, no. 10, pp. 2044–2064, 2010.

[39] W. J. Conover, *Practical nonparametric statistics*. John Wiley & Sons, 1998, vol. 350.

[40] A. Vargha and H. D. Delaney, "A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong," *Journal of Educational and Behavioral Statistics*, 2000.

[41] R. Lapeña, J. Font, C. Cetina, and Ó. Pastor, "Exploring new directions in traceability link recovery in models: The process models case," in *International Conference on Advanced Information Systems Engineering*. Springer, 2018, pp. 359–373.

[42] R. Lapeña, F. Pérez, C. Cetina, and Ó. Pastor, "Improving Traceability Links Recovery in Process Models Through an Ontological Expansion of Requirements," in *International Conference on Advanced Information Systems Engineering*. Springer, 2019, pp. 261–275.

[43] M. Eaddy, A. V. Aho, G. Antoniol, and Y.-G. Guéhéneuc, "Cerberus: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis, and Program Analysis," in *ICPC 2008 conference*. IEEE, 2008, pp. 53–62.

[44] M. Eaddy, A. Aho, and G. C. Murphy, "Identifying, Assigning, and Quantifying Crosscutting Concerns," in *Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques*, 2007, p. 2.

[45] A. Marcus and J. I. Maletic, "Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing," in *Proceedings of the 25th International Conference on Software Engineering*. IEEE, 2003, pp. 125–135.

[46] G. Antoniol, G. Canfora, G. Casazza, A. de Lucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[47] A. Zisman, G. Spanoudakis, E. Pérez-Miñana, and P. Krause, "Tracing Software Requirements Artifacts," in *Software Engineering Research and Practice*, 2003, pp. 448–455.

[48] P. Wiśniewski, "Decomposition of business process models into reusable sub-diagrams," in *ITM Web of Conferences*, vol. 15. EDP Sciences, 2017, p. 01002.

[49] A. de Lucia *et al.*, "Enhancing an Artefact Management System with Traceability Recovery Features," in *Proceedings of the 20th IEEE International Conference on Software Maintenance*. IEEE, 2004, pp. 306–315.

[50] S. Eder, H. Femmer, B. Hauptmann, and M. Junker, "Configuring Latent Semantic Indexing for Requirements Tracing," in *Proceedings of the 2nd International Workshop on Requirements Engineering and Testing*, 2015.

[51] A. C. Marcén, R. Lapeña, Ó. Pastor, and C. Cetina, "Traceability link recovery between requirements and models using an evolutionary algorithm guided by a learning to rank algorithm: Train control and management case," *Journal of Systems and Software*, vol. 163, p. 110519, 2020.

[52] R. Lapeña, F. Pérez, C. Cetina, and O. Pastor, "Leveraging BPMN particularities to improve traceability links recovery among requirements and BPMN models," *Requirements Engineering*, pp. 1–26, 2021.

[53] H. Sultanov and J. H. Hayes, "Application of Swarm Techniques to Requirements Engineering: Requirements Tracing," in *18th IEEE International Requirements Engineering Conference*, 2010.

[54] C. Duan and J. Cleland-Huang, "Clustering Support for Automated Tracing," in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, 2007.

[55] D. Falessi, G. Cantone, and G. Canfora, "Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 18–44, 2013.

[56] C. Arora, M. Sabetzadeh, A. Goknil, L. C. Briand, and F. Zimmer, "Change Impact Analysis for Natural Language Requirements: An NLP Approach," in *IEEE 23rd International Requirements Engineering Conference*, 2015.

[57] J. Sànchez-Ferreres, H. van der Aa, J. Carmona, and L. Padró, "Aligning Textual and Model-Based Process Descriptions," *Data & Knowledge Engineering*, vol. 118, pp. 25–40, 2018.

[58] H. A. López, S. Debois, T. T. Hildebrandt, and M. Marquard, "The process highlighter: From texts to declarative processes and back." *BPM (Dissertation/Demos/Industry)*, vol. 2196, pp. 66–70, 2018.

[59] H. A. López, M. Marquard, L. Muttenthaler, and R. Strømsted, "Assisted declarative process creation from natural language descriptions," in *2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)*. IEEE, 2019, pp. 96–99.

[60] H. A. López, R. Strømsted, J.-M. Niyodusenga, and M. Marquard, "Declarative process discovery: Linking process and textual views," in *International Conference on Advanced Information Systems Engineering*. Springer, 2021, pp. 109–117.

[61] H. van der Aa, H. Leopold, and H. A. Reijers, "Comparing textual descriptions to process models–the automatic detection of inconsistencies," *Information Systems*, vol. 64, pp. 447–460, 2017.

[62] ——, "Checking process compliance against natural language specifications using behavioral spaces," *Information Systems*, vol. 78, pp. 83–95, 2018.

[63] L. Quishpi, J. Carmona, and L. Padró, "Extracting annotations from textual descriptions of processes," in *International Conference on Business Process Management*. Springer, 2020, pp. 184–201.

[64] ——, "Improving the extraction of process annotations from text with inter-sentence analysis," in *International Conference on Process Mining*. Springer, 2020, pp. 149–161.

[65] J. Sànchez-Ferreres, A. Burattin, J. Carmona, M. Montali, L. Padró, and L. Quishpi, "Unleashing textual descriptions of business processes," *Software and Systems Modeling*, pp. 1–23, 2021.

[66] J. Mendling, H. Leopold, L. H. Thom, and H. van der Aa, "Natural Language Processing with Process Models (NLP4RE Report Paper)," in *Requirements Engineering FSQ Workshops*, ser. CEUR Workshop Proceedings, vol. 2376. CEUR-WS.org, 2019.

[67] C. Klinkmüller, I. Weber, J. Mendling, H. Leopold, and A. Ludwig, "Increasing Recall of Process Model Matching by Improved Activity Label Matching," in *Business Process Management*. Springer, 2013, pp. 211–218.

[68] H. Leopold, J. Mendling, and A. Polyvyanyy, "Supporting Process Model Validation through Natural Language Generation," *IEEE Transactions on Software Engineering*, vol. 40, no. 8, pp. 818–840, 2014.