Noname manuscript No.
(will be inserted by the editor)

Leveraging BPMN Particularities to Improve Traceability Links Recovery among Requirements and BPMN Models

Raúl Lapeña · Francisca Pérez · Carlos Cetina · Óscar Pastor

Received: October 2, 2020 / Accepted: October 30, 2021

Abstract Traceability Links Recovery (TLR) has been a topic of interest for many years. However, TLR approaches are based on the latent semantics of the software artifacts, and are not equipped to deal with software artifacts that lack those inherent semantics, such as BPMN models. The aim of this work is to enhance TLR approaches in BPMN models by incorporating the linguistic particularities of BPMN models into the TLR process. Our approach runs through a threefold contribution: (i) we identify the particularities of BPMN models; (ii) we describe how to leverage the particularities; and (iii) we build three variants of the best exploratory TLR approach which specifically cater to BPMN models. The approach is evaluated through both an academic case study and a real-world industrial case study. Results show that incorporating the particularities of BPMN into the TLR process leads the specific approach to improve the traceability results obtained by generalist approaches, maintaining precision levels and improving recall. The novel findings of this paper suggest that there is a benefit in researching and taking in account the particularities of the different kinds of models in order to optimize the results of TLR between requirements and models, instead of relying on generalist approaches.

Keywords Model-Driven Software Engineering · Traceability Links Recovery · Requirements Engineering

Raúl Lapeña, Francisca Pérez, Carlos Cetina
SVIT Research Group
Universidad San Jorge
E-mail: {rlapena, mfperez, ccetina}@usj.es

Óscar Pastor
Centro de Investigación en Métodos de Producción de Software
Universitat Politècnica de València
E-mail: opastor@pros.upv.es

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 run-time. 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 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 then used 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, defined as the mapping of the traceability links between the software artifacts, or in other words, as the mapping of the dependencies and relationships that exist between the software artifacts, is a key to success in these industrial scenarios. In particular, traceability between requirements and the models that are derived from them is considered a good and necessary practice in industrial MDD contexts for many different factors. Apart from the usefulness of traceability for all kinds of software maintenance purposes, many kinds of engineering projects require the inclusion of traceability reports along with the finished products for certification purposes in major software standards such as CMMI or ISO 15504 [10]. In addition, affordable traceability can be critical to the success of a project [11], and leads to an increase in the maintainability and reliability of software systems by making it possible to verify and trace non-reliable parts [12]. Specifically, more complete traceability decreases the expected defect rate in developed software [13].

Even though all of these factors vouch for sound traceability, the latter is not always available, complete, or accurately updated when the need for its existence arises. Manually establishing and maintaining traceability links has proven to be a time consuming, error prone, and person-power intensive task [10, 14], and many companies simply cannot afford the workload derived from manual traceability efforts in the competitive market of software products. Motivated by the challenges posed by manual traceability in industrial MDD scenarios, and taking into account the preeminence of natural language requirements [15], the popularity of BPMN models [9], and the increase in the adoption of MDD practices in the industry [1], it becomes necessary to provide automated support to the engineers during the traceability process between requirements and BPMN models. Traceability Links Recovery (TLR) is defined as the software engineering task that deals with the automated identification and comprehension of traceability links [10], or

in other words, as the software engineering task that deals with the automated mapping of the interconnections that exist between software artifacts. TLR has been a subject of investigation for many years within both the software engineering community [16, 17] and the requirements engineering community [18, 19], and in recent years, it has been attracting more attention, becoming a subject of both fundamental and applied research [20].

However, most of the TLR approaches that stem from research efforts in both communities are based on Information Recovery approaches that rely on linguistic techniques. In that sense, TLR approaches often utilize the latent semantics of the software artifacts, understood as the textual cues and natural language that appear within them, in order to identify the dependencies and relationships, that is, to produce the traceability links between the software artifacts. As a result, TLR approaches obtain better results when used over artifacts that have abundant latent semantics, such as requirements, source code, or code generation models. These kinds of artifacts contain plenty of latent semantics in the form of natural language descriptions, developer comments, and designer notes. Therefore, most of the works in the TLR field focus on these kinds of software artifacts [21].

In contrast, BPMN models contain little to none latent semantics. So far, research about TLR techniques between requirements and BPMN models is practically nonexistent. Thus, several research challenges remain open in the field. Is it possible to apply commonplace TLR techniques to a research scenario that uses BPMN models as the main software artifacts, obtaining valuable traceability results in the process? How can we mitigate the impact that the lack of inherent latent semantics has over the TLR process in BPMN models? The aim of this paper is to fill this research gap by thoroughly studying the application of TLR techniques to an industrial MDD scenario where requirements and BPMN models are the protagonist software artifacts in use.

A first exploration of this particular research gap was carried out by Lapeña et al. [22, 23], where the authors put the focus on the first part of the research challenge by adapting existing approaches to work for BPMN models and by tackling the issue of tacit knowledge in requirements. We build on the ideas by Lapeña et al., shifting the focus towards the second part of the research challenge by researching how to circumvent or mitigate the issues that arise from the lack of latent semantics in BPMN models. We take advantage of works that study the linguistics of BPMN models, and leverage the linguistic particularities presented by BPMN models to build a novel approach, specific for TLR between requirements and BPMN models. Therefore, the main goal of this paper can be summarized as the proposal of a specific approach for TLR between requirements and BPMN models. The contribution of this paper is threefold: (i) we identify the particularities and traits of BPMN models on which we can capitalize to improve the TLR process between requirements and BPMN models (no-text elements and language patterns); (ii) we describe how to leverage the particularities to improve the TLR process; and (iii) we build three variants of the Mutation Search baseline that specifically cater to BPMN models by incorporating the BPMN models particularities in different manners to the TLR process.

The three variants are evaluated through two case studies, an academic case study and an industrial case study from one of our industrial partners. The results obtained by the three variants are compared against those obtained by baseline works that do not take in consideration any of the particularities of BPMN models.

The novel findings presented by this work highlight that the proposed BPMN-specific approaches improve the results of TLR between requirements and BPMN models, maintaining precision values and increasing recall.

The rest of the paper is structured as follows: Section 2 motivates the need for our work in TLR between requirements and BPMN models. Section 3 reviews the works related to this one. Section 4 provides the research framework for our work in the form of a description of previous approaches for TLR between requirements and BPMN models. Section 5 presents the BPMN model particularities that this work leverages to improve the TLR process. Section 6 describes both how to incorporate the particularities to the TLR process, and the approaches proposed by this work, depicting how to use them to carry out TLR between natural language requirements and BPMN models. Section 7 details the evaluation designed for the three variants. Section 8 presents and statistically analyzes the obtained results. Section 9 discusses the outcomes of the paper and highlights possibilities for future works. Section 10 presents the threats to the validity of our work. Finally, Section 11 concludes the paper by summarizing the main contributions and results.

2 Motivation

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. BPMN models are the models that implement the OMG's BPMN standard, which is the de-facto standard for graphically representing processes [9]. Lately, our industrial partner has been focusing some efforts on traceability between their natural language requirements and their BPMN models.

However, manual traceability is a time consuming, error prone, and person-power intensive task [10, 14]. Traceability between natural language requirements and BPMN models is no exception. Take for instance the image shown in Figure 1. The figure is an excerpt taken from the e-mail vote diagram, extracted from the e-mail voting system example found within the BPMN examples available on the BPMN standard official website[1]. From the excerpt, it is possible to manually extract the process for defining a list of issues on which votes must be taken. 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.

In industrial scenarios as the one from our industrial partner, the complexity of the BPMN models and the number of elements in place render manual traceability virtually impossible to attain. As an example, imagine that we try to manually trace the requirements to the model elements that comprise the data set provided by our industrial partner for this research. The data set comprises 5

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

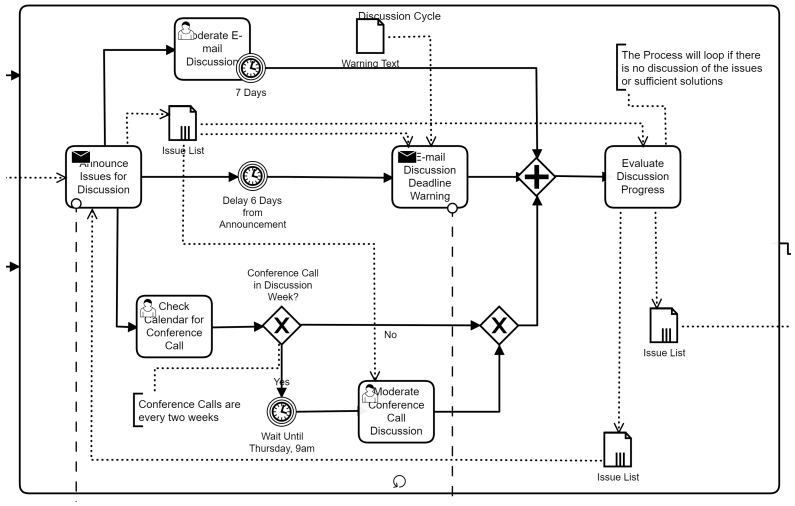

Fig. 1 Excerpt taken from the e-mail voting system diagram, an example found within the official BPMN standard website

trains, with around 100 requirements and one BPMN model per train, being the models composed, in turn, by an average of 850 model elements each. 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 [24], creating the fragment that retrieves the traceability to a requirement would take slightly more than one hour. Thus, tracing the total 100 requirements that implement a full train would take more than 100 hours, which translate into 13 full-time working days. Figure 2 shows a simplified example of a real-world industrial BPMN model from our industrial partner, along with a requirement and the traceability result in the form of a model fragment, composed by those model elements from the model that conform the traceability for the requirement.

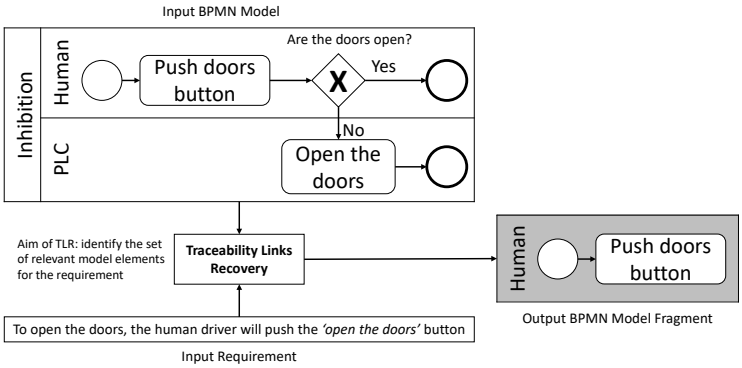

Fig. 2 Aim of our research and example inputs and output

Despite the complexity of the process, the application of automated TLR techniques to BPMN models is a largely unexplored research challenge, especially due to the fact that BPMN models have little to none inherent semantics, which TLR techniques base their results upon. Through our research, we aim to tackle this research challenge. The following section contains more information on the state of the art and the research gap that we aim to fulfill through our work.

3 Related Work

This section presents the works related to this one, grouped according to the community they belong to. In that sense, there are two main families of works that relate to this one: works devoted to Traceability Links Recovery, and works devoted to the study of Software Linguistics and their application to Software Engineering and Requirements Engineering tasks. This section also reflects on the development of research in the field up to this point, and about the existing research gap in TLR techniques among requirements and BPMN models that our work aims to cover.

3.1 Traceability Links Recovery

The main works related to this one are the works on the topic by Lapeña et al. [22, 23]. In [22], Lapeña et al. explore TLR between requirements and BPMN models through three different approaches of a generalist character: a baseline approach for models in general, and two techniques based on Latent Semantic Indexing, transported from code and adapted to work over models. The evaluation of the techniques over both an academic and an industrial set of BPMN models highlighted that the two techniques based on LSI performed better than the baseline technique in both case studies. The authors identified a series of particularities in the text of the requirements and the models that could potentially lead to improvement opportunities. Throughout the pages of this paper, we have incorporated BPMN particularities into the process, abandoning the generalist point of view of Lapeña et al. in order to tailor TLR to the unique characteristics of BPMN models. This allows us to mitigate the challenge that the lack of inherent semantics in BPMN poses to the TLR process between requirements and BPMN models. Through [23], Lapeña et al. leverage some of the particularities that they found in [22] to further research TLR between requirements and models. More precisely, Lapeña et al. put the focus on minimizing the tacit knowledge challenge in requirements by improving the requirements used as input for TLR, which they achieve through an ontological expansion of the requirements. In this study, the evaluation showed improvements over the previously discussed work. However, Lapeña et al. still focus on the generalist aspects of TLR, and do not cater to the particularities of BPMN models. Through the currently presented work, we are putting the focus on those particularities and the issues that they pose on TLR between requirements and BPMN models. In other words, our aim is to study in an isolated manner how those particularities affect the TLR process between requirements and BPMN models, and how to incorporate them into the process in order to improve the core of the TLR techniques for the specific challenge of TLR

between requirements and BPMN models. Nevertheless, we are building on the latest works by Lapeña et al., and the findings from this work are complementary to some of their research. For instance, it would be possible to aim for additional improvements on the whole process by simultaneously performing an ontological expansion of the requirements and incorporating BPMN particularities into the process as described in this paper.

Other works related to our research are mainly found within the knowledge area of Traceability Links Recovery. CERBERUS [25] 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. [26] 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 [27] use LSI for TLR between code and documentation (manuals, design documentation, requirement documents, test suites). Antoniol et al. [28] 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. [29] automate TLR between requirements and object models using heuristic rules. In more recent years, Schlutter and Vogelsang [18] proposed an approach based on semantic relation graphs to trace requirements to other requirements, and Madala et al. [19] performed an empirical study on automatic model elements identification for component state transition models from use case documents. These approaches perform TLR between different kinds of software artifacts, but none of them perform TLR between requirements and BPMN models.

Other authors target the application of LSI to TLR tasks. De Lucia et al. [30] present a TLR method and tool based on LSI in the context of an artifact management system, which includes models. The paper presented in [31] 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 present three variants of the Mutation Search approach with the aim of leveraging BPMN models particularities to improve TLR between requirements and BPMN models.

Finally, there is a recent paper [32] that researches TLR in MDD models, code generation models that must conform to the MOF standard of the OMG organization. The paper explores novel directions in Evolutionary Algorithms for TLR guided by an approach named Learning to Rank. Oppositely, the goal of this work is to transport and adapt TLR techniques to BPMN models, a particular type of non-MDD models, and more precisely to mitigate the impact that the lack of inherent semantics in BPMN models has on the TLR process. The work presented in [32] does not deal with BPMN models and does not question the quality or completeness of the semantics that compose the software artifacts in use. While both our work and the work presented in [32] deal with TLR in models, they do so through different mechanisms and with very different research goals in mind. This does not mean, however, that the two papers are completely independent and exclusive of each other, since there is a shared context in TLR in models. As

a matter of fact, our research in TLR for BPMN, presented in this paper, can benefit from the novel SBSE TLR techniques introduced in [32], 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 [32] for different kinds of MDD and non-MDD models. The study of the potential collaborations between the two research branches and their authors remains as future work.

3.2 Software Linguistics

Some other works focus on the impact and application of linguistics to Software Engineering tasks at several levels of abstraction. Works like [33] or [34] use linguistic approaches to tackle specific TLR problems and tasks. In [35], 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 [35] 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 [36] 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 [33] and [34], since our approach is not based on linguistic techniques as a means of TLR, but we rather present three variants of the Mutation Search approach to perform TLR between requirements and BPMN models. Moreover, we do not study how linguistic techniques must be tweaked for specific problems as [35] does. In addition, differing from [36], we do not tackle changes in requirements nor TLR between requirements, but instead focus our work on TLR between requirements and BPMN models.

In more recent years, there has been a research trend towards the automated generation of software artifacts through the use of linguistics. Pudlitz et al. [37] present a semi-automated approach based on a self-trained named-entity recognition model to extract system states from requirements specifications. Deshpande et al. [38] leverage the textual content of requirements to propose a requirements dependency extraction system based on active learning and an ontology-based information retrieval technique. Sequerloo et al. [39] generate test cases from requirement specifications through BPMN model transformations. Moitra et al. [40] developed a requirements capture and test case generation tool, called ASSERT, based on a formal requirements analysis engine. Finally, Reinhartz-Berger and Kemelman [41] designed an approach, named CoreReq, that generated core requirements for Software Product Lines through requirements clustering, NLP techniques, and an ontological variability framework. Qian et al. [42] formalized an approach to extract BPMN models from textual descriptions via neural networks trained on NLP techniques. Rebmann and van der Aa [43] mine BPMN event logs to extract information about the processes represented by the models through semantic role labeling. While all of these works put the focus on the automated generation of several different kinds of software artifacts, our work deals with TLR between already existing artifacts, namely, requirements and BPMN models.

Other works such as [44] are focused on aligning process models with textual descriptions. In the paper, 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 our paper, we do not present a novel representation of the BPMN models. Rather, we utilize BPMN linguistic particularities that can also be found in textual requirements to enhance the TLR process between both.

Finally, other works, derived from the the research of the authors of [45, 46, 47], 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 [44, 45, 46, 47] 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.

3.3 Analysis of the Research Gap

TLR has proven to be a major support activity for all kinds of SE and RE tasks regarding various kinds of software artifacts. Current TLR approaches often rely on the latent semantics of the software artifacts in use, and hence obtain better results over artifacts that contain an abundance of these latent semantics, such as requirements, source code, or MDD (code generation) models. As a result, most of the works in the field focus on researching TLR among these kinds of artifacts. However, there are other kinds of software artifacts that could benefit from TLR as well. Among those artifacts are BPMN models, which are also popular in the software development industry for a variety of tasks, mainly the specification and management of processes. These models have less text, and in consequence, less latent semantics that can be exploited by TLR techniques. Other works have studied the linguistics and latent semantics of the software artifacts in use, but their findings have not been applied to TLR. So far, the application of TLR approaches and the application of Software Linguistics to enhance the TLR process in scenarios where requirements and BPMN models are the main software artifacts in use has been a largely neglected and unexplored field of study.

Table 1 shows a relationship of the works that have put the focus on TLR and on Software Linguistics. The table highlights the particular community to which the works belong, the adopted approaches, and the artifacts in use for each of these state-of-the-art papers. From the table, it is possible to conclude that most works belong to either the TLR community, where the main goal is to obtain the traceability links in an automated manner, or to the Software Linguistics

community, where the focus shifts to the linguistics of the artifacts in use. Save this work, none of the other works presented in the table apply the available knowledge on BPMN linguistics to enhance the results of TLR approaches between requirements and BPMN models.

Work	Community	Approach	Artifacts
Eaddy et. al. [25]	TLR	IR+Tracing+PDA	Requirements and code
Eaddy et al. [26]	TLR	Methodology and metrics	Requirements and code
Marcus and Maletic [27]	TLR	LSI	Documentation and code
Antoniol et al. [28]	TLR	IR	Documentation and code
Zisman et al. [29]	TLR	Heuristic rules	Requirements and object models
Schlutter and Vogelsang [18]	TLR	Semantic Relation Graphs	Requirements
Madala et al. [19]	TLR	Neural Networks	Documentation and state models
De Lucia et al. [30]	TLR	LSI	Software artifacts
Eder et. al. [31]	TLR	Tuning of LSI	Requirements and test cases
Marcén et. al. [32]	TLR	SBSE approach	Requirements and MDD models
Lapeña et al. [22]	TLR	LSI-based approach	Requirements and BPMN models
Lapeña et al. [23]	TLR + SW Linguistics	LSI + Req. Linguistics	Requirements and BPMN models
This work	TLR + SW Linguistics	LSI + BPMN Linguistics	Requirements and BPMN models
Sultanov and Hayes [33]	SW Linguistics	Requirements Tracing	Requirements
Duan and Cleland-Huang [34]	SW Linguistics	Clustering for Traceability	Requirements
Falessi et. al. [35]	SW Linguistics	Requirements Equivalence	Requirements
Arora et. al. [36]	SW Linguistics	Requirements Evolution	Requirements
Pudlitz et. al. [37]	SW Linguistics	Named entity recognition	Requirements and system states
Deshpande et al. [38]	SW Linguistics	Active learning+Ontology	Requirements dependencies
Sequerloo et al. [39]	SW Linguistics	Model transformations	Requirements and test cases
Moitra et al. [40]	SW Linguistics	Requirements analysis	Requirements and test cases
Reinhartz-Berger and Kemelman [41]	SW Linguistics	Clustering+NLP+Ontology	Requirements
Qian et al. [42]	SW Linguistics	NLP Neural Networks	Textual descriptions
Rebmann and van der Aa [43]	SW Linguistics	Semantic role labeling	BPMN event logs
Sánchez-Ferreres et. al. [44]	SW Linguistics	Process Model Alignment	BPMN models and NL artifacts
Mendling et. al. [45]	SW Linguistics	Process Model Matching	BPMN models and NL artifacts
Klinkmuller et. al. [46]	SW Linguistics	Process Model Matching	BPMN models and NL artifacts
Leopold et. al. [47]	SW Linguistics	Process Model Matching	BPMN models and NL artifacts

Table 1 Analysis of the research gap

Thus, several research challenges remain open in the field. Is it possible to apply commonplace TLR techniques to a research scenario that uses BPMN models as the main software artifacts? How can we bridge the differences in the language in use by the different software artifacts? How can we mitigate the impact that the lack of inherent latent semantics has over the TLR process in BPMN models? Can we develop TLR techniques that are not affected by the lack of semantics? Can we apply Search-Based Software Engineering techniques guided by other factors other than linguistics to solve the problem?

The research gap can be filled by thoroughly studying the application of TLR approaches to a research scenario where requirements and BPMN models, a specific type of models that are used to specify processes and support process management, are the protagonist software artifacts in use. Nonetheless, it is not possible to solve all of these questions at once in a single piece of work, or even within a single research cycle. Instead, it is necessary to break the problem into smaller challenges. Through a first research iteration in the field, Lapeña et al. put the focus on the first question, and managed to transport TLR techniques to a research scenario where requirements and BPMN models are the protagonist software artifacts.

From there, it was possible to confirm several issues that were affecting the TLR process between requirements and BPMN models. For a start, Lapeña et al. corroborated that the texts of the requirements and the BPMN models were not aligned. To solve this issue, Lapeña et al. turned their eyes to the works in the Software Linguistics community, where they found an opportunity to mitigate the linguistic issues in the software artifacts by incorporating the knowledge of

the Software Linguistics community into TLR. Research works in the Software Linguistics community provided a framework for enhancing the linguistics of the software artifacts in use. In a second iteration of their work, Lapeña et al. expanded the input requirements through the usage of a domain ontology in an attempt to align the text of the requirements with the language in use in the BPMN models. The first application of linguistics guided their research to enhanced TLR results, improved from those obtained without the incorporation of linguistics. However, the lack of linguistics in the BPMN models still penalized the results.

Through this particular work, we have put the focus on covering the gap on this particular research challenge. In other words, our aim is to research how to circumvent or mitigate the issues that arise from the lack of latent semantics in BPMN models. To cover this gap, we take advantage of works that study the linguistics of BPMN models, and leverage the linguistic particularities presented by BPMN models, integrating these particularities into an approach that specifically performs TLR between requirements and BPMN models.

In order to solve the issues posed by the challenge of applying TLR techniques to BPMN models, which have little to none inherent semantics, we have joined the efforts from the two communities, enhancing TLR techniques with the power of linguistic techniques and approaches. Nevertheless, these are the first efforts of applying TLR and Software Linguistics to the challenge of performing TLR in BPMN models. There are plenty of challenges ahead, and a plethora of research questions that are yet to be posed and responded. In that sense, we firmly believe that we have begun exploring a very promising novel line of work.

4 Reseach Framework

Through the following paragraphs, we introduce the approaches for TLR between requirements and models considered as baseline in this work: the Linguistic Rule-Based approach, the Aggregation approach, and the Mutation Search approach.

4.1 Linguistic Rule-Based approach

Spanoudakis et al. [48] 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 [49] 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* (RTM) rules, specified by investigating grammatical patterns in requirements. These rules are specified as sequences of terms, and define relations between requirements and model elements. The rules are atomic: the matching succeeds if the model element contains the same words in the same pattern. We worked with a set of rules adapted to work over BPMN models.

4.2 Aggregation approach

The Aggregation approach receives a *query* requirement and a BPMN model as input, and generates a ranking of model elements through Latent Semantic Indexing (LSI). From the ranking, a model fragment is generated. To that extent, the BPMN 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 Figure 3 shows this process, having the example input BPMN model on the left, and the resulting model elements on the right. Afterwards, the text of the requirement and the model elements is treated through natural language processing techniques. To that extent, general phrase styling techniques (lowercasing and tokenizing), Parts-Of-Speech tagging [50], and lemmatizing [51] 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. LSI [52] is an automatic mathematical/statistical technique that analyzes relationships between *queries* and *documents* (bodies of text). LSI produces a *term-by-document co-ocurrence matrix*. The bottom left part of Figure 3 shows an example *term-by-document co-occurrence matrix*, with values associated to our running example. Each row in the matrix (*term*) stands for each of the words that appear in the processed text of the requirement and the model elements. In Figure 3, 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 BPMN model. In Figure 3, 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 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 Figure 3, 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) [52]. In Figure 3, 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 cosines between the *query* vector and the *documents* vectors are 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*). The model elements are ordered into a relevancy ranking according to the cosine measurement.

The relevancy ranking (which can be seen in Figure 3) 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-document* 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.

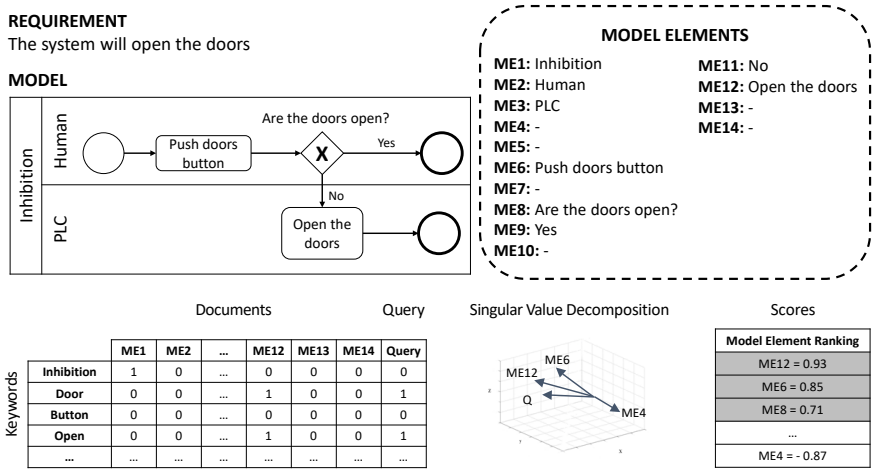

Fig. 3 Aggregation approach example

From the ranking, of all the model elements, those that have a similarity measure greater than x must be taken into account. We adopted the $x = 0.7$ heuristic, since it is used in other works [53, 54]. This value corresponds to a 45° angle between the corresponding vectors. However, there are other works that argue for the deprecation of this particular measurement [35], and for the usage of non-fixed thresholds. The study of the impact of the thresholds and the proper parametrization of our approaches remains as an open issue that must be tackled in the future.

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 Figure 3, '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 approach.

4.3 Mutation Search approach

The Mutation Search approach receives a *query* requirement and a BPMN 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 population of model fragments, algorithm 1 is followed. 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 model fragment is mutated and each new mutation is added to the population, avoiding the addition of repeated model fragments.

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
```

In the algorithm, a mutation in a model 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 model fragment), (2) removing a model element with only one connection (and the flow that causes said connection), or (3) adding or removing a lane from the model fragment. The performed mutation is chosen randomly on each iteration.

The top part of Figure 4 shows this process, having the example input BPMN 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 (lowercasing and tokenizing), Parts-Of-Speech tagging [50], and lemmatizing [51].

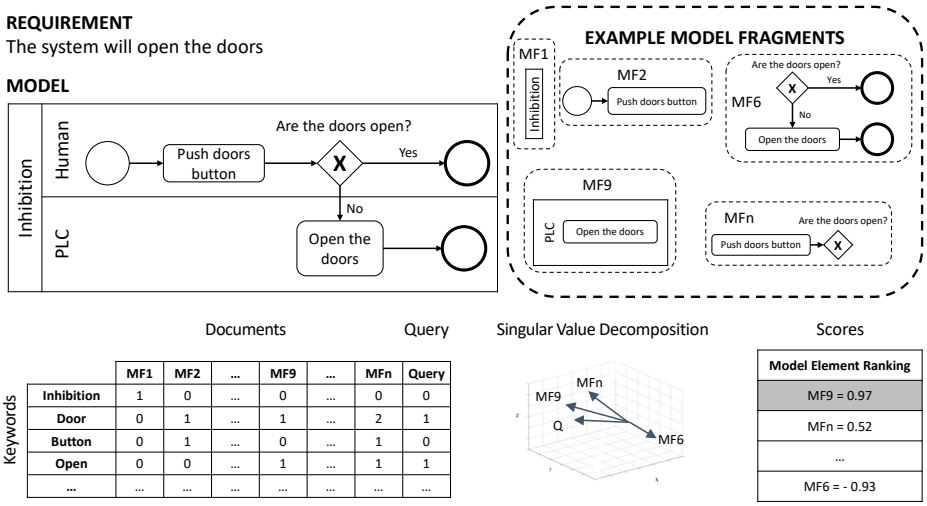

Fig. 4 Mutation Search approach example

Finally, the requirement and the model fragments are fed into LSI, which ranks the model fragments according to their similitude to the requirement. The technique works exactly as it does in the Aggregation approach, 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 model fragments are ordered according to the cosine measurement, producing the relevancy ranking shown on the bottom right part of Figure 4. In this example, LSI retrieves the 'MF9' model fragment in the first position of the relevancy ranking due to its *query-document* 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 approach.

5 BPMN Model Particularities

The results explored in the work by Lapeña et al. corroborated that the lack of inherent semantics in BPMN models negatively impacts the TLR process, and brought to light two potential kinds of particularities that can be leveraged in order to improve the TLR process between requirements and BPMN models. The study of these particularities, and how to incorporate them into the TLR process, conforms the core of this paper. In particular, we aim to incorporate the particularities into the most advanced of the baseline techniques, Mutation Search, since it is the technique that has obtained the best results so far.

5.1 No-text elements

Model elements with little or no text appear often in BPMN models, mainly in the form of flows (arcs that link elements) and sometimes in the form of events. Even though these elements have no text, they serve as important connections within the BPMN model. These elements can never be retrieved by the Linguistic approach: since there are no words, there is no pattern that can be matched. They are not retrieved by the Aggregation approach 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 approach, the algorithm does add these elements to the candidate fragments. Moreover, the addition of these elements does not penalize the results of the approach, since the *term* occurrences are not altered in any way by them. Therefore, the candidate fragments are more correct and complete in the Mutation Search approach, leading it to better results. Nonetheless, although these model elements can be added to the fragments constructed by Mutation Search, a detailed inspection of the population of model fragments revealed that many essential events and flows are still left out of the generated fragments by the approach.

5.2 BPMN models language patterns

As discussed in Section 3.3, our aim in this work is incorporate BPMN-specific linguistics into the TLR process in order to tackle the challenge of mitigating the lack of inherent semantics in BPMN models. To that extent, we identify and take into consideration five different kinds of language patterns, specific to the language in use in the artifacts associated to the development of BPMN models: (1) the usage of the term 'if' in a requirement almost always indicates the presence of an associated gateway in the BPMN model, (2) the terms 'start' or 'end' are usually implemented as events of the same type in the BPMN model, (3) questions are often related with gateways in the BPMN model, (4) verbs appear mostly as tasks in the BPMN model, and (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 BPMN model as a lane. By studying these patterns of the BPMN models language, it is possible to take in account these particularities in the TLR approaches, leading them to enhanced traceability results.

6 Approach

Through the pages of this section, we firstly provide an overview of our approach, explain how to leverage the particularities of BPMN to improve the TLR process between requirements and BPMN models afterwards, and finally, we build three variants of the Mutation Search approach that incorporate the BPMN models particularities.

6.1 Incorporating BPMN models particularities into Mutation Search

Figure 5 presents an overview of our approach. Square boxes in the figure represent the inputs and outputs of the approach and each of its steps. The inputs for the approach are a BPMN model, and a set of requirements that must be traced. Rounded boxes represent the steps and algorithms in use in the approach.

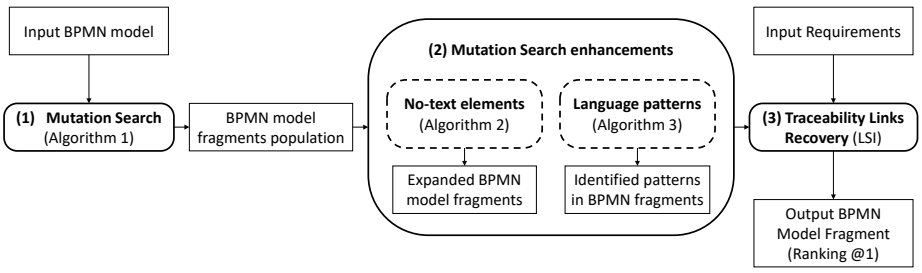

Fig. 5 Approach Overview

The approach runs in three steps:

(1) Firstly, we use the Mutation Search approach, implemented through Algorithm 4.3 and introduced in Section 4, to generate a population of BPMN model fragments from the input model.
(2) The second step oversees the enhancements to the Mutation Search approach through the incorporation of BPMN particularities. We identified and incorporated two types of particularities, described in Section 5: no-text elements (see Algorithm 6.1.1), and language patterns (see Algorithm 6.1.2). The incorporation of the particularities is the focus of the rest of this subsection.
(3) The third and final step of the approach is to perform the TLR process itself through Latent Semantic Indexing (LSI), as described in Section 4.

The following paragraphs describe how to incorporate each of the identified particularities into the Mutation Search approach.

6.1.1 No-text elements

The trace of a particular requirement is corresponded with a set of elements from a BPMN model fragment. One or several of the elements that belong to the trace may have no associated text. The Mutation Search approach can leave these elements with no text outside of the BPMN model fragments in the generated population. We incorporate no-text model elements into Mutation Search through algorithm 2.

Algorithm 2 Incorporating no-text elements into the BPMN fragments

```
1: P ← P                                          ▷ Get the population of model fragments
2: for each fragment F in P do
3:     F ← addMissingEvents(F)                    ▷ Add the missing events
4:     F ← addMissingFlows(F)                     ▷ Add the missing flows
5: end for
6: return P                                       ▷ Return the modified population
```

The algorithm iterates over the model fragments in the population, searching for missing no-text model elements and adding them into the model fragments:

Missing events: A model element that appears in a particular model fragment can be connected to a no-text event in the input BPMN model, with the latter being absent from the model fragment. When such a case is due, the no-text event is added to the model fragment.

Missing flows: Two model elements that share a flow connection within the input BPMN model may appear unconnected in a particular model fragment. If such a missing link is found within a model fragment, the flow is added to the model fragment. This also takes in account the flows that connect the events added as per the ruling in 'missing events'.

The addition of no-text elements to model fragments is portrayed in Figure 6. The left part of the figure depicts a model fragment, candidate solution for the requirement, obtained through the Mutation Search approach. In the input BPMN model, the task 'push doors button' is connected to a no-text event. According to

the ruling regarding 'missing events', the event is added to the model fragment. Then, due to the ruling regarding 'missing flows', the no-text flow that connects both the task and the event is added to the fragment as well. The right part of the figure depicts the resulting fragment, with the red dashed line and square representing the addition process and showing the added model elements, respectively.

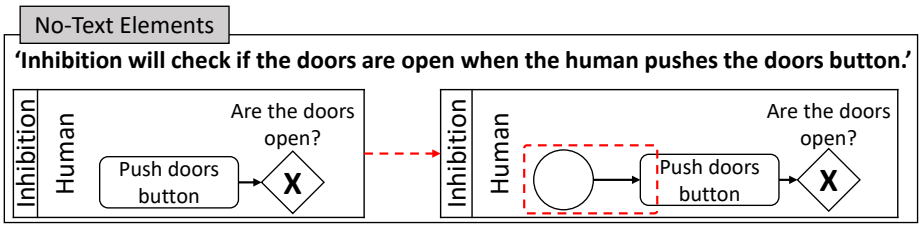

Fig. 6 Incorporating no-text elements into the Mutation Search approach

6.1.2 BPMN models language patterns

The second group of particularities comprehends the language patterns used for the development of BPMN models. We have identified five different patterns, described in Section 5.2, and incorporated into Mutation Search through algorithm 3.

In order to incorporate these language patterns into Mutation Search, algorithm 3 iterates over the population of model fragments, searching for model elements that can be mapped to the specific language patterns. The found model elements are included into LSI as *terms*, that is, rows in the matrix (avoiding duplicates). Then, the values of the cells of the matrix are calculated taking in account the newly added *terms*.

Algorithm 3 Incorporating language patterns into Mutation Search

```
1: P ← P                                          ▷ Get the population of model fragments
2: M ← buildMatrix(P)                             ▷ Build the LSI matrix
3: for each fragment F in P do
4:     C ← getTermElements(F)                     ▷ Retrieve the particular elements
5:     M ← incorporateElements(M, C)              ▷ Add elements to LSI matrix
6: end for
7: M ← calculateValues(M)                         ▷ Calculate the values of the matrix
8: Q ← calculateQueryColumn()                     ▷ Calculate the values of the query column
9: S ← performLSI()                               ▷ Retrieve the solution fragment
10: return S                                      ▷ Return the solution fragment
```

An example of this process can be found within Figure 7, where an example requirement is shown along with a model fragment (MF20), candidate solution for the requirement, generated through the Mutation Search approach. In the fragment, certain elements are found that can be matched to the language patterns: (1) the gateway ('are the doors open?'), (2) the task ('open the doors'), (3) the lane ('PLC'), and (5) the end event (without text). The model elements are included in the LSI matrix as *terms* (rows in the matrix), and then the matrix values are

calculated taking in account the newly added model elements. In the figure, for the *document* (column in the matrix) corresponding to the model fragment MF20, the values in the cells associated to the newly added *terms* are set to '1', since the model elements in the rows appear once in the model fragment.

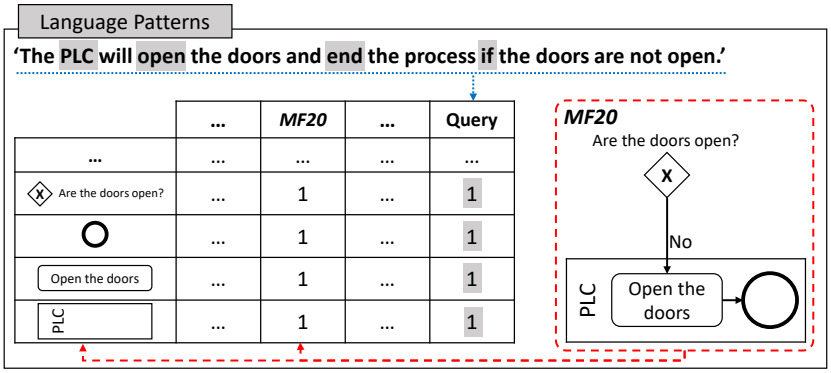

Fig. 7 Incorporating BPMN language patterns into the Mutation Search approach

Afterwards, the *query* column is built. To that extent, the requirement is analyzed in search for language patterns ('if' clauses, event keywords 'start' and 'end', question marks, verbs, and nouns at the beginning of the requirement).

The values of the cells of the *query* column are calculated taking in account the matches between the language patterns and the newly included *terms*:

'If' clauses 'If' clauses can be associated to gateways, so the values of the cells of the *query* column associated to the newly added gateway *terms* are increased by one per each 'if' in the requirement.

Keywords 'start' and 'end' The 'start' and 'end' keywords can be associated to events of their type. Therefore, the values of the cells of the *query* column associated to start and end event *terms* are increased by one per each 'start' and 'end' in the requirement, respectively.

Question marks Question marks can be associated to gateways, so the values of the cells of the *query* column associated to gateway *terms* are increased by one per each question mark in the requirement.

Verbs Verb clauses can be associated to tasks, so the values of the cells of the *query* column associated to task *terms* are increased by one per each verb in the requirement, provided the task contains the verb in its text.

Nouns Nouns at the start of a requirement can be associated to lanes, so the values of the cells of the *query* column associated to lane *terms* are increased by one per each noun at the start of the requirement, provided the lane contains the noun in its text.

In that way, the model fragments that contain model elements which can be mapped in any way to the requirement are weighed in a positive manner. As an example, in Figure 7, the aforementioned language patterns can be identified in the requirement ('if' clause, the 'end' keyword, the verb 'open', and the noun 'PLC' at the start of the requirement). The language patterns match the newly

added *terms*, that is, the added model elements. Hence, values are set accordingly in the matrix. In the figure, the language patterns and their incorporation to the matrix can be seen highlighted in light gray. Since MF20 and the *query* requirement now share the matches between the model elements and the language of the requirement, identified through the defined language patterns, the model fragment will be weighed in a positive manner and will appear in a higher position of the ranking generated through LSI, thus augmenting its probabilities for becoming the candidate model fragment solution for the requirement.

6.2 Mutation Search Variants

We have designed three possible variants of Mutation Search, in accordance to the two groups of particularities described above:

Variant 1 Mutation Search + no-text elements: this variant incorporates the no-text elements into the Mutation Search fragments as per the algorithm described above, and then performs the TLR process through LSI.

Variant 2 Mutation Search + language patterns: this variant incorporates the identified language patterns into Mutation Search as per the algorithm described above, performing the TLR process through LSI after the modification of the matrix.

Variant 3 Mutation Search + no-text elements + language patterns: this variant firstly incorporates the no-text elements into the Mutation Search fragments, and then incorporates the identified language patterns into the matrix, prior to performing LSI.

In order to test the impact of the aforementioned variants over the TLR process, we applied them to two case studies, comparing their results against those obtained by the baseline techniques.

7 Evaluation

Through the following paragraphs, we introduce the research questions, experimental setup, and case studies used to evaluate our work. We also present the oracles used to evaluate our work, and detail the design and implementation of the evaluation steps.

7.1 Research Questions

We have conducted our evaluation following the principles by Wohlin et. al. [55]. According to the guidelines, the evaluation process must be conducted through the study of research questions, which must be specific to the research problem under study and narrow enough to adequately pinpoint the research.

Our evaluation has been designed with the aim of responding the following research questions:

RQ_1 *Do BPMN-specific variants improve the results of the generalist baselines when performing TLR between requirements and BPMN models?*
RQ_2 *Do the baseline and variants vary on accuracy?*

According to the principles presented in Section 8.2 of [55], in order to prove that a particular piece of research satisfies its goals, it is necessary to disprove 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 this opposite hypothesis. The opposite hypothesis is known as null hypothesis. According to [55], 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 *BPMN-specific variants do not improve the results of the generalist baselines when performing TLR between requirements and BPMN models.*
H_2 *The accuracy does not vary between the baseline and variants.*

Through our work, we aim to disprove the null hypotheses, in order to prove that BPMN-specific variants improve the results of the generalist baselines, and that the accuracy does vary between the baseline and variants. The rest of this section is devoted to defining the evaluation process followed to respond these questions.

7.2 Experimental Setup

Figure 8 shows an overview of the evaluation process. The top part shows the inputs, which are extracted from the documentation provided in the case studies: requirements, BPMN models, and the approved traceability between requirements and BPMN models. Each case study comprises a set of requirements, a BPMN model, and an approved traceability document, which maps each requirement to a BPMN model fragment. The approved traceability document conforms the oracle of our evaluation.

For each case study, all the approaches generate a model fragment, containing the elements of the model that are related to the requirement according to each approach. The model fragments generated in this way are compared against their respective oracles, which are considered to be the ground truth. Once the comparisons are performed, a confusion matrix is calculated for each approach. A confusion matrix is a table that is often used to describe the performance of a classification model (in this case, each of the approaches) on a set of test data (the solutions) for which the true values are known (from the oracle). In our case, each outputted solution is a model fragment, subset of the model elements that are part of the BPMN model. Since the granularity is at the level of model elements, the presence or absence of each model element is considered as a classification.

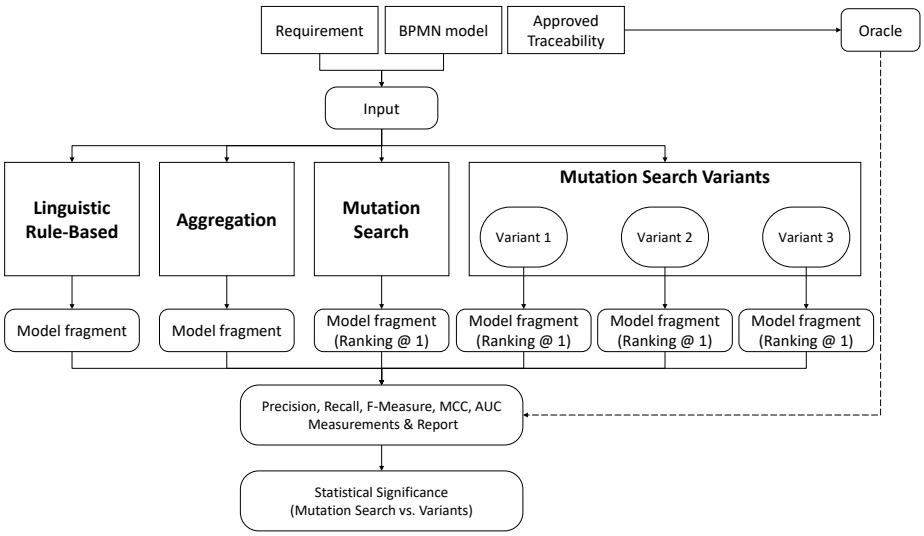

Fig. 8 Experimental setup

The confusion matrix arranges the results of the comparison between the model fragment from the oracle and the model fragment from the approach into four categories of values: True Positive, False Positive, True Negative, and False Negative values. Although the confusion matrix holds the results of the comparison between the results of the approach and the results of the oracle, it is necessary to extract some measurements from the confusion matrix in order to evaluate the performance of each approach. To that extent, some performance measurements are derived from the values in the confusion matrix. In particular, we report five performance measurements for all the approaches and both case studies: recall, precision, F-measure, MCC (Matthews Correlation Coefficient) and AUC (Area Under the Curve) [56, 53, 35].

7.3 Case Study

In order to perform the evaluation of the three variants, we rely on two different case studies : (1) an academic case study, and (2) a set of BPMN models provided by one of our industrial partners:

Academic case study: The academic case study consists of four BPMN modeling exercises. Each exercise contains an associated textual description and the solution BPMN model for the provided description. In order to apply the TLR approaches to the academic 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 BPMN models in the case study contain an approximate average of 25 elements per model.

Industrial case study: For our evaluation, one of our industrial partners provided us with the natural language requirements and the BPMN models of five railway solutions. The functionality is specified through about 100 natural language requirements per each of the trains, for an approximate total of 500 requirements, with an approximate average of 50 words per requirement. The number of identified language patterns differs greatly between requirements, but on average, 3 language patterns are identified per requirement. With regards to the BPMN models, the functionality is specified through one BPMN model per train, with an average of 850 total model elements per BPMN model.

While our industrial case study is larger and more realistic than the academic case study, the latter comprises an interesting set of scenarios that we would not be able to model or replicate otherwise. Without the academic case study, we might concur in the risk of tailoring our approaches to the problem domain, and hence we might lose the perspective and purpose of generalizable research. The academic case study also contributes to prove that this work is independent from the domain, and that its functioning does not rely on the peculiarities of the industrial case study.

7.4 Oracle

In order to obtain the performance results of the approaches under study, the produced outcomes must be compared against the correct solutions of the two case studies:

Academic case study: In the case of the academic case study, each exercise has an associated solution BPMN model. 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. Model fragments in the academic case study oracle range from 5 to 10 model elements. The same engineer also mapped each of the generated fragments to each of the derived requirements. Thus, we were provided with a set of requirements, the model fragments that implement them, and the TLR mapping between each requirement and the model fragment that implements the requirement.

Industrial case study: Our industrial partner provided us with their existing documentation on the traceability between the provided requirements and the provided BPMN models. In the documentation, each requirement from a particular train is mapped to a single model fragment from the full BPMN model of the same train. The oracle model fragments vary greatly in size and complexity, but on average, the oracle fragments are composed by 35 elements.

In both cases, we consider the existing documentation on traceability as the ground truth (oracle) of the case studies, which serves us as a means of evaluating the outcomes of the approaches by comparing them against the oracle.

7.5 Implementation details

A prototype of our research can be found online [omitted for blind review purposes]. We have used three libraries to implement the different approaches taken

in account through this work: (1) the Camunda BPMN API[2], (2) the OpenNLP[3] toolkit, and (3) the Efficient Java Matrix Library[4] (EJML). For the evaluation, we used a Lenovo E330 laptop, with a processor Intel(R) Core(TM) i5-3210M@2.5GHz with 16GB RAM and Windows 10 64-bit.

8 Results and Statistical Analysis

This section presents the outcomes of the variants, the statistical analysis of the results, the responses to the research questions posed in Section 7, and the reception of the approach by the end users.

8.1 Results

Tables 2 and 3 outline the results of the baselines and variants for both case studies. In the tables, each row shows the precision, recall, F-measure, MCC, and AUC values obtained through each approach for each case study. The novel findings presented by this work highlight that the proposed BPMN-specific approaches improve the results of TLR between requirements and BPMN models, maintaining precision values and increasing recall. The following paragraphs detail the particular results obtained in each case study.

8.1.1 Academic case study

Results in the academic case study show that Variant 3 is the one that achieves the best results for all of the measured performance indicators except for precision, in which Variant 3 obtains similar results than those obtained by the baseline and the other two variants. Variant 3 provides a mean recall value of 85%±16%, a combined F-measure of 71%±12%, an MCC value of 0.65±0.14, and an AUC value of 0.786. In contrast, the first and second variants and the baseline present worse results than Variant 3 in these same measurements in the same case study. In the case of recall, Variant 1 and Variant 2 obtain worse results than Variant 3, but better results than those obtained by the baseline. In the case of precision, results are very similar, with the deviation of the values being the only differentiating factor between them.

Academic case study					
Approach	**Precision**	**Recall**	**F-Measure**	**MCC**	**AUC**
Linguistic Rule-Based	40%±25%	35%±22%	33%±13%	0.25±0.19	0.372
Aggregation	56%±18%	72%±22%	61%±17%	0.52±0.24	0.616
Mutation Search	63%±21%	77%±22%	68%±19%	0.60±0.24	0.691
MS Variant 1	63%±17%	80%±18%	68%±14%	0.62±0.15	0.713
MS Variant 2	62%±15%	83%±18%	70%±15%	0.63±0.17	0.718
MS Variant 3	63%±13%	85%±16%	71%±12%	0.65±0.14	0.786

Table 2 Precision, recall, F-measure, MCC, and AUC in the academic case study

[2] https://github.com/camunda/camunda-bpmn-model
[3] https://opennlp.apache.org/
[4] http://ejml.org/

Figure 9 shows the boxplots for the results of precision, recall, F-measure and the MCC in the academic case study. The figure also shows the ROC curves associated to the obtained AUC values for the baselines and variants in the academic case study. In the boxplots, it is possible to appreciate the distributions for the results, and to visualize the equivalence of the precision measurement and the improvements in recall obtained by the variants.

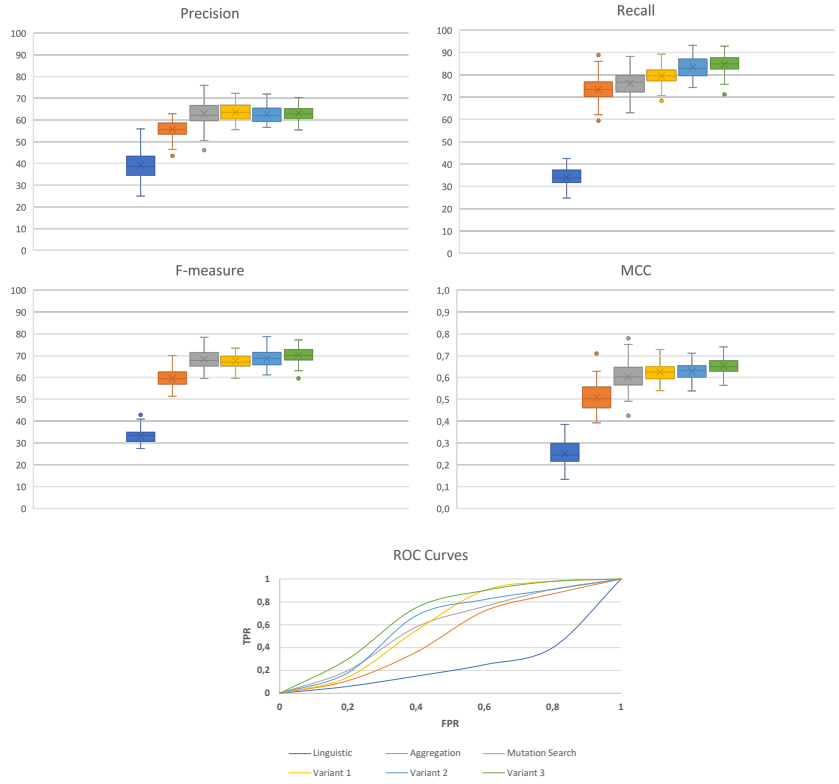

Fig. 9 Box plots for Precision, Recall, F-measure, and MCC in the academic case study

8.1.2 Industrial case study

In the industrial case study, Variant 3 achieves the best results all the measurements except precision. Variant 3 achieves a mean recall value of 93%±14%, a combined F-measure of 80%±7%, an MCC value of 0.76±0.09, and an AUC of 0.791. In this case study, Variant 3 does not improve the precision values of other variants or the baseline. As a matter of fact, it is the baseline, plain Mutation Search, the approach that attains the best precision value, with a mean value of 79%±19%. In this case study, the differences in recall are what guide Variant 3 to the enhanced F-measure, MCC, and AUC values.

Industrial case study					
Approach	**Precision**	**Recall**	**F-Measure**	**MCC**	**AUC**
Linguistic Rule-Based	35%±28%	35%±10%	30%±7%	0.18±0.13	0.287
Aggregation	69%±29%	66%±17%	64%±17%	0.58±0.21	0.599
Mutation Search	79%±19%	72%±19%	74%±16%	0.69±0.20	0.672
MS Variant 1	75%±11%	86%±16%	79%±6%	0.74±0.09	0.714
MS Variant 2	68%±15%	86%±20%	76%±16%	0.71±0.19	0.690
MS Variant 3	71%±4%	93%±14%	80%±7%	0.76±0.09	0.791

Table 3 Precision, recall, F-measure, MCC, and AUC in the industrial case study

Figure 10 shows the boxplots for the results of precision, recall, F-measure and MCC in the industrial case study. The figure also shows the distribution of the ROC curves associated to the obtained AUC values for the baselines and the variants in the industrial case study.

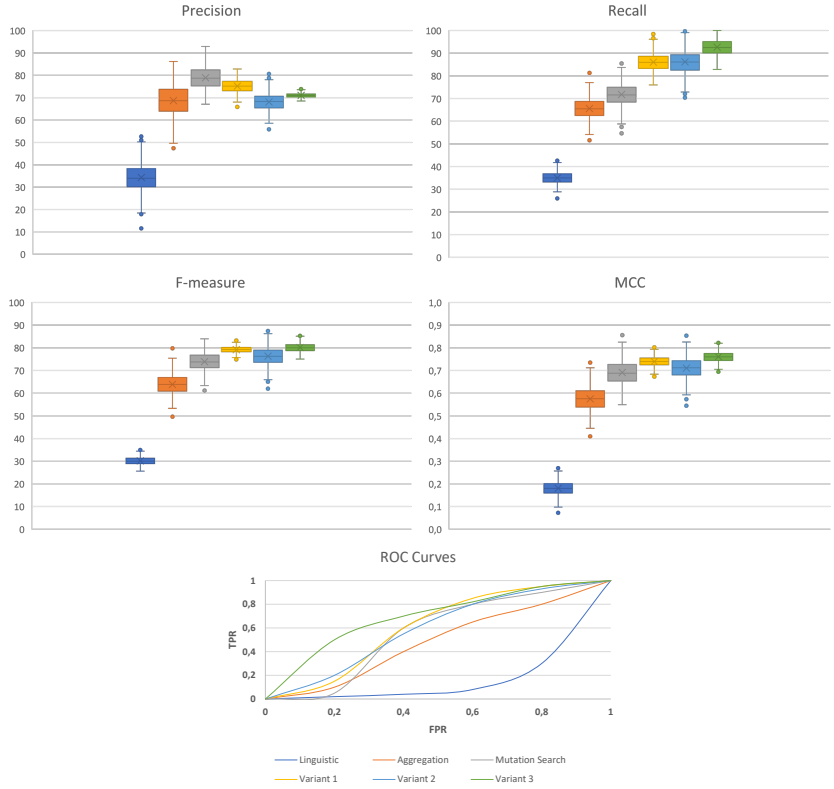

Fig. 10 Box plots for Precision, Recall, and F-measure in the industrial case study

In the boxplots, it is possible to appreciate the distributions for the results, and to visualize the improvements in recall obtained by the variants, especially by Variant 3. With regards to precision, in the boxplots it is possible to visually perceive that Variant 1 also outperforms Variant 3.

For Variant 2, this is uncertain: the average values are close, but the deviation of the values does not allow for a straightforwards comparison, since a few very low values would suffice to greatly alter the average without affecting the uppermost results. This comparison is clarified through the statistical tests presented in the following subsection.

8.2 Statistical Analysis

The obtained results look promising and indicate that the variants improve the results of the baselines. However, it is necessary to address whether the improvements are statistically significant. To that extent, we compare the results of the variants against those obtained by Mutation Search. Mutation Search was chosen as the baseline over which the three variants are built upon for being the baseline that obtained the best results in our previous research works in the field. For this very same reason, a significant improvement against the results of Mutation Search would also implicitly imply a significant improvement against the results obtained by the other two baselines.

In order to assess whether there are significant differences in performance between the baseline and the variants, their results must be properly compared through statistical methods, following the guidelines presented in [57]. The goals of the statistical analysis are twofold: (1) provide formal evidence that the variants 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. The null hypothesis H_2, defined in Section 7 along with the research questions, states that the differences in the results of the baseline and the variants are not significant. The statistical tests aim to disprove H_2, and verify that it can be rejected. 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 H_2 can be rejected. It is accepted by the research community that a *p-value* under 0.05 is statistically significant [57] towards disproving the null hypothesis.

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. There are several tests for analyzing this kind of data. However, the Quade test is more powerful than other tests when working with real data [58], and according to [59], has shown better results than other tests when the number of algorithms is low (no more than 4 or 5 algorithms). The Quade test returns the following *p-values*: (1) 0.04 for precision and $2x10^{-16}$ for recall in the academic case study, and (2) $3.2x10^{-5}$ for precision, $2x10^{-16}$ for recall in the industrial case study. 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 baseline and the variants.

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 $\hat{A}_{12}$ [60]. $\hat{A}_{12}$ 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.

The following paragraphs describe the values of the effect size statistics for precision and recall when comparing the Mutation Search baseline and the variants in both case studies.

8.2.1 Academic case study

Table 4 shows the $\hat{A}_{12}$ values for precision and recall in the academic case study.

Academic case study		
Compared approaches	**Precision**	**Recall**
Baseline vs Variant 1	0.4763	0.4600
Baseline vs Variant 2	0.4985	0.4157
Baseline vs Variant 3	0.4896	0.4001
Variant 1 vs Variant 2	0.5362	0.4371
Variant 1 vs Variant 3	0.5259	0.4216
Variant 2 vs Variant 3	0.4919	0.4837

Table 4 $\hat{A}_{12}$ statistic for Mutation Search vs. its variants in the academic case study

Regarding **precision**, the $\hat{A}_{12}$ values range closely to 0.5, reflecting the near equivalence scenario depicted by the results and boxplot of Figure 9:

- The three variants obtain better precision results than the baseline: Variant 1 does so in 52.37% of the runs, Variant 2 does so in 51.15% of the runs, and Variant 3 does so in 51.04% of the runs.
- Variant 1 outperforms the other two variants: Variant 2 in 53.62% of the runs, and Variant 3 in 52.59% of the runs.
- Finally, Variant 3 outperforms Variant 2 in 50.81% of the runs.

Regarding **recall**, the $\hat{A}_{12}$ values also confirm the results scenario shown in the boxplot of Figure 9, where Variant 3 ranks as the variant that obtains the best results:

- The three variants obtain better recall results than the baseline: Variant 1 does so in 54.00% of the runs, Variant 2 does so in 58.43% of the runs, and Variant 3 does so in 59.99% of the runs.

– Variant 2 outperforms Variant 1 in 56.29% of runs.
– Finally, Variant 3 outperforms the other two variants: Variant 1 in 57.84% of the runs, and Variant 2 in 51.63% of the runs.

8.2.2 Industrial case study

Table 5 shows the $\hat{A}_{12}$ values for precision and recall in the industrial case study.

Industrial case study		
Compared approaches	**Precision**	**Recall**
Baseline vs Variant 1	0.5547	0.2813
Baseline vs Variant 2	0.6094	0.2969
Baseline vs Variant 3	0.6250	0.2031
Variant 1 vs Variant 2	0.5313	0.4688
Variant 1 vs Variant 3	0.5938	0.3750
Variant 2 vs Variant 3	0.5625	0.4219

Table 5 $\hat{A}_{12}$ statistic for Mutation Search vs. its variants in the industrial case study

In this case study, the $\hat{A}_{12}$ values for **precision** are in favor of the baseline:

– The baseline outperforms the three variants: Variant 1 in 55.47% of the runs, Variant 2 in 60.94% of the runs, and Variant 3 in 62.50% of the runs.
– Variant 1 outperforms the other two variants: Variant 2 in 53.13% of the runs, and Variant 3 in 59.38% of the runs.
– Finally, Variant 2 outperforms Variant 3 in 56.25% of the runs.

As stated in the results report, the differences between Variant 2 and Variant 3 are uncertain, since the average values are close and the deviation does not allow for a straightforwards comparison. The values of the $\hat{A}_{12}$ measurement show that Variant 2 outperforms Variant 3 in terms of precision in 56.25% of the runs. Regarding **recall**, the $\hat{A}_{12}$ values confirm the results scenario shown in the boxplot of Figure 10:

– The three variants outperform the baseline: Variant 1 does so in 71.87% of the runs, Variant 2 does so in 70.31% of the runs, and Variant 3 does so in 79.69% of the runs.
– Variant 2 outperforms Variant 1 in 53.12% of the runs.
– Finally, Variant 3 outperforms the other two variants: Variant 1 in 62.50% of the runs, and Variant 2 in 57.51% of the runs.

In this case study, it is possible to perceive that the higher deviation of the values of Variant 2 causes the following situation: while Variant 2 outperforms Variant 1 in terms of recall (albeit by a small margin), the results of the first quartile of the recall data obtained by Variant 2 are lower than those of the first quartile of the recall data obtained by Variant 1. This allows for the baseline to catch up with the first and second quartile of Variant 2. As a result, Variant 1 outperforms the baseline for a slightly greater margin than Variant 2 (71.87% against 70.31%, a 1.56% margin).

8.3 Research Question Responses

Through the results, the two research questions presented in Section 7 can be responded:

RQ_1 Do BPMN-specific variants improve the results of the generalist baselines when performing TLR between requirements and BPMN models? Yes, the results obtained by the BPMN-specific variants outperform those obtained by the generalist approaches, maintaining precision levels and increasing recall values. Hence, we can reject the null hypothesis H_1.

RQ_2 Do the baseline and variants vary on accuracy? Yes, the Quade statistical test shows that the differences in performance between the variants and the baselines are significant. In addition, we have calculated the $\hat{A}_{12}$ values that measure the differences in performance. Hence, we can reject the null hypothesis H_2.

8.4 Reception of our work

In order to obtain qualitative data from practitioners, we ran a focus group interview [61] with software engineers from our industrial partner. Through the focus group interview, we posed a series of open questions to acquire feedback from the engineers about the results obtained by the approaches in use. Specifically, we asked the engineers the following open questions: (1) How do you feel about the results of each approach? and (2) What would make you choose one approach over another of the approaches?

The engineers stated that they preferred the results obtained by the advanced variants of Mutation Search over the baselines, indicating that their results were better aligned with the reality of the case study. The software engineers indicated that a train has around 100 requirements on average, and that manually tracing those requirements to the models is a tedious and error-prone process. They stated that requirements and models evolve throughout the duration of a project, and that both artifacts rarely remain as defined in their original inception, mainly due to changes in the requirements stemming from their meetings with clients.

Theoretically, requirements and models should be synchronized at all times, but with manual traceability, it is practically impossible to guarantee the alignment of the artifacts. The engineers stated that, through our approaches, they can attain at all times and in an inexpensive manner a proposal of the relevant model elements for each requirement, which helps them check whether the requirements have been completely and correctly transported into the models.

When further questioned about the usefulness of our approaches, engineers stated that the approaches were not used only internally, but also to support the certification of the final products, that is, the trains they sell in an international context. Each country has different regulations regarding certification, but the need to accredit traceability between requirements and software is becoming more and more common. When the software is developed through Model-Driven Development techniques, traceability between requirements and models must be provided for certification purposes. The engineers stated that in these scenarios, our approaches support the traceability activity and the subsequent certification.

9 Discussion and Future Work

The following paragraphs discuss the outcomes of the paper, the conclusions that we were able to extract on the behaviour of the approaches after a thorough analysis of their results, and directions for future works.

The Linguistic Rule-Based approach obtains extremely poor results compared to the rest of the approaches: 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. In the Aggregation and Mutation Search approaches the atomic matching of text patterns is abandoned in favor of the semantic similitude of individual terms.

Moreover, model elements with little or no text can never be retrieved by the Linguistic Rule-Based approach: since there are no words, there is no pattern that can be matched. They are not retrieved by the Aggregation approach 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 approach and also in the variants, the algorithm does add these elements to the candidate fragments. Moreover, for these approaches, the addition of these elements does not penalize the results, since the *term* occurrences are not altered in any way by them. Therefore, the candidate fragments are more correct and complete, which leads to enhanced precision and recall results.

While inspecting the results obtained by our work, we noticed a series of facts that help explain the behavior of the variants against Mutation Search. First of all, we have to consider the general structure of the BPMN models and requirements that are being studied. The BPMN models in our case studies tend to have few start and end events (only one BPMN model has more than 3 of these events). Thanks to the addition of no-text model elements, start and end events with no text and flows with no text are incorporated more often than not to the model fragments. Through our implementation, it is possible for the variants that use the addition of no-text model elements to add both a start and an end event to a model fragment, specially in the larger model fragments and also in fragments belonging to smaller BPMN models where start and end events are close. However, requirements describe particular branches of the BPMN models, and it is very rare for a single requirement to correspond to a whole start-to-end model fragment. Rather, requirements tend to depict start-plus-task, task-plus-end, or task-gateway-task combinations of model elements.

Therefore, we realized that the variants of Mutation Search that utilize the addition of no-text elements may be adding both a start and an end event to model fragments where only one of them is in fact needed to represent the requirement. When this happens, two elements are added to the model fragment, one being correct and another one being incorrect, always according to the oracle. Sometimes, for task-gateway-task requirements, none of them are correct. The recall measurement only accounts for true positive and false negative classifications, so it can only increase as correct model elements are added to the solution. However, the precision measurement does account for false positive classifications, and thus, is negatively impacted by the addition of wrong model elements to the solution model fragment. Upon a closer inspection of the results, we concluded that

correct and incorrect events and flows with no text are being added in a similar proportion, not causing a significant impact on precision. Therefore, when model fragments where this phenomenon happens are chosen as candidate solutions for the requirement, recall values increase but precision values are maintained stable.

A similar issue happens when incorporating language patterns into Mutation Search. In the end, the addition of more model elements into LSI and the matching of those model elements with the text of the query benefits the larger model fragments. By their very nature, larger model fragments tend to have more text and terms than smaller ones, therefore having more matching opportunities. In addition, when adding model elements to LSI, through sheer numbers, larger model fragments have more opportunities of matching the added model elements. Requirements tend to use the full expressiveness of the natural language, incorporating as many terms from the domain as possible, so as to be better understood by modelers. Therefore, the added model elements tend to be matched by the language patterns of the requirements as well. In turn, this fact brings larger model fragments and requirements closer in similarity. As a result, the variants that use the linguistic patterns to enhance Mutation Search tend to present larger model fragments in the first positions of the LSI ranking. Larger model fragment solutions account for better recall values, but the addition of too many incorrect model elements to the solution worsens precision. However, LSI also accounts for the textual similitude of the model fragments to the requirements, so those model fragments that hold too many terms unrelated to the query are not able to maintain the similarity level and tend to be disregarded as potential solutions. The combination of these two factors leads to the selection of large model fragments with a moderate amount of incorrect elements. Once again, recall values increase, but in this case, precision values struggle to remain stable. Following these conclusions, it may be possible to further improve the process and the results by tackling the discussion points above and through the incorporation of further language patterns discussed in the literature into the TLR techniques [62, 63].

Moreover, Search-Based Software Engineering (SBSE) approaches have been gaining momentum in the community within the last few years [64, 5, 65]. SBSE approaches utilize either single objective or multi-objective genetic algorithms to maintain and evolve populations of model fragments. SBSE techniques imitate the crossovers and random mutations that occur in nature, and then select the best individuals of the generated populations through fitness functions. This process is iterated until a certain condition is met, generating several self-enhancing populations of individuals in the process. As future work, we intend to explore the application and results of SBSE approaches for TLR tasks between requirements and BPMN models. In addition, we will compare the results of those approaches against the approaches presented through the research works discussed throughout the paper. Additionally, we believe that the lack of inherent semantics in BPMN may also constitute a research challenge in SBSE, and hence we will also research how to incorporate BPMN linguistics into SBSE techniques.

Overall, from the results of our work, a reflection can be extracted on the impact of this paper for the community that is working on this novel line of research: in order to optimize the results of the TLR process, there is a need for taking in account the particularities of the different kinds of models and software artifacts in use, instead of relying on approaches that do not account for these particularities.

10 Threats to Validity

In this section, we use the classification of threats to validity of [55] to acknowledge the limitations of our approach.

Conclusion validity: This validity is concerned with the relationship between the treatment and the outcome. We want to make sure that there is a statistical relationship, that is, a statistical significance. To minimize this threat, we have utilized reliable, well-known statistical measurements such as Holm's post analysis and $\hat{A}_{12}$, utilized in the state of the art literature. We have also based our work on research questions and disproving null hypotheses. In addition, the requirements and BPMN models used in our approach were taken from an academical case study and from an industrial case study, and none of the authors of this work was involved in the generation of the data.

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). We have carried out the same natural language techniques as preprocessors over the requirements and BPMN models prior to the application of the different TLR approaches. Moreover, we have followed the same evaluation process for all the approaches. In addition, the available test cases (60 for the academic case study, and 500 for the industrial case study) represent 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 five widespread measurements: precision, recall, f-measure, MCC, and AUC. These measurements, presented in tables 2 and 3, are widely accepted in the software engineering research community. Moreover, we have used the same kinds of software artifacts in all of our case studies (requirements and BPMN models), representing the same scenarios (processes in an academic and 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? Both artifacts in use, natural language requirements and BPMN models alike, are frequently leveraged to specify all kinds of different processes, whether academic or industrial. The academic case study provides different examples from radically different domains. In addition, 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 any of those domains. In addition, Mutation Search is the tool leveraged by the industrial partner from the case study, which makes it the most representative baseline in practice. Hence, comparing the results of our approaches against Mutation Search is coherent in order to conform to practice.

Nevertheless, our results should be replicated with other case studies before assuring 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.

11 Summary and Conclusions

Traceability Links Recovery (TLR) is defined as the software engineering task that deals with the automated identification and comprehension of dependencies and relationships between software artifacts. Research so far has studied TLR between natural language requirements and BPMN models through three different generalist approaches that have provided good results when performing TLR between requirements and code. However, prior work does not account for the specific traits and particularities of the different kinds of models in use. Through this work, we extend the contributions of previous research works in the field, leveraging the particularities presented by BPMN models to enhance the baseline approaches. In that sense, the achievements of our work are threefold: (i) we identify the particularities and traits of BPMN models to incorporate to the approaches; (ii) we propose methods to leverage the aforementioned particularities; and (iii) we apply those methods to build three variants of the Mutation Search approach, specific for BPMN models.

The novel techniques are evaluated through two case studies, contrasting the results with those obtained by the baseline approaches. The novel approaches succeed at improving the results of TLR between requirements and BPMN models, maintaining precision values and increasing recall. Overall, this research proves the importance of incorporating the particularities of the software artifacts in use when performing TLR. Through a thorough analysis of the results, we have identified a series of issues that impact the performance of the proposed approaches, mainly related to the linguistics of the software artifacts in use. These issues bring to light future work possibilities in the field, which range from the enrichment of the proposed approaches through the incorporation of further language patterns stemming from the works of other authors in the field, to the development of more advanced approaches through the application of the novel trends and latest state-of-the-art approaches in Search Based Software Engineering (SBSE).

Acknowledgements [Omitted for blind review purposes]

References

1. Brambilla M, Cabot J, Wimmer M (2012) Model-driven software engineering in practice. Synthesis Lectures on Software Engineering 1(1):1–182
2. Macaulay LA (2012) Requirements engineering. Springer Science & Business Media

3. Winkler S, Pilgrim J (2010) A Survey of Traceability in Requirements Engineering and Model-Driven Development. Software and Systems Modeling (SoSyM) 9(4):529–565
4. Loniewski G, Insfran E, Abrahão S (2010) 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, pp 213–227
5. Font J, Arcega L, Haugen Ø, Cetina C (2016) 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, ACM, MODELS '16
6. Martinez J, Ziadi T, Bissyande TF, Klein J, Le Traon Y (2015) 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, pp 396–406
7. Martinez J, Ziadi T, Papadakis M, Bissyandé TF, Klein J, Le Traon Y (2018) Feature location benchmark for extractive software product line adoption research using realistic and synthetic eclipse variants. Information and Software Technology 104:46–59
8. Krüger J, Mukelabai M, Gu W, Shen H, Hebig R, Berger T (2019) Where is my feature and what is it about? a case study on recovering feature facets. Journal of Systems and Software 152:239–253
9. Chinosi M, Trombetta A (2012) BPMN: An Introduction to the Standard. Computer Standards & Interfaces 34(1):124–134
10. Oliveto R, Gethers M, Poshyvanyk D, De Lucia A (2010) On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In: 2010 IEEE 18th International Conference on Program Comprehension, IEEE, pp 68–71
11. Watkins R, Neal M (1994) Why and How of Requirements Tracing. IEEE Software 11(4):104–106
12. Ghazarian A (2010) A Research Agenda for Software Reliability. IEEE Reliability Society 2009 Annual Technology Report
13. Rempel P, Mäder P (2017) Preventing Defects: the Impact of Requirements Traceability Completeness on Software Quality. IEEE Transactions on Software Engineering 43(8):777–797
14. Zhang Y, Witte R, Rilling J, Haarslev V (2008) Ontological Approach for the Semantic Recovery of Traceability Links Between Software Artefacts. IET software 2(3):185–203
15. Zowghi D, Jin Z (2011) Requirements engineering. Springer
16. Gotel OC, Finkelstein C (1994) An Analysis of the Requirements Traceability Problem. In: Proceedings of the First International Conference on Requirements Engineering, IEEE, pp 94–101
17. Spanoudakis G, Zisman A (2005) Software Traceability: a Roadmap. Handbook of Software Engineering and Knowledge Engineering 3:395–428
18. Schlutter A, Vogelsang A (2020) Trace link recovery using semantic relation graphs and spreading activation. In: 2020 IEEE 28th International Requirements Engineering Conference (RE), IEEE, pp 20–31
19. Madala K, Piparia S, Blanco E, Do H, Bryce R (2021) Model elements identification using neural networks: a comprehensive study. Requirements Engi-

neering 26:67–96
20. Parizi RM, Lee SP, Dabbagh M (2014) Achievements and Challenges in State-of-the-Art Software Traceability between Test and Code Artifacts. IEEE Transactions on Reliability 63(4):913–926
21. Rubin J, Chechik M (2013) A Survey of Feature Location Techniques. In: Domain Engineering, Springer, pp 29–58
22. Lapeña R, Font J, Cetina C, Pastor Ó (2018) Exploring new directions in traceability link recovery in models: The process models case. In: International Conference on Advanced Information Systems Engineering, Springer, pp 359–373
23. Lapeña R, Pérez F, Cetina C, Pastor Ó (2019) Improving Traceability Links Recovery in Process Models Through an Ontological Expansion of Requirements. In: International Conference on Advanced Information Systems Engineering, Springer, pp 261–275
24. Pérez F, Font J, Arcega L, Cetina C (2019) Collaborative Feature Location in Models through Automatic Query Expansion. Automated Software Engineering 26(1):161–202
25. Eaddy M, Aho AV, Antoniol G, Guéhéneuc YG (2008) Cerberus: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis, and Program Analysis. In: ICPC 2008 conference, IEEE
26. Eaddy M, Aho A, Murphy GC (2007) Identifying, Assigning, and Quantifying Crosscutting Concerns. In: Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques, p 2
27. Marcus A, Maletic JI (2003) Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing. In: Proceedings of the 25th International Conference on Software Engineering, IEEE
28. Antoniol G, Canfora G, Casazza G, De Lucia A, Merlo E (2002) Recovering Traceability Links between Code and Documentation. IEEE Transactions on Software Engineering 28(10):970–983
29. Zisman A, Spanoudakis G, Pérez-Miñana E, Krause P (2003) Tracing Software Requirements Artifacts. In: Software Engineering Research and Practice, pp 448–455
30. De Lucia A, Fasano F, Oliveto R, Tortora G (2004) Enhancing an Artefact Management System with Traceability Recovery Features. In: Proceedings of the 20th IEEE International Conference on Software Maintenance, IEEE, pp 306–315
31. Eder S, Femmer H, Hauptmann B, Junker M (2015) Configuring Latent Semantic Indexing for Requirements Tracing. In: Proceedings of the 2nd International Workshop on Requirements Engineering and Testing
32. Marcén AC, Lapeña R, Pastor Ó, Cetina C (2020) 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 163:110519
33. Sultanov H, Hayes JH (2010) Application of Swarm Techniques to Requirements Engineering: Requirements Tracing. In: 18th IEEE International Requirements Engineering Conference
34. Duan C, Cleland-Huang J (2007) Clustering Support for Automated Tracing. In: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering

35. Falessi D, Cantone G, Canfora G (2013) Empirical Principles and an Industrial Case Study in Retrieving Equivalent Requirements via Natural Language Processing Techniques. IEEE Transactions on Software Engineering 39(1):18–44
36. Arora C, Sabetzadeh M, Goknil A, Briand LC, Zimmer F (2015) Change Impact Analysis for Natural Language Requirements: An NLP Approach. In: IEEE 23rd International Requirements Engineering Conference
37. Pudlitz F, Brokhausen F, Vogelsang A (2019) Extraction of system states from natural language requirements. In: 2019 IEEE 27th International Requirements Engineering Conference (RE), IEEE, pp 211–222
38. Deshpande G, Motger Q, Palomares C, Kamra I, Biesialska K, Franch X, Ruhe G, Ho J (2020) Requirements dependency extraction by integrating active learning with ontology-based retrieval. In: 2020 IEEE 28th International Requirements Engineering Conference (RE), IEEE, pp 78–89
39. Seqerloo AY, Amiri MJ, Parsa S, Koupaee M (2019) Automatic test cases generation from business process models. Requirements Engineering 24(1):119–132
40. Moitra A, Siu K, Crapo AW, Durling M, Li M, Manolios P, Meiners M, McMillan C (2019) Automating requirements analysis and test case generation. Requirements Engineering 24(3):341–364
41. Reinhartz-Berger I, Kemelman M (2020) Extracting core requirements for software product lines. Requirements Engineering 25(1):47–65
42. Qian C, Wen L, Kumar A, Lin L, Lin L, Zong Z, Wang J, et al. (2020) An approach for process model extraction by multi-grained text classification. In: International Conference on Advanced Information Systems Engineering, Springer, pp 268–282
43. Rebmann A, van der Aa H (2021) Extracting semantic process information from the natural language in event logs. In: International Conference on Advanced Information Systems Engineering, Springer, pp 57–74
44. Sànchez-Ferreres J, van der Aa H, Carmona J, Padró L (2018) Aligning Textual and Model-Based Process Descriptions. Data & Knowledge Engineering 118:25–40
45. Mendling J, Leopold H, Thom LH, van der Aa H (2019) Natural Language Processing with Process Models (NLP4RE Report Paper). In: REFSQ Workshops, CEUR-WS.org, CEUR Workshop Proceedings, vol 2376
46. Klinkmüller C, Weber I, Mendling J, Leopold H, Ludwig A (2013) Increasing Recall of Process Model Matching by Improved Activity Label Matching. In: Business Process Management, Springer, pp 211–218
47. Leopold H, Mendling J, Polyvyanyy A (2014) Supporting Process Model Validation through Natural Language Generation. IEEE Transactions on Software Engineering 40(8):818–840
48. Spanoudakis G, Zisman A, Pérez-Minana E, Krause P (2004) Rule-Based Generation of Requirements Traceability Relations. Journal of Systems and Software 72(2):105–127
49. Leech G, Garside R, Bryant M (1994) CLAWS4: the Tagging of the British National Corpus. In: Proceedings of the 15th Conference on Computational Linguistics - Volume 1, Association for Computational Linguistics
50. Hulth A (2003) 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
51. Plisson J, Lavrac N, Mladenic D, et al. (2004) A Rule Based Approach to Word Lemmatization. In: Proceedings of the 7th International Multi-Conference Information Society, Citeseer, vol 1, pp 83–86
52. Landauer TK, Foltz PW, Laham D (1998) An Introduction to Latent Semantic Analysis. Discourse Processes 25(2-3):259–284
53. Marcus A, Sergeyev A, Rajlich V, Maletic J (2004) An Information Retrieval Approach to Concept Location in Source Code. In: Proceedings of the 11th Working Conference on Reverse Engineering, pp 214–223, DOI 10.1109/WCRE.2004.10
54. Salman HE, Seriai A, Dony C (2014) 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
55. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in Software Engineering. Springer Science & Business Media
56. Salton G, McGill MJ (1986) Introduction to Modern Information Retrieval. McGraw-Hill, Inc., New York, NY, USA
57. Arcuri A, Briand L (2014) A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. Software Testing, Verification and Reliability 24(3):219–250
58. García S, Fernández A, Luengo J, Herrera F (2010) Advanced Nonparametric Tests for Multiple Comparisons in the Design of Experiments in Computational Intelligence and Data Mining: Experimental Analysis of Power. Information Sciences 180(10):2044–2064
59. Conover WJ (1998) Practical nonparametric statistics, vol 350. John Wiley & Sons
60. Vargha A, Delaney HD (2000) A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. Journal of Educational and Behavioral Statistics
61. Krueger RA, Casey MA (2014) Focus groups: A practical guide for applied research. Sage publications
62. van der Aa H, Di Ciccio C, Leopold H, Reijers HA (2019) Extracting Declarative Process Models from Natural Language. In: International Conference on Advanced Information Systems Engineering, Springer, pp 365–382
63. Friedrich F, Mendling J, Puhlmann F (2011) Process Model Generation from Natural Language Text. In: International Conference on Advanced Information Systems Engineering, Springer, pp 482–496
64. Font J, Arcega L, Haugen Ø, Cetina C (2016) Feature Location in Model-Based Software Product Lines Through a Genetic Algorithm. In: Proceedings of the 15th International Conference on Software Reuse: Bridging with Social-Awareness, ICSR 2016, Limassol, Cyprus
65. Affenzeller M, Winkler SM, Wagner S, Beham A (2009) Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications. CRC Press