UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Design of a Machine Learning-based Approach for Fragment Retrieval on Models

November of 2020

Author: Ana Cristina Marcén Terraza

Directors: Dr. Óscar Pastor López
Dr. Carlos Cetina Englada

Acknowledgements

First of all, I would like to thank my two directors, Dr. Óscar Pastor and Dr. Carlos Cetina for the opportunity to perform this Thesis. Thanks Óscar for helping me to further consider the research problems, and about all, the explanations. Thanks Carlos for having always the door open to advise me about technical, professional, or even, personal aspects. Thanks for being the strength, which have encouraged me to improve as a researcher.

Secondly, I would like to thank the Generalitat Valenciana under the project ACIF/2018/171. Thanks for providing the opportunity and strength to support this work.

Thirdly, I want to express my *holistic* gratitude to the people of the PROS Research Group. Thanks for being the strength, to make me feel so welcome being away from home.

Fourthly, I would like to thank the people of SVIT Research Group and of the School, for those break times with a coffee and many laughs. Special thanks to Francisca Pérez, Jaime Font, Lorena Arcega, and Raúl Lapeña, nothing like a conversation about contracts with punch proof, paper parties, or board games to recover the loss motivation. Thanks for these memorable moments, which have been the strength to overcome any difficulty.

Moreover, I do not want to finish this section without thanking my friends. Special thanks to Damián, Mabel, Sonia, Josema, Leo, and Alba, for being the strength to keep being myself, to not forget my skills and my hobbies.

Finally, the last thank is for my family. It is difficult to express through words a suitable gratitude for your unconditional support, not only during these years, but also during my whole life. Without this support and your love, I would not be an engineer today, let's not talk about being a Doctor. Therefore, thanks to my parents to always believe in me, for their patience and guidance, and for encouraging me to fight for my goals, personal and professional. Thanks to my sisters and my brother, without them my life would be surely more quiet, but also extremely boring. Thanks Sara for your endless smile and your artist perspective, at anytime and anywhere (except in a tent during a storm). Thanks María for so many details that make my days more enjoyable: the cool milk in the morning, the green *Lacasitos*, the calls at the end of the day. Thanks José for those visits that bright my weekends with interesting meals and time to rest. Thanks grandmother for always being aware of my advances and goals and trying to help me as you can. For all of this, and for a lot of other things that do not fit in these acknowledgements, thanks for the strength to live and to grow as person and as researcher within a great family.

Kyuta - *The word "Strong" has lots of meanings, huh?*

Kumatetsu - *You have to find the meaning for yourself!*

The boy and the Beast

Abstract

Machine Learning (ML) is known as the branch of artificial intelligence that gathers statistical, probabilistic, and optimization algorithms, which learn empirically. ML can exploit the knowledge and the experience that have been generated for years to automatically perform different processes. Therefore, ML has been applied to a wide range of research areas, from medicine to software engineering.

In fact, in software engineering field, up to an 80% of a system's lifetime is spent on the maintenance and evolution of the system. The companies, that have been developing these software systems for a long time, have gathered a huge amount of knowledge and experience. Therefore, ML is an attractive solution to reduce their maintenance costs exploiting the gathered resources. Specifically, Traceability Link Recovery, Bug Localization, and Feature Location are amongst the most common and relevant tasks when maintaining software products. To tackle these tasks, researchers have proposed a number of approaches. However, most research focus on traditional methods, such as Latent Semantic Indexing, which does not exploit the gathered resources. Moreover, most research targets code, neglecting other software artifacts such as models.

In this dissertation, we present an ML-based approach for fragment retrieval on models (FRAME). The goal of this approach is to retrieve the model fragment which better realizes a specific query in a model. This allows engineers to retrieve the model fragment, which must be traced, fixed, or located for software maintenance. Specifically, the FRAME approach combines evolutionary computation and ML techniques.

In the FRAME approach, an evolutionary algorithm is guided by ML to effectively extract model fragments from a model. These model fragments are then assessed through ML techniques. To learn how to assess them, ML techniques takes advantage of the companies' knowledge (retrieved model fragments) and experience. Then, based on what was learned, ML techniques determine which model fragment better realizes a query. However, model fragments are not understandable for most ML techniques. Therefore, the proposed approach encodes the model fragments through an ontological evolutionary encoding. In short, the FRAME approach is designed to extract model fragments, encode them, and assess which one better realizes a specific query.

The approach has been evaluated in our industrial partner (CAF, an international provider of railway solutions) and compared to the most common and recent approaches. The results show that the FRAME approach achieved the best results for most performance indicators, providing a mean precision value of 59.91%, a recall value of 78.95%, a combined F-measure of 62.50%, and a MCC (Matthews correlation coefficient) value of 0.64. Leveraging retrieved model fragments, the FRAME approach is less sensitive to tacit knowledge and vocabulary mismatch than the approaches based on semantic information. However, the approach is limited by the availability of the retrieved model fragments to perform the learning. These aspects are further discussed, after the statistical analysis of the results, which assesses the magnitude of the improvement in comparison to the other approaches.

Resumen

El aprendizaje automático (ML por sus siglas en inglés) es conocido como la rama de la inteligencia artificial que reúne algoritmos estadísticos, probabilísticos y de optimización, que aprenden empíricamente. ML puede aprovechar el conocimiento y la experiencia que se han generado durante años en las empresas para realizar automáticamente diferentes procesos. Por lo tanto, ML se ha aplicado a diversas áreas de investigación, que estudian desde la medicina hasta la ingeniería del software.

De hecho, en el campo de la ingeniería del software, el mantenimiento y la evolución de un sistema abarca hasta un 80% de la vida útil del sistema. Las empresas, que se han dedicado al desarrollo de sistemas software durante muchos años, han acumulado grandes cantidades de conocimiento y experiencia. Por lo tanto, ML resulta una solución atractiva para reducir sus costos de mantenimiento aprovechando los recursos acumulados. Específicamente, la Recuperación de Enlaces de Trazabilidad, la Localización de Errores y la Ubicación de Características se encuentran entre las tareas más comunes y relevantes para realizar el mantenimiento de productos software. Para abordar estas tareas, los investigadores han propuesto diferentes enfoques. Sin embargo, la mayoría de las investigaciones se centran en métodos tradicionales, como la indexación semántica latente, que no explota los recursos recopilados. Además, la mayoría de las investigaciones se enfocan en el código, descuidando otros artefactos de software como son los modelos.

En esta tesis, presentamos un enfoque basado en ML para la recuperación de fragmentos en modelos (FRAME). El objetivo de este enfoque es recuperar el fragmento del modelo que realiza mejor una consulta específica. Esto permite a los ingenieros recuperar el fragmento que necesita ser trazado, reparado o ubicado para el mantenimiento del software. Específicamente, FRAME combina la computación evolutiva y las técnicas ML.

En FRAME, un algoritmo evolutivo es guiado por ML para extraer de manera eficaz distintos fragmentos de un modelo. Estos fragmentos son posteriormente evaluados mediante técnicas ML. Para aprender a evaluarlos, las técnicas ML aprovechan el conocimiento (fragmentos recuperados de modelos) y la experiencia que las empresas han generado durante años. Basándose en lo aprendido, las técnicas ML determinan qué fragmento del modelo realiza mejor una consulta. Sin embargo, la mayoría de las técnicas ML no pueden entender los fragmentos de los modelos. Por lo tanto, antes de aplicar las técnicas ML, el enfoque propuesto codifica los fragmentos a través de una codificación ontológica y evolutiva. En resumen, FRAME está diseñado para extraer fragmentos de un modelo, codificarlos y evaluar cuál realiza mejor una consulta específica.

El enfoque ha sido evaluado a partir de un caso real proporcionado por nuestro socio industrial (CAF, un proveedor internacional de soluciones ferroviarias). Además, sus resultados han sido comparados con los resultados de los enfoques más comunes y recientes. Los resultados muestran que FRAME obtuvo los mejores resultados para la mayoría de los indicadores de rendimiento, proporcionando un valor medio de precisión igual a 59.91%, un valor medio de exhaustividad igual a 78.95%, una valor-F medio igual a 62.50% y un MCC (Coeficiente de Correlación Matthews) medio igual a 0.64. Aprovechando los fragmentos recuperados de los modelos, FRAME es menos sensible al conocimiento tácito y al desajuste de vocabulario que los enfoques basados en información semántica. Sin embargo, FRAME está limitado por la disponibilidad de fragmentos recuperados para llevar a cabo el aprendizaje automático. Esta tesis presenta una discusión más amplia de estos aspectos así como el análisis estadístico de los resultados, que evalúa la magnitud de la mejora en comparación con los otros enfoques.

Resum

L'aprenentatge automàtic (ML per les seues sigles en anglés) és conegut com la branca de la intel·ligència artificial que reuneix algorismes estadístics, probabilístics i d'optimització, que aprenen empíricament. ML pot aprofitar el coneixement i l'experiència que s'han generat durant anys en les empreses per a realitzar automàticament diferents processos. Per tant, ML s'ha aplicat a diverses àrees d'investigació, que estudien des de la medicina fins a l'enginyeria del programari.

De fet, en el camp de l'enginyeria del programari, el manteniment i l'evolució d'un sistema abasta fins a un 80% de la vida útil del sistema. Les empreses, que s'han dedicat al desenvolupament de sistemes programari durant molts anys, han acumulat grans quantitats de coneixement i experiència. Per tant, ML resulta una solució atractiva per a reduir els seus costos de manteniment aprofitant els recursos acumulats. Específicament, la Recuperació d'Enllaços de Traçabilitat, la Localització d'Errors i la Ubicació de Característiques es troben entre les tasques més comunes i rellevants per a realitzar el manteniment de productes programari. Per a abordar aquestes tasques, els investigadors han proposat diferents enfocaments. No obstant això, la majoria de les investigacions se centren en mètodes tradicionals, com la indexació semàntica latent, que no explota els recursos recopilats. A més, la majoria de les investigacions s'enfoquen en el codi, descurant altres artefactes de programari com són els models.

En aquesta tesi, presentem un enfocament basat en ML per a la recuperació de fragments en models (approach). L'objectiu d'aquest enfocament és recuperar el fragment del model que realitza millor una consulta específica. Això permet als enginyers recuperar el fragment que necessita ser traçat, reparat o situat per al manteniment del programari. Específicament, FRAME combina la computació evolutiva i les tècniques ML.

En FRAME, un algorisme evolutiu és guiat per ML per a extraure de manera eficaç diferents fragments d'un model. Aquests fragments són posteriorment avaluats mitjançant tècniques ML. Per a aprendre a avaluar-los, les tècniques ML aprofiten el coneixement (fragments recuperats de models) i l'experiència que les empreses han generat durant anys. Basant-se en l'après, les tècniques ML determinen quin fragment del model realitza millor una consulta. No obstant això, la majoria de les tècniques ML no poden entendre els fragments dels models. Per tant, abans d'aplicar les tècniques ML, l'enfocament proposat codifica els fragments a través d'una codificació ontològica i evolutiva. En resum, FRAME està dissenyat per a extraure fragments d'un model, codificar-los i avaluar quin realitza millor una consulta específica.

L'enfocament ha sigut avaluat a partir d'un cas real proporcionat pel nostre soci industrial (CAF, un proveïdor internacional de solucions ferroviàries). A més, els seus resultats han sigut comparats amb els resultats dels enfocaments més comuns i recents. Els resultats mostren que FRAME va obtindre els millors resultats per a la majoria dels indicadors de rendiment, proporcionant un valor mitjà de precisió igual a 59.91%, un valor mitjà d'exhaustivitat igual a 78.95%, una valor-F mig igual a 62.50% i un MCC (Coeficient de Correlació Matthews) mig igual a 0.64. Aprofitant els fragments recuperats dels models, FRAME és menys sensible al coneixement tàcit i al desajustament de vocabulari que els enfocaments basats en informació semàntica. No obstant això, FRAME està limitat per la disponibilitat de fragments recuperats per a dur a terme l'aprenentatge automàtic. Aquesta tesi presenta una discussió més àmplia d'aquests aspectes així com l'anàlisi estadística dels resultats, que avalua la magnitud de la millora en comparació amb els altres enfocaments.

Contents

List of Figures

LIST OF TABLES

PART I

Introduction

The only way to achieve the impossible is to believe it is possible.

Alice in Wonderland

Chapter 1

INTRODUCTION

Contents

1.1 Motivation

Amongst the most common and relevant tasks in the Software Engineering field, especially when maintaining software products, are Traceability Link Recovery, Bug Localization, and Feature Location [1, 2, 3, 4]. Since up to an 80% of a system's lifetime is spent on the maintenance and evolution of the system [5], there is a great demand for Traceability Link Recovery, Bug Localization, and Feature Location approaches that can help developers to retrieve relevant software artifacts in software products.

In Model Driven Engineering, models are the main software artifacts. Models raise the abstraction level using concepts that are much less bound to the underlying implementation and technology and are much closer to the problem domain [6]. The practice of Model Driven Engineering has proved to increase efficiency and effectiveness in software development [6]. In fact, in industrial contexts, fostering modeling efforts brings benefits in order to improve productivity, while ensuring quality and performance [6].

Therefore, in a model-driven industrial context, companies tend to have a myriad of products with large and complex models behind [7]. In this context, the software engineers have to consume high amounts of time and effort in order to identify the model elements that have to be maintained or evolved. Figure 1.1 depicts an example, taken from a real-world train, specified using the Domain Specific Language (DSL) that formalizes the railway control and management of the products manufactured by our industrial partner. Specifically, on the left of the figure, the example presents a product model, where a software engineer needs to identify the model elements related to a query. After the fragment retrieval process, the example presents a model fragment. A model fragment is a subset of one or more elements, which belong to the product model. In fragment retrieval, a model fragment contains the model elements identified by the software engineer regarding the query. In the example, the model fragment is highlighted using a gray dotted shape, on the right of the figure.

Although the example of Figure 1.1 makes that the manual retrieval process seems easy, it is important to remember that the figure shows an example and the real conditions differ from that. Basically the product models are more large and complex than the product model in the figure. Moreover, the models are created and maintained over long periods of time by different software engineers, and the engineers in charge of the software maintenance tasks (Traceability Link Recovery, Bug Localization, and Feature Location) often lack knowledge over the entirety of the product details. Under these

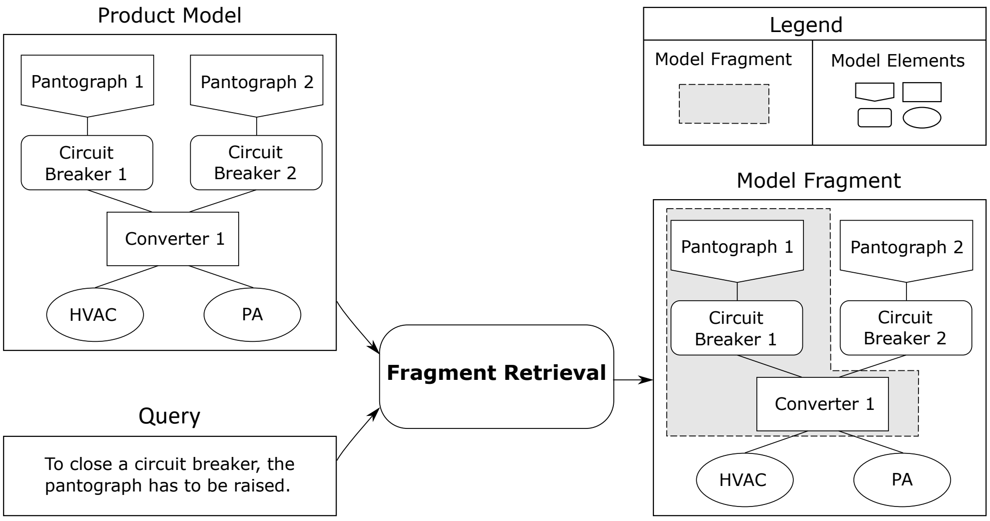


Figure 1.1: Example of a product model and a model fragment for fragment retrieval in software maintenance tasks on models

conditions, fragment retrieval in software maintenance tasks consumes high amounts of time and effort, without guaranteeing good results.

In addition to the engineers' experience, the dimensions of the data set and the collaboration of several engineers are also relevant factors for time and effort costs. Suppose we ask to a group of 19 domain experts to manually retrieve the model elements that correspond to the 121 queries of a data set provided by our industrial partner. Taking into account that the data set comprises a family of product models with 23 models of 1200 model elements, at least 27,600 model elements should be evaluated. Moreover, since each model element has about 15 properties, about 414,000 properties should be considered. Assuming that a domain expert only needs 1 second to consider a property of a model element, the domain expert would need 4.79 days to manually locate each query. Considering the 121 queries and the 19 domain experts, the result is 30.17 years [8].

Considering these numbers, an approach that automatically retrieves model fragments is strongly needed [7]. To address this challenge, we propose a Machine Learning-based approach for fragment retrieval. In that approach, Machine Learning (ML) techniques are the key to exploit the knowledge (manually retrieved model fragments) and the experience that have been generated in software companies for years. Learning from this knowledge and experience, the approach is designed to automate the fragment retrieval process in software maintenance tasks, such as Traceability Link Recovery, Bug Localization, or Feature Location.

1.2 Dissertation Objectives

Information retrieval approaches are being widely used for Traceability Link Recovery, Bug Localization, and Feature Location tasks, in order to maintain and evolve the software [1, 2, 3, 4]. However, there is a need for approaches that target models as the main software artifacts [9, 10, 11]. In this dissertation, we move towards this direction tackling three objectives: review the works related to the application of ML techniques for software maintenance tasks on models (OB1); provide a ML-based approach to automatically perform fragment retrieval in software maintenance tasks on models (OB2); and validate the contribution of this research in an industrial context (OB3).

Following the structure provided by Wieringa [12], the research questions (RQs) are divided into two categories: knowledge questions are asked to gather information about the world, and design problems call for the design of an artifact that will improve a problem context and contributes to answer knowledge questions.

To review the works related to the application of ML techniques for the software maintenance tasks on models (OB1), one knowledge question was considered. Specifically, this question focuses on gathering information about the domain of interest:

RQ1: What other approaches apply ML techniques for software maintenance tasks on models?

Once the context of the problem has been determined, we have to tackle the design of the ML-based approach for fragment retrieval in software maintenance tasks on models (OB2). This leads to a set of design problems:

RQ2: How to apply ML techniques on models?

When ML techniques are applied, objects being observed are characterized through a format, which is understandable for the techniques. For example, in case of weather forecasting, objects being observed can be daily analysis of the weather. The weather in a specific day is characterized by means of the minimum and maximum temperature, the wind speed, the wind direction, the relative humidity, or the probability of precipitation (see Figure 1.2). All these characteristics are collected by different sensors, and their values are directly used for the ML techniques to predict the weather of the following day.

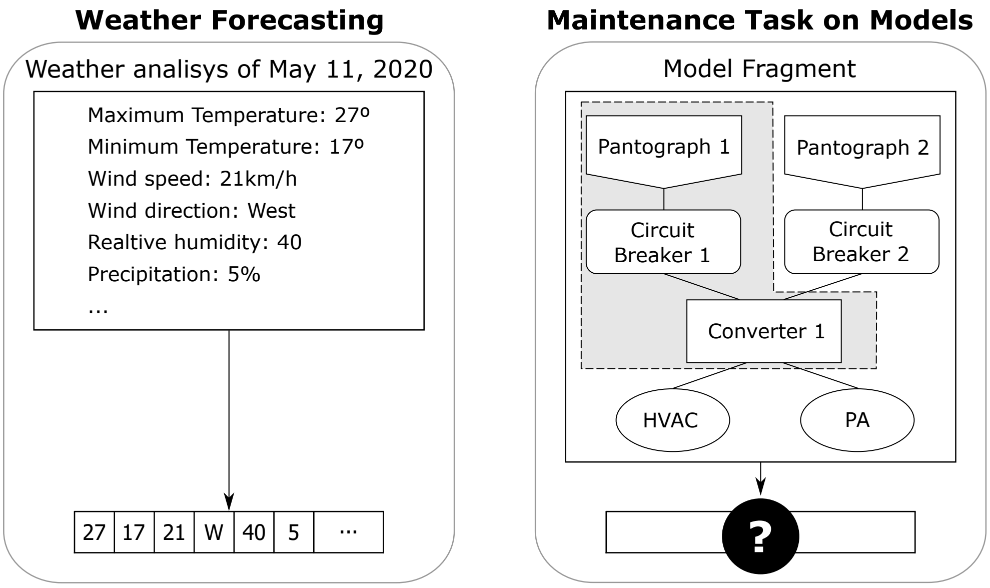


Figure 1.2: Design problem behind RQ2: Characterization of model fragments

In Model Driven Engineering, objects being observed are model fragments. However, how to characterize a model fragment is not so clear as in weather forecasting (see Figure 1.2). *Should we use the frequency of a specific model element? Should we use the relations among the model elements?* These kinds of questions have not a clear answer until now. Therefore, the second research question focused on this design problem.

RQ3: How to assess model fragments through ML techniques?

In fragment retrieval, it is necessary to determine which model fragment of a product model better realizes a specific query. However, the same product model contains different model fragments. The left part of Figure 1.3 shows an example of two different model fragments that belong to the same product model. Both model fragments contain five model elements, but the model elements are different. The *Model Fragment 1* is composed of one pantograph, one circuit breaker, one converter, one connection between the pantograph and the circuit breaker, and one connection between the circuit breaker and the converter. On the other hand, the *Model Fragment 2* is composed of two circuit breakers, one converter, and two connections that connect the circuits breaker to the converter.

To determine which model fragment better realizes the query in the figure, the model fragments have to be assessed. How to assess the model fragments is the design problem addressed by the third research question.

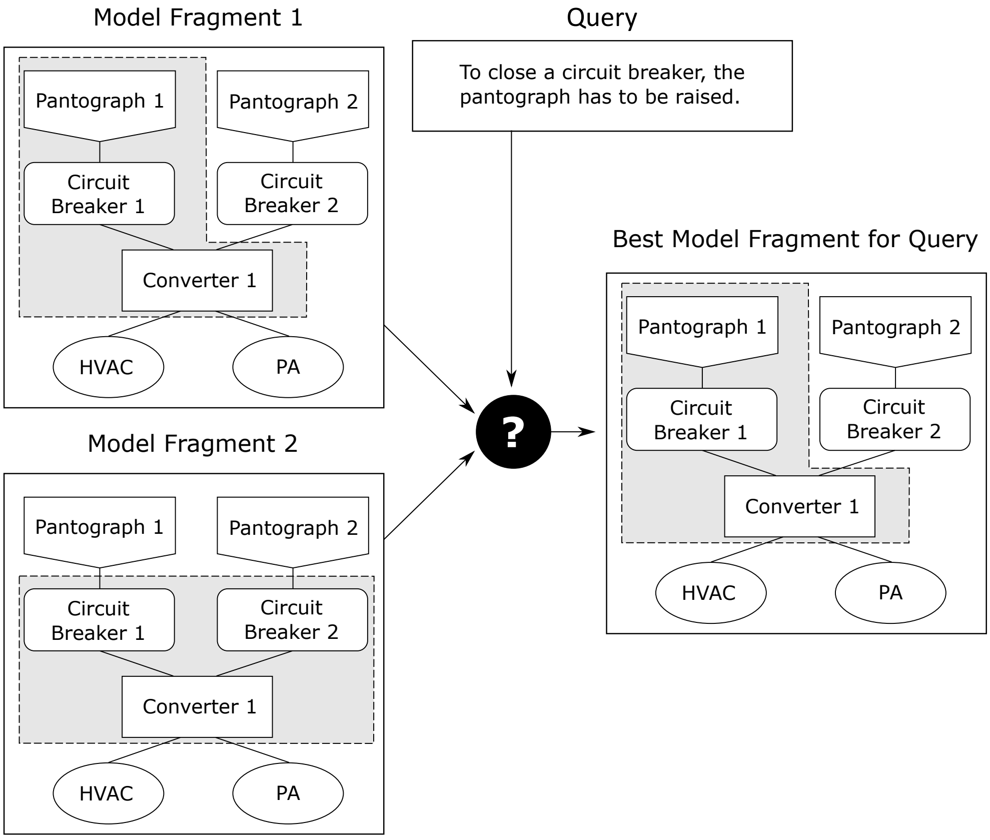


Figure 1.3: Design problem behind RQ3: Assessing model fragments regarding a query

RQ4: How to extract model fragments from a model?

The last research question for OB2 (provide a ML-based approach for fragment retrieval) focused on the extraction of model fragments[1]. Models frag-

[1]At this point, it is important to highlight that a model fragment is not extracted from its parent model as a new isolated model. The model fragment is used to identify elements of the model that are relevant for a query (i.e. a requirement, a bug, or a feature). This could be understood as

ments contain one or more model elements, than can be or connected among them. Therefore, a lot of model fragments can be extracted from a product model. For example, a product model of our industrial partner have about $4672x10^{196}$ different model fragments. Even with an automatic approach, to assess all these model fragments would take months of work.

Therefore, the fourth research question focused on researching how to extract the model fragments in order to assess only the most relevant model fragments for a specific query instead of assessing the whole set of model fragments from a product model (see Figure 1.4).

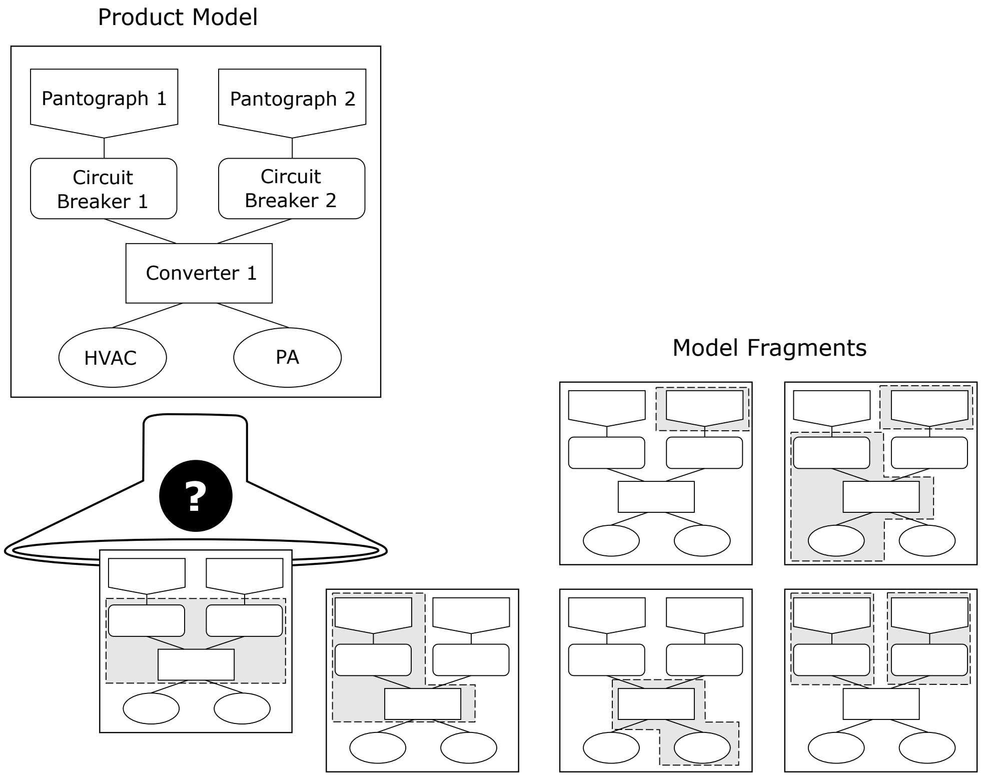


Figure 1.4: Design problem behind RQ4: Extraction of model fragments from a model

Moreover, to validate the contribution of this research in an industrial context (OB3), one more research question was addressed:

highlighting model elements of the model (that is, no new artifact is created). Different combinations of model elements can be highlighted and considered as model fragments.

RQ5: What results does the designed approach achieve in comparison to other approaches for software maintenance tasks on models?

1.3 Dissertation Contribution

In response to the RQ1, we present a preliminary systematic review of the literature (see Chapter 3). This systematic review identifies ML-based approaches for software maintenance tasks on models. Specifically, the systematic review focuses on Traceability Link Recovery, Bug Localization, and Feature Location, because these tasks require the retrieval of software artifacts. However, the systematic review not only focuses on models, but it also considers other software artifacts, such as source code.

In response to RQ2, we present an ontological evolutionary encoding [13, 9] (see Chapter 4). The encoding characterizes model fragments selecting the most suitable set of characteristics to describe the model fragments. This encoding allows ML techniques to understand the model fragments, so that the ML techniques can be applied. Therefore, this encoding allows to apply ML techniques on model fragments in the proposed approach (FRAME).

In response to RQ3, we present a fitness function [13, 9] (see Chapter 5). The fitness function takes advantage of ML techniques to determine what model fragment better realizes an specific query. This function allows to assess model fragments in the proposed approach (FRAME).

In response to RQ4, we propose the use of an evolutionary algorithm to extract model fragments from a model [14] (see Chapter 6). Instead of assessing all the model fragments of a model, the evolutionary algorithm allows to guide the extraction towards the model fragments which are more related with the query. This evolutionary algorithm allows to guide the extraction of model fragments in the proposed approach (FRAME).

In response to RQ5, we compare the results of our ML-based approach to the results of five different approaches [14] (see Chapter 8). Specifically, the evaluation considers (1) the two traditional approaches that are based on Natural Language Processing and obtain the best results for Traceability Links Recovery on models, (2) two deep learning techniques that have also successfully been applied in Traceability Link Recovery in some recent works, and (3) an approach that explore the search space by means of brute-force. Furthermore, as part of this research question, we extend the comparison among the approaches considering other aspects, such as the input artifacts. Specifically, we provide a discussion about the prerequisites and properties of the approaches and the specific advantages that our approach has over the other approaches.

It is noteworthy that we have collaborated with other researchers to address some problems which are related to our research. Therefore, although the main contributions of this dissertation are presented in [13, 9, 14], [15] and [16] also present relevant contributions for this dissertation. [15] extends existing Feature Location approaches based on Information Retrieval and Linguistic rules to locate features in models. In this work, our contribution was mainly related to apply the Linguistic Rule-Based approach in railway domain, which is fundamental to answer the RQ5 in this dissertation (see Chapter 7). [16] proposes using five measurements (size, volume, density, multiplicity, and dispersion) to report the location problems. In this work, our contribution was related to the three measurements for model fragments: density, multiplicity, and dispersion. This contribution is also fundamental for RQ5, not only to report but also to design the evaluation of this dissertation (see Chapter 7).

In addition, we have evaluated the presented contributions with our industrial partners, applying them to industrial product models. However, this evaluation has only taken into consideration the performance of the proposed approach. We are still working on the evaluation of the benefits and the satisfaction from the engineers' perspective. Furthermore, the contributions have been developed under National and International research projects aligned with the research performed in this dissertation. The contributions have been shared with the community in the form of conference and journal peer-reviewed publications. Finally, we have identified some challenges that remain unaddressed in this dissertation and that constitute our ongoing research.

1.4 Dissertation Overview

Figure 1.5 shows an overview of the work performed as part of this dissertation. It is structured into seven different rows: (row 1) identifies the challenge that is addressed; (row 2) shows the research questions about the challenge; (row 3) shows the solutions proposed in this dissertation for each research question; (row 4) shows the chapters where the solutions are presented; (row 5) lists the scientific publications generated; (row 6) lists the research projects where the work has been contributed to; (row 7) shows the industrial partner where the solution has been matured and evaluated.

Challenge	Machine Learning for Fragment Retrieval on Models				
Research Questions	**RQ1:** ML techniques in software maintenance tasks on models	**RQ2:** Characterization of model fragments	**RQ3:** Assessing model fragments regarding a query	**RQ4:** Extraction of model fragments from a model	**RQ5:** Comparison of our approach to other approaches
Solution proposed	Systematic review of the literature	Fragment Retrieval Approach based on Machine Learning and Evolutionary Algorithms (FRAME): Ontological Evolutionary Encoding	Fitness Function	Evolutionary algorithm	Evaluation of the approaches in a real case
Dissertation	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 8
Publications		REVE'17	ER'17	JSS'20	CoopIS'17, MODELS'18
Funded research projects	ACIF:	Ontological and cechnological support for the development of Big Data applications Generalitat Valencia Local Spanish funds - ACIF/2018/171			
	VARIAMOS:	Model-Driven Variability Extraction for Software Product Line Adoption Spanish National R+D+i Plan and ERDF funds - TIN2015-64397-R			
	ALPS:	Intelligent Evolutive Assistants to initiate Software Product Lines Spanish National R+D+i Plan and ERDF funds - RTI2018-096411-B-I00			
	DataME:	A model driven software production method for Big Data applications Spanish National R+D+i Plan and ERDF funds - TIN2016-80811-P			
	REVaMP^2:	Round-trip Engineering and Variability Management Platform and Process Information Technology for European Advancement - ITEA 3 Call 2			
Industrial partner	**CAF:** Variability modeling, code generation and evolution for railway systems' software				

Figure 1.5: Overview of the work performed as part of the dissertation

For the challenge of ML for fragment retrieval on models, five research questions are identified. The solution for the first research question (RQ1) is a systematic review of the literature, which is addressed in Chapter 3. The solutions for the following two research questions (RQ2 and RQ3) contribute to the design of the approach. Specifically, the solution for RQ2 is an ontological evolutionary encoding and the solution for RQ3 is a fitness function to assess model fragments. These two solutions are addressed in Chapters 4 and 5 respectively, and were presented in the publications REVE'17 [13] and ER'17 [9]. The fourth research question (RQ4) completes the design of the approach, using an evolutionary algorithm to extract model fragments from a model. This solution is addressed in Chapter 6 and was presented in the publication JSS'20 [14]. Finally, the solution for the last research question (RQ5) is the comparison of the results obtained for different approaches, including ours. This solution is addressed in Chapter 8 and was presented in the publication JSS'20 [14]. Moreover, in publications CoopIS'17 and MODELS'18, we present the first results in railway domain applying the Linguistic Rule-Based approach and the measurements for model fragments, which are used in the evaluation to answer the RQ5.

The work presented in this dissertation contributed to five projects: (ACIF) a Spanish local research grant whose objective is the definition of the support for the development of Big Data applications; (VARIAMOS) a Spanish national research project whose objective is the extraction of variability in the form of model fragments to achieve the adoption of software product line approaches; (ALPS) a Spanish national research project whose objective is the development of intelligent evolutive assistants for the initialization of software product lines; (DataME) a Spanish national research project whose objective is the definition of a model driven method for Big Data applications; (REVaMP2) an international ITEA 3 Call 2 project whose main objective is the creation of a holistic platform and process for variability extraction and management over time.

Moreover, the industrial partner, where this work was evaluated, was (CAF) a worldwide provider of railway solutions. The results of this dissertation contribute to the creation of a solution for managing the variability of the software existing in their railway systems.

1.5 Research Methodology

In order to perform the work of this dissertation, we have followed the design science approach of Wieringa, defined as the design and investigation of artifacts in context [12]. Therefore, we designed a ML-based approach (the artifact) for fragment retrieval on models (the context).

The research has been outlined as an engineering cycle and at the basis of this approach is the design cycle, consisting in three phases:

- **Problem Investigation:** the goal of this phase is to identify, describe, explain, and evaluate the problem to be treated. In problem investigation, the research goal is to investigate an improvement problem before an artifact is designed and when no requirements for an artifact have been identified yet [12]. Therefore, in this first phase, we defined the problem

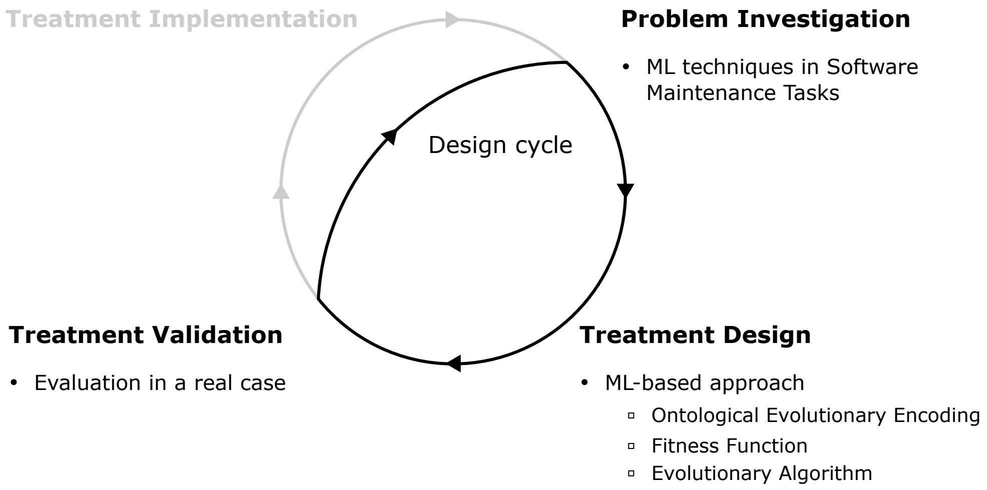


Figure 1.6: Design cycle of the research methodology followed in this dissertation

context (i.e. fragment retrieval on models) considering the background and the state of the art.

- **Treatment Design:** the goal of this phase is the design of one or more artifacts that could treat the problem [12]. Therefore, in this second phase, we provide the design of a ML-based approach, as the artifact to tackle the problem.

- **Treatment Validation:** the goal of this phase is to justify that the designed treatment would contribute to stakeholder goals when implemented in the problem context [12]. In this third phase, we validate the designed ML-based approach in order to provide evidence that our approach can benefit software engineers and modelers.

Following the design science approach, there is a last phase (Treatment implementation), whose goal is to transfer the treatment to the problem context [12]. However, as Wieringa highlights in [12], design science research projects do not perform the entire engineering cycle but are restricted to the design cycle. Therefore, the technological transfer associated to the real-world design and implementation is out of the scope of this PhD project.

Figure 1.6 shows the key points of this dissertation according to each phase of the design cycle and the objectives described in section 1.2. We start the research work reviewing the state of the art with regard to the ML techniques in software maintenance tasks. This helps us to understand the domain and have a deep knowledge of the problem context. When we verified that there is not a current solution for the problem, we continue the research work with the treatment design. It consists in the design of a ML-based approach for fragment retrieval on models. The main parts of this approach are an ontological evolutionary encoding to characterize model fragments, a fitness function to assess model fragments, and an evolutionary algorithm to extract model fragments from a model. Finally, we validate that the proposed approach is useful to solve the problem. To do this, we evaluate the proposed approach in a real case (CAF case) and compare these results against the results of the best approaches in the literature.

1.6 Dissertation Structure

This dissertation is structured into six parts:

Part I The first part is the introduction of the dissertation.

Chapter 1. Introduction This chapter introduces the motivation for the dissertation, the problem statement, the contribution, the overview of the work done, the methodology followed, and the structure of the dissertation.

Part II The second part of the dissertation focuses on the investigation problem.

Chapter 2. Background This chapter presents some background related to the topics covered in the dissertation. Specifically, it presents Model Driven Development, software maintenance tasks, Machine Learning, and the running example extracted from our industrial partner in order to illustrate the rest of the dissertation.

Chapter 3. State of the Art This chapter reviews the state of the art in relation to the ML techniques in software maintenance tasks.

Part III The third part of the dissertation focuses on the treatment design.

Chapter 4. Ontological Evolutionary Encoding This chapter presents the ontological evolutionary encoding to characterize model fragments.

Specifically, this encoding allows ML techniques to understand the model fragments, so that the ML techniques can be applied on model fragments.

Chapter 5. Fitness Function This chapter presents the fitness function to assess model fragments using ML techniques. Specifically, this function allows to determine what model fragment better realizes an specific query.

Chapter 6. Evolutionary Algorithm This chapter presents the evolutionary algorithm to extract model fragments from a model. Moreover, this chapter provides an overview of the Fragment Retrieval Approach based on Machine learning and Evolutionary algorithms (FRAME).

Part IV The fourth part of the dissertation focuses on the validation part.

Chapter 7. Evaluation Design This chapter presents the design of the evaluation performed to validate the proposed approach. Specifically, it presents the real case, the approaches under evaluation, the comparison and measure setup, and the threats to validity.

Chapter 8. Results of the Evaluation This chapter presents the reported results, the statistical analysis, the response to the research question of the evaluation, and the discussion of the results.

Part V The sixth part of the dissertation presents the conclusion.

Chapter 11. Conclusion and Future Work This chapter includes the conclusion, the recapitulation of the research questions presented and their answers, the next steps in the research, and the concluding remarks.

Part VI Finally, the seventh part of the dissertation includes the five papers selected for the dissertation.

PART II

Investigation Problem

You can't see the whole picture until you look at it from the outside.

One Piece

Chapter 2

Background

Contents

2.1 Overview of the Chapter

In this chapter, we introduce the background, which is conformed by the context of the two first objectives: (1) review the works related to the application of ML techniques for software maintenance tasks on models; (2) provide a ML-based approach for fragment retrieval in software maintenance tasks on models. Specifically, we present Model Driven Development, software maintenance tasks, Machine Learning, and the running example that will be used to illustrate the dissertation.

First, we present Model Driven Development (Section 2.2), which is a paradigm where the models are not merely artifacts of documentation. The goal of this paradigm is to automatically translate an abstract specification of the system into a fully functional software product.

Second, we present software maintenance tasks (Section 2.3), which are applied to maintain the systems. Specifically, we focus on Traceability Link Recovery, Bug Localization, and Feature Location. These tasks are oriented to retrieve software artifacts regarding a specific target in software systems.

Third, we present Machine Learning (Section 2.4), which is a discipline focused on empirically learning. The goal of the ML techniques is to build automated systems that can adapt and learn from their experience.

Finally, we present our running example extracted from our industrial partner and the main acronyms used in this dissertation (Section 2.5).

2.2 Model Driven Development

Modeling is an essential part of any engineering process [17]. With the emerging paradigm of Model Driven Development (MDD), the models has become to play a central role in the software development [18]. According to Mellor et al. [19], *"Model-driven development is simply the notion that we can construct a model of a system that we can then transform into the real thing"*. This paradigm is embraced by various organizations and companies, which have been proposing a wide variety of different techniques and several environments claiming to support MDD [17]. Among them, the Object Management Group (OMG) proposed a framework for software development, called Model Driven Architecture (MDA) [1] and the Eclipse community provides a unified

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

set of modeling frameworks, tooling, and standards implementations, under the Eclipse Modeling Project [2].

2.2.1 Models in MDD

A model is an abstraction of a system often used to replace the system under study [20, 21, 22]. Models have been used in software engineering to represent a partial and simplified view of a system. This allows engineers to better understand the system under development [23].

However, the arrival of the MDD is changing the way of using models in the development of software. In MDD, models are not only used to better understand the systems, but also to develop them. The models are used to synthesize concrete software development artifacts (e.g., other models, source code, configuration generation, and test scripts) through model transformations [24, 17]. As stated by Agrawal et al. [25]:

"The models are not merely artifacts of documentation, but living documents that are transformed into implementations. This view radically extends the current prevailing practice of using UML: UML is used for capturing some of the relevant aspects of the software, and some of the code (or its skeleton) is automatically generated, but the main bulk of the implementation is developed by hand. MDA, on the other hand, advocates the full application of models, in the entire life-cycle of the software product."

2.2.2 Domain Specific Language in MDD

According to the definition proposed by Van Deursen et al. [26], a Domain Specific Language (DSL) is:

"a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power, focused on, and usually restricted to, a particular problem domain."

DSLs are not a new topic. Going back to the last century, APT, a DSL for programming numerically controlled machine tools, was developed in 1957–1958 [27] and the well-known syntax specification formalism, dates back to 1959 [28, 29]. In fact, many DLS have been designed and used over the years and some of them are well-known and widely-used (e.g. LATEX, YACC, Make, SQL, and HTML).

[2]http://www.eclipse.org/modeling/

In MDD, DSLs play a cornerstone role for representing models and metamodels [30]. In general, a DSL is defined with regard to its abstract syntax and its concrete syntax. For the abstract syntax, a metamodel describes the concepts of the language, the relationships between them, and the structuring rules. On the other hand, the concrete syntax specifies the representations of the domain concepts in the metamodel. These representations are usually defined as a mapping between the metamodel and a textual or graphical notation [30].

2.3 Software Maintenance Tasks

Maintainability has become one of the most essential attributes of software quality, as software maintenance has shown to be one of the most costly and time-consuming tasks of software development [31]. In 1994, the software maintenance was estimated to account for 50% or more of the total development cost, and this maintenance cost showed no sign of declining [32]. In fact, seven years later, maintenance typically consumed about 40% to 80% (60% average) of software costs [33]. After six years, up to an 80% of a system's lifetime was spent on the maintenance and evolution of a system [5].

Traceability Link Recovery, Bug Localization, and Feature Location are amongst the most relevant tasks performed during software maintenance [7]. Below there is a brief description of each one:

Traceability Link Recovery: consists of tracing the software artifacts of a system (e.g. source code, requirements, and test cases) to significantly reduce the time cost that the engineers need to comprehend the system [34, 2]. The term *traceability* was coined by the requirements engineering community. In particular, in this community, the requirements traceability is defined as the ability to describe and follow the life of a requirement, in both a forward and backward direction [35]. In MDD, the requirements traceability is the ability to identify the elements that implements a specific requirement in a product model.

Bug Localization: consists of determining where to fix a bug based on bug report documents or incidence tickets [36]. For the traditional development, Bug Localization assists developers in locating culprit source code that must be modified to fix a bug [37]. In contrast, for MDD, Bug Localization assists developers in locating the elements that causes a particular error in a product model in order to fix it.

Feature Location: consists of finding the software artifacts that realize a feature [38]. A feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system [39, 40]. The presence or absence of a characteristic in a product, in that sense, entails the existence of different product configurations [7]. In MDD, the feature location is concerned with identifying the elements associated with a specific feature in a product model. It should be noted that the location of features is not only intended for software maintenance tasks, but also in reengineering tasks. In fact, Feature Location is a key activity in reengineering a set of product variants (e.g., variants created through clone-and-own to satisfy the needs of different customers) into an Software Product Line [41, 42].

2.4 Machine Learning

Machine Learning (ML) is known as the branch of artificial intelligence that gathers statistical, probabilistic, and optimization algorithms, which learn empirically. ML has a wide range of applications, including search engines, medical diagnosis, text and handwriting recognition, image screening, load forecasting, marketing and sales diagnosis, etc.

The ML techniques can be grouped in several categories, among the most known ones are:

Supervised Learning: groups the techniques whose goal is to build a concise model of the distribution of class labels in terms of predictor features. The resulting model is then used to assign class labels to the testing samples where the values of the predictor features are known, but the value of the class label is unknown [43].

Unsupervised Learning: groups the techniques whose goal is to discover patterns in large data sets or classifying the data into several categories without being trained explicitly [44]. The inputs of unsupervised learning techniques are not labelled.

This dissertation focuses on Supervised Learning techniques. However, to simplify the understanding or avoid misunderstandings, some of ML terms have been adapted:

Fitness value: is used instead of *class label.* The term *class label* is usually replaced with a more friendly term for the application domain (e.g. target or score). In our case, the term *class label* has been replaced taken into account the step, where the labels are predicted in the proposed approach.

Feature vector: is used instead of *predictor features*. Feature vector is not a new term for ML. The predictor features are variables that contain relevant information in predicting the output [45]. When these features are comprised in a vector, this is called feature vector in ML.

Classifier: is used instead of *model*. In ML, the *model* is the object built from the learning. In this dissertation, the term model could be used to mean the models in MDD or the model in ML. Since this would be confused, we have been adopted a synonym (i.e. classifier) that is also used in many ML works.

Characteristic: is used instead of feature. In this dissertation, the concept has two meanings. On the one hand, in feature location, a feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system [39, 40]. On the other hand, in ML, a feature is an individual measurable characteristic of the object being observed [46]. Therefore, to avoid misunderstandings, the term *feature* for ML has been replaced by the term characteristic.

2.5 Runtime Example

This section presents the railway domain and the Domain Specific Language used by our industrial partner to specify their product models. The language and graphical representations presented in this section will serve as the basis of the running example used to illustrate the rest of the dissertation.

2.5.1 Railway Domain

CAF is a worldwide leader in train manufacturing. CAF has produced a family of software systems to control the trains that they have been manufacturing over more than 25 years. Their trains can be found all over the world and in different forms (regular trains, subway, light rail, monorail, etc.).

A train unit is furnished with multiple pieces of equipment in its vehicles and cabins. These pieces of equipment are often designed and manufactured by different providers, and their aim is to carry out specific tasks for the train. Some examples of these devices are: the traction equipment, the compressors that feed the brakes, the pantograph that harvests power from the overhead wires, and the circuit breaker that isolates or connects the electrical circuits of the train. The control software is created with two goals in mind: (1) orchestrating the equipments to achieve flawless train functionality, and (2) guaranteeing the compliance of the train unit with the prevalent regulations of the country where the train unit is to be installed.

2.5.2 Train Control and Management Language

The Train Control and Management Language (TCML) is a DSL used to formalize the products manufactured by our industrial partner. TCML has enough expressiveness to describe both the interactions between the main pieces of equipment installed in a train unit and the non-functional aspects related to regulation (such as signal quality or installed redundancy levels). However, in order to gain legibility and due to intellectual property rights concerns, in this section we use a simplified subset of the TCML which only shows 6 meta-classes and 8 relationships (see the top of Figure 2.1). Specifically, this simplified subset of TCML focuses on four different kinds of equipment:

1 **High Voltage Equipment**, which is in charge of harvesting the energy that powers the different elements of the train.

2 **Contactors**, which are in charge of opening or closing the circuits between the High Voltage equipment and the Voltage Converters.

3 **Voltage Converters**, which are in charge of transforming the harvested electric power into a current that the Consumer Equipment can work with.

4 **Consumer Equipment**, which is in charge of carrying out all of the tasks required for the train to work properly and provide comfort to the passengers.

The bottom of Figure 2.1 depicts an example of a product model and a model fragment. The product model is taken from a real-world train and presents a converter assistance scenario. In the example, two separate pantographs (High Voltage Equipment) collect energy from the overhead wires and send it to their respective circuit breakers (Contactors), which in turn send it to a Voltage Converter. The converter then powers their assigned Consumer Equipments: the HVAC (the air conditioning system of the train) device and the PA (public address system).

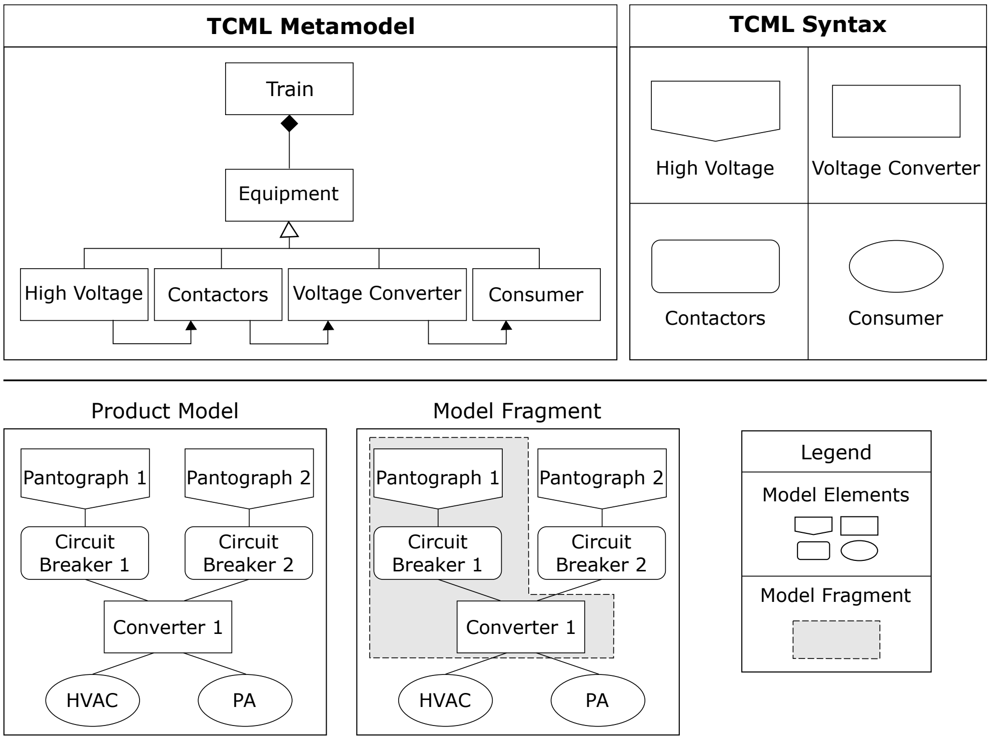


Figure 2.1: Example of a TCML model and model fragment

The model fragment is highlighted using a gray dotted shape. In this example, the model fragment is composed of three equipments: the *Pantograph 1*, the *Circuit Breaker 1*, the *Converter 1*; and these equipments are connected between them. In the following sections, we will use the TCML syntax, the product model, and the model fragment to present a running example through the rest of the dissertation.

The following video illustrates the CAF models: `youtube.com/watch?v=Ypcl2evEQB8`

TLAs You Need

OMG: The Object Management Group is an international, not-for-profit industrial consortium that creates and maintains software interoperability specifications.

MOF: Meta-Object Facility is the OMG metalanguage for defining modeling languages.

MDA: The Model-Driven Architecture is a set of OMG standards that enables the specification of models and their transformation into other models and complete systems.

MDD: Model Driven Development is an emerging paradigm for software construction that uses models to specify programs, and model transformations to synthesize executables.

DSL: A domain-specific language is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

ML: Machine Learning is known as the branch of artificial intelligence that gathers statistical, probabilistic, and optimization algorithms, which learn empirically.

LtoR: Learning to Rank is the name given to a family of ML techniques, which automatically address ranking tasks.

FNN: Feedforward Neural Network is a traditional neural network structure and lay the foundation for many other structures.

RNN: Recurrent Neural Network is a type of artificial deep learning neural network designed to process sequential data and recognize patterns in it.

IR: Information Retrieval is a sub-field of computer science that deals with the automated storage and retrieval of documents.

CAF: Construcciones y Auxiliar de Ferrocarriles is a worldwide provider of railway solutions. Their trains can be found all over the world and in different forms (regular trains, subway, light rail, monorail, etc.)

TLA: Three-letter acronym.

Chapter 3

STATE OF THE ART

Contents

3.1 Overview of the chapter

This chapter presents the state of the art for the dissertation. In order to find the recent related works, we conducted a preliminary systematic review of the literature. The goal of this systematic review is to provide an overview of ML techniques in software maintenance tasks on models. Starting from an initial set of 200 articles, we found that 56 of them actually adopted ML techniques to tackle Traceability Link Recovery, Bug Localization, or Feature Location.

Specifically, the systematic review (SR) was performed according to the guidelines proposed in [47, 48, 49], in order to answer the following two research questions:

SR-RQ1: What kind of software artifacts is the most common target for Traceability Link Recovery, Bug Localization, or Feature Location?

SR-RQ2: What are the most common ML techniques for Traceability Link Recovery, Bug Localization, Feature Location?

The following sections present the search process (Section 3.2), the related works selected by the search process (Section 3.3), and the answers for the research questions (Section 3.4).

3.2 Search Process

To collect all the available published literature relevant for the research questions, we adopted a database search. Therefore, we defined a search string as Kitchenham and Charters suggested in [47]. We used PICO (Population, Intervention, Comparison, and Outcomes) criteria to derive the major terms.

Table 3.1 shows the search terms regarding the PICO criteria. From this table, we used boolean operators to construct the search string. Specifically, all Population terms were combined by using the Boolean "OR" operator; all Intervention terms were combined by using the Boolean "OR"; and all the Outcomes terms were combined by using the Boolean "OR". Then, we combined the Population terms, the Intervention terms, and the Outcome terms by using the Boolean "AND" operator, which implies that an article only had to include one of the terms for Population, one of the terms for Intervention, and one of the terms for Outcomes.

Table 3.1: Terms for the search string regarding PICO criteria

	Terms
Population: In software engineering, population may refer to specific software engineering role, category of software engineer, an application area or an industry group.	Traceability Link Recovery, Bug Localization, Feature Location, Software Maintenance Task
Intervention: In software engineering, intervention refers to a software methodology, tool, technology, or procedure that addresses a specific issue.	Machine learning, decision tree, regression tree, classification tree, nearest neighbo*, neural net*, genetic algorithm, genetic program*, bayesian belief network, bayesian net*, association rule*, support vector machine, support vector regression, support vector*
Comparison: In software engineering, the comparison is the software engineering methodology, tool, technology, or procedure with which the intervention is being compared.	*Given the goal of the systematic review, the comparison is not applied.*
Outcomes: In software engineering, the outcomes should relate to factors of importance to practitioners.	Approach, method, tool, framework, process, guidelines

Given the search string, we followed the article selection process depicted in Fig. 3.1. Specifically, the selection process was composed of four steps:

1. The search string was used to collect the primary studies present in Scopus, until May 2020. Through this search, we found 135 articles respecting the search string.

2. The entire list of retrieved articles was filtered to exclude non-relevant articles. Specifically, The exclusion criteria were: (EC1) Studies not presented in English, (EC2) Non-computer science literature, and (EC3) Proceedings. These exclusion criteria were applied taking into account the title, abstract, and keywords of the articles. As result, 38 articles were discarded for satisfying the exclusion criteria. The remaining articles, 97 articles, were selected for the following step.

3. The 97 articles were filtered to include only the relevant articles to answer the research questions. The inclusion criteria were: (IC1) Studies

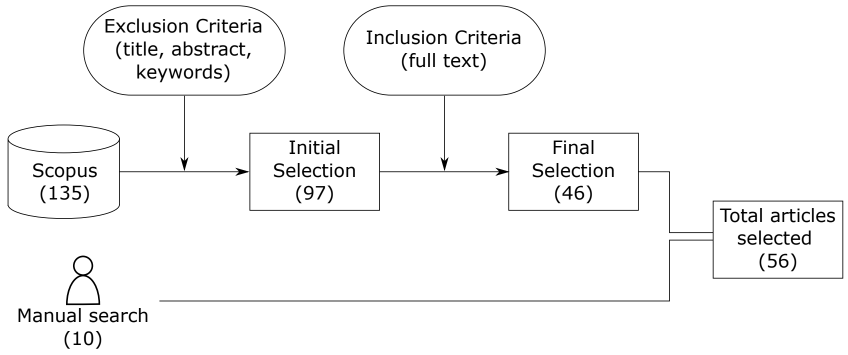

Exclusion Criteria (title, abstract, keywords)
Inclusion Criteria (full text)
Scopus (135)
Initial Selection (97)
Final Selection (46)
Total articles selected (56)
Manual search (10)

Figure 3.1: Overview of the search process

for Traceability Link Recovery through ML techniques, (IC2) Studies for Feature Location through ML techniques, and (IC3) Studies for Bug Localization through ML techniques. These inclusion criteria were applied taking into account the full-text of the articles. As result, 51 articles were discarded for not satisfying the inclusion criteria. The remaining, 46 articles, were selected for this chapter, which is about 34% of the papers found in Scopus. Moreover, it is noteworthy that among these 46 articles are two of the articles for this dissertation [9] and [14].

4. Finally, we complete the search through a manual search, in order to search for possible missing papers. In particular, 10 articles were included through the manual search.

Therefore, the 56 articles, that were included through the Scopus search and the manual search, were considered the related works for this dissertation.

3.3 Related Works

This section provides an overview of the related works, that were found through the search. Specifically, these related works focus on applying ML techniques for Traceability Link Recovery, Bug Localization, or Feature Location. Table 3.2 shows the related works (56 articles) grouped by these tasks. In the table, our articles are highlighted using bold font.

According to the number of articles for each task, ML techniques are more commonly used for Bug Localization than for Traceability Link Recovery or for Feature Location. While 47% of the articles apply ML techniques for Bug

Table 3.2: Articles grouped by Software Maintenance Task

Traceability Link Recovery	Bug Localization	Feature Location
[9], **[14]**, [50], [51], [52], [53], [54], [55], [56], [57], [58], [59]	[60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85]	[10], [86], [87], [88], [89], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99], [100], [101], [102]

Localization, Feature Location and Traceability Link Recovery are only tackled by 32% and 21% of the articles, respectively.

Moreover, Figure 3.2 shows the evolution of the articles taking into account the publication dates. We can note a renewed interest for ML application in software maintenance tasks. Specifically, the number of articles that apply ML techniques in the last four years (between 2016 and 2019) has been doubled, and even, tripled regarding previous years.

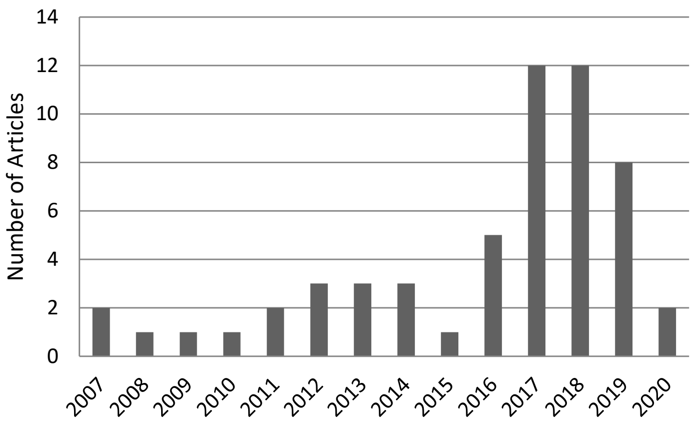


Figure 3.2: Number of articles by year

3.3.1 Related works regarding the kind of software artifacts

This section analyzes the works, that were found through the search, according to the kind of software artifact. Figure 3.3 shows two plots that graphically summarize the results. These plots show that most research articles target source code, neglecting other software artifacts such as models and product descriptions. In fact, on the left of Figure 3.3, the pie plot shows that only 12% of the articles target models, while 84% of the articles focus on source code.

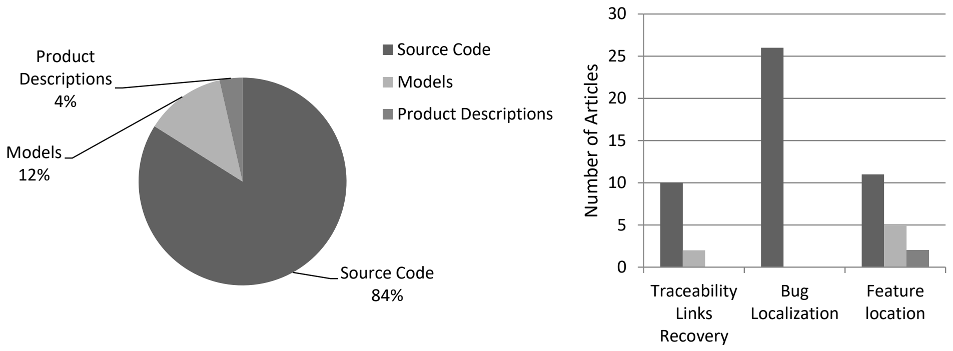


Figure 3.3: Number of articles by software maintenance task according to their main software artifact

Moreover, on the right part of the figure, the bar plot shows the number of articles for each software maintenance task. According to this plot, the number of articles that target source code is higher to the number of articles that target models and product descriptions for each one of the software maintenance task. In fact, regarding Bug Localization, the search performed did not find any article that applies ML techniques on models or on product descriptions.

Table 3.3 shows the related works grouped by the software maintenance task and the software artifact. 47 works focus on source code, 7 works focus on models, and 2 works focus on product descriptions. In the table, our articles are highlighted using bold font, to highlight the main difference between our work and most related works. While most works focus on source code as the main software artifact, our research work, and specifically this dissertation, focuses on models.

In fact, without our articles, there are only five articles that target models instead of source code [90, 91, 98, 99, 10]. All these articles tackle the same software maintenance task: feature location. In contrast, our work is not

Table 3.3: Articles grouped by software maintenance task and software artifact

	Traceability Link Recovery	Bug Localization	Feature Location
Source Code	[50], [51], [52], [53], [54], [55], [56], [57], [58], [59]	[60], [61], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82], [83], [84], [85]	[86], [87], [88], [89], [92], [93], [94], [95], [96], [97], [101]
Models	**[9]**, **[14]**		[90], [91], [98], [99], [10]
Product Descriptions			[100], [102]

focused on solving an specific maintenance task, but solving a problem that is common to the three software maintenance tasks: fragment retrieval.

3.3.2 Related works regarding the ML techniques

This subsection analyzes the works, that were found through the search, according to the ML techniques applied. Figure 3.4 shows two plots that graphically summarize the results. These plots show that most research articles apply Neural Networks or Genetic Algorithms. In fact, on the top of Figure 3.4, the pie plot shows that about 21% of the articles apply genetic algorithms and 25% of the articles apply neural networks.

Moreover, on the bottom of the figure, the bar plot shows the number of articles for each maintenance task. According to this plot, most of the articles that apply genetic algorithms tackle Bug Localization and most of the articles that apply neural networks tackle Feature Location. In contrast, there is an uniform range of ML techniques that are applied for Traceability Link Recovery.

Table 3.4 shows the related works grouped by the software maintenance task and the ML technique. In this table, our articles are highlighted using bold font. Thanks to this table, we can see that genetic algorithms are not only used for our works [9, 14], but they also are used for many other research works [10, 54, 55, 87, 89, 90, 91, 94, 99, 62, 67, 79].

All these works apply genetic algorithms guided by different type of fitness functions. Several works rely on genetic algorithms guided by Latent Semantic Analysis [10, 89, 91, 99]. Other works rely on genetic algorithms guided by Latent Dirichlet Allocation [54, 89]. The rest of the works rely on genetic algorithms guided by fitness functions, which have been specifically defined for

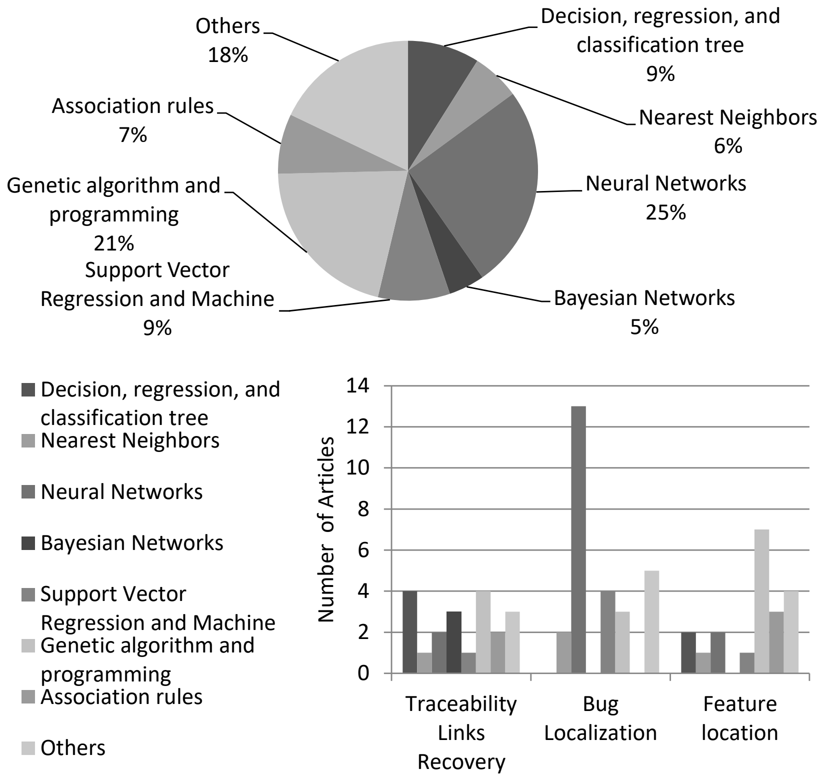


Figure 3.4: Number of articles by software maintenance task according to the ML technique applied

their works. In [55], the fitness function uses semantic similarity. In [87, 94], the fitness function uses a functional cohesion measure. In [90], the fitness function uses placement signatures. In [62], the fitness function uses the lexical and historical similarity. In [67], the fitness function uses a mathematical formula designed to calculate the relative performance. Finally, in [79], the fitness function uses a formula based on the positive and negative test cases.

In contrast, our work applies a genetic algorithm guided by a different fitness function. Specifically, the fitness function uses a Learning to Rank algorithm, which belongs to the ML techniques oriented to rank to objects.

In fact, among the retrieved articles, there are only three articles that use Learning to Rank algorithms [74, 84, 85]. These three articles apply Learning to Rank algorithms for Bug Localization on source code. None of them uses

Table 3.4: Articles grouped by software maintenance task and ML technique

	Traceability Link Recovery	Bug Localization	Feature Location
Decision, regression, and classification tree	[50], [51], [52], [53]		[95], [98]
Nearest Neighbors	[51]	[70], [80]	[95]
Neural Networks	[59], [58]	[60], [61], [64], [65], [66], [68], [69], [71], [72], [73], [75], [76], [78]	[86], [93]
Bayesian Networks	[51], [52], [53]		
Support Vector Regression and Machine	[51]	[61], [63], [81], [83]	[95]
Genetic algorithm and programming	**[9]**, **[14]**, [54], [55]	[62], [67], [79]	[10], [87], [89], [90], [91], [94], [99]
Association rules	[53],[56]		[88], [92], [96]
Others	**[9]**, **[14]**, [57]	[74], [77], [82], [84], [85]	[97], [100], [101], [102]

the Learning to Rank algorithms to guide a genetic algorithm and none of them target models as the main software artifact, as our work does.

3.4 Research Questions for the Systematic Review

This section provides answers for the two research questions, whose final goal is to provide an overview of ML techniques for software maintenance tasks.

In response to SR-RQ1, source code is the most common target for Traceability Link Recovery, Bug Localization, Feature Location. In contrast, our work targets models for fragment retrieval in Traceability Link Recovery, Bug Localization, or Feature Location.

In response to SR-RQ2, there is no a clear ML technique for Traceability Link Recovery. Neural Networks are the most common ML techniques for Bug Localization and Genetic algorithms are the most common ML techniques for Feature Location. Furthermore, in our case, we propose the use of a genetic algorithm guided by a Learning to Rank algorithm for fragment retrieval in Traceability Link Recovery [9, 14], Bug Localization, or Feature Location [13].

PART III

Treatment Design

The way I see it, if you're gonna to build a time machine into a car, why not do it with some style?
Back to the Future

Chapter 4

ONTOLOGICAL EVOLUTIONARY ENCODING

Contents

4.1 Overview of the chapter

The second objective (OB2) of this dissertation is to provide a ML-based approach to automatically perform fragment retrieval in software maintenance tasks on models, called FRAME. Specifically, the approach has to assess different model fragments of a model and identify which model fragment better realizes a specific query. To assess the model fragments, the approach is based on ML techniques. However, ML techniques cannot understand model fragments without being characterized.

Most of the ML techniques are designed to process feature vectors as inputs [103]. Feature vectors are known as the ordered enumeration of characteristics that describe the object being observed [46]. Therefore, to apply ML techniques on model fragments, the first challenge consists in identifying the characteristics from model fragments and selecting the most suitable ones to encode the model fragments in feature vectors. To do this, this chapter presents an ontological evolutionary encoding.

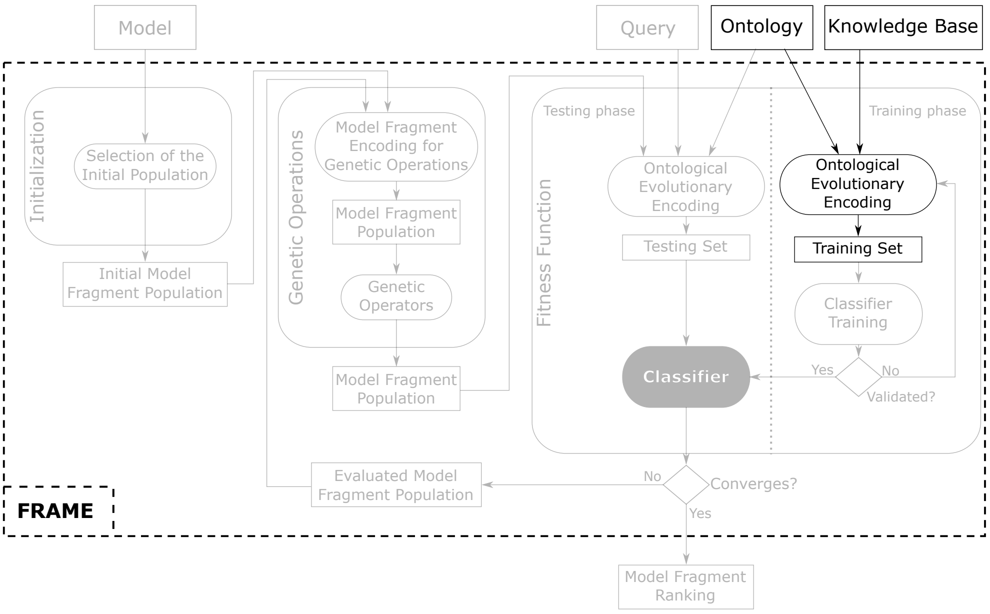

Model
Query
Ontology
Knowledge Base
Initialization
Selection of the Initial Population
Initial Model Fragment Population
Genetic Operations
Model Fragment Encoding for Genetic Operations
Model Fragment Population
Genetic Operators
Model Fragment Population
Fitness Function
Testing phase
Training phase
Ontological Evolutionary Encoding
Testing Set
Ontological Evolutionary Encoding
Training Set
Classifier Training
Classifier
Yes
No
Validated?
Evaluated Model Fragment Population
No
Converges?
Yes
FRAME
Model Fragment Ranking

Figure 4.1: Overview of the ontological evolutionary encoding in the FRAME approach

Figure 4.1 shows an overview of the FRAME approach. This figure highlights in black colour the steps and the artifacts of the approach that are described in this chapter. According to this figure, this chapter describes the ontological evolutionary encoding, which is an step of the approach. Moreover, this chapter also describes the inputs (ontology and knowledge base) and outputs (training set) for that step.

The following sections present the input and output artifacts (Section 4.2), the three stages of this encoding (Section 4.3), and a summary of the ontological evolutionary encoding (Section 4.4).

4.2 Input and output artifacts

The input of the ontological evolutionary encoding consists of a domain ontology and a knowledge base (see Figure 4.1).

The **ontology** represents the main concepts and relations of a specific domain. Figure 4.2 shows an example of the ontology according to the running example. This ontology contains the main concepts for the railway domain (e.g. Pantograph or Converter) and the main relations between the concepts (e.g. *R1* is a relation between a Pantograph and a Circuit Breaker).

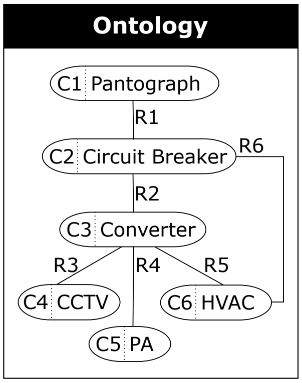


Figure 4.2: Example of a domain ontology

The **knowledge base** consists in a set of samples, whose content depends of the object being observed. If we want to predict the tomorrow's weather, we will need the analyses of the weather in the previous days. Therefore, the knowledge base would contain these analyses. Similarly, if we want to assess a model fragment according to a query, we will need other model fragments that have been previously assessed. Therefore, the knowledge base contains the model fragments that have been manually retrieved by the engineers and modellers for years.

Specifically, each sample in the knowledge base contains a query, a model fragment, and a fitness value. The query describes using natural language the requirement, bug, or feature, that was searched by the engineer in a product model. The model fragment consists of the element or the set of elements that the engineer manually retrieved from the model taking into account the description in the query. The fitness value determines how well the model fragment realizes the query. The better the model fragment realizes the query, the greater the fitness value.

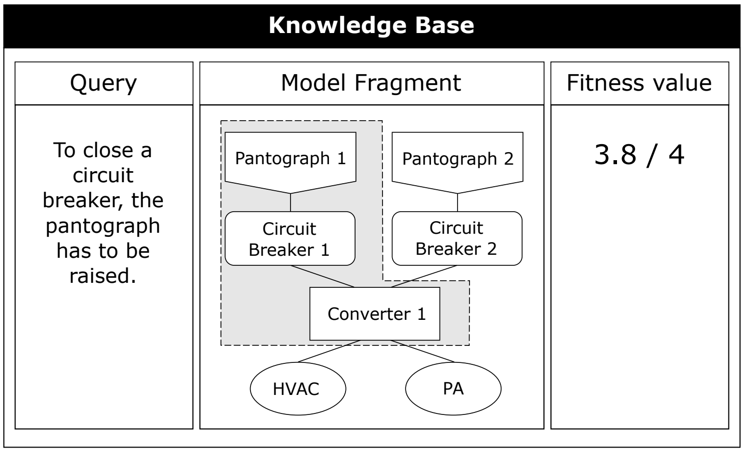
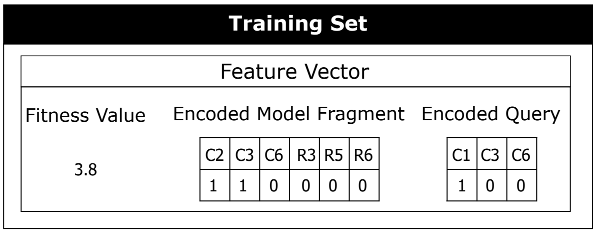


Figure 4.3: Example of a sample in the knowledge base

Figure 4.3 shows an example of a sample in the knowledge base. In this sample, the query describes a requirement, the model fragment contains five elements (i.e. a pantograph, a circuit breaker, a converter, and two connections), and the fitness value is high but it is not the maximum value possible. This fitness value can indicate that the model fragment realizes well the query, but the model fragment is not perfect or complete. The query could need model elements that have not been considered in the model fragment or some elements in the model fragment are not really necessary for the query.

Finally, the ontological evolutionary encoding has one output artifact: the training set. The **training set** contains feature vectors, where each feature vector encodes a sample of the knowledge base. Specifically, each feature vector contains a target value (the fitness value of the sample) and a set of the characteristic/value pairs that characterizes the objects being observed (model fragments regarding queries).

Figure 4.4 shows an example of a feature vector in the training set. The first value in the feature vector is the target value, which corresponds to the fitness value in a sample of the knowledge base. Then, the feature vector contains a set of characteristic/value pairs that characterize the model fragment in the sample of the knowledge base. In figure 4.4, the model fragment is encoded using six different characteristics: *C2*, *C3*, *C6*, *R3*, *R5*, and *R6*. The numerical values of these characteristics correspond to the model fragment characteri-



C2	C3	C6	R3	R5	R6
1	1	0	0	0	0

C1	C3	C6
1	0	0

Figure 4.4: Example of a feature vector in the training set

zation. Finally, the feature vector contains a set of characteristic/value pairs that characterize the query in the sample of the knowledge base. In figure 4.4, the query is encoded using three characteristics (i.e. *C1*, *C3*, *C6*) and their numerical values correspond to the query characterization.

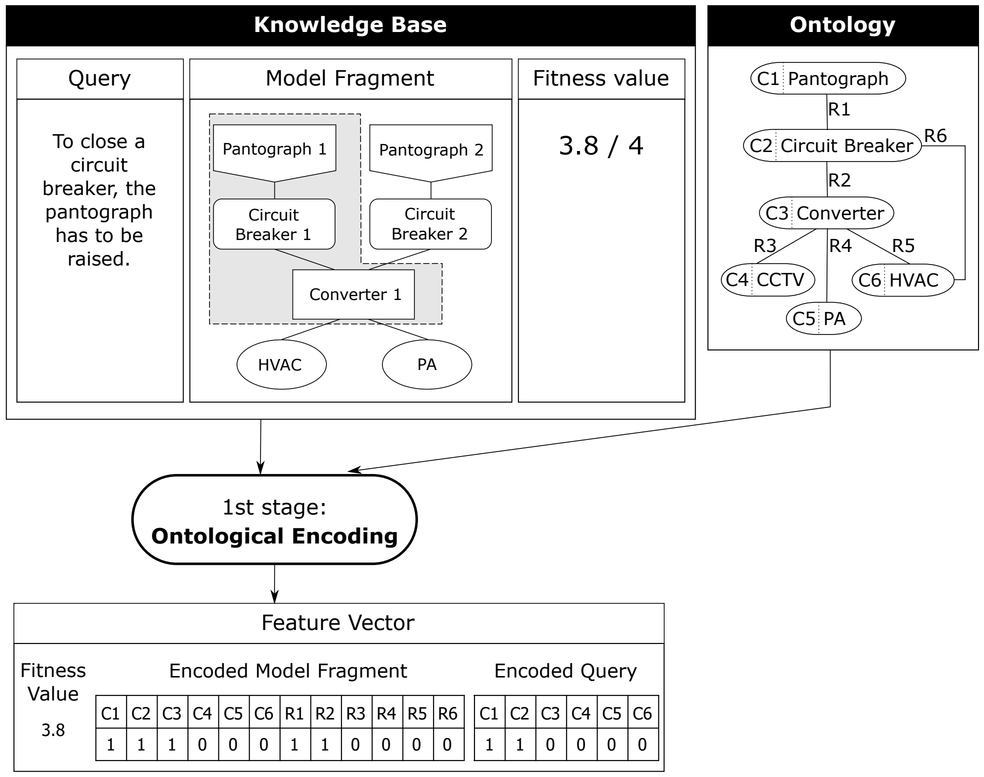
4.3 Stages of the ontological evolutionary encoding

The ontological evolutionary encoding consists of three stages: ontological encoding, evolutionary encoding, and feature selection.

4.3.1 Ontological Encoding

In this first stage, the samples of the knowledge base are turned into feature vectors based on a domain ontology. For each sample in the knowledge base, the fitness value is assigned as the target value of a feature vector. Then, the model fragment is encoded based on the ontology. We consider each concept and relation in the ontology as a characteristic in the feature vector. The value of each characteristic is computed as the frequency of the concept or the relation in the model fragment. Similarly, the query is encoded as part of the feature vector taking into account the ontology. Specifically, each concept in the ontology is represented as a characteristic in the feature vector and the value of each characteristic is computed as the frequency of the concept in the query. Therefore, the output of this stage is a set of feature vectors, where each feature vector represents a sample of the knowledge base according to the concepts and the relations of the ontology.

Moreover, since model fragments and queries are based on natural language, the terms used in the ontology do not always align well with the terms in the model fragments and with the terms in the queries. For this reason, Natural Language Processing (NLP) techniques are used to process both the model fragments and the queries before applying the encoding. Specifically, the model fragments and the requirements are processed by a combination of NLP techniques defined in [104], which consists of tokenizing, lowercasing, removal of duplicate keywords, syntactical analysis, lemmatization, and stop-word removal.



Fitness Value	C1	C2	C3	C4	C5	C6	R1	R2	R3	R4	R5	R6	C1	C2	C3	C4	C5	C6
3.8	1	1	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0

Figure 4.5: Example of the inputs and output for the ontological encoding stage

Figure 4.5 shows an example of how the ontological encoding stage encodes a sample of the knowledge base based on the ontology. The fitness value in the feature vector is the same as in the sample of the knowledge base. Then, the concepts and relations of the ontology are the characteristics to encode the model fragment. For example, the concept *Pantograph* is mapped as *C1*, and the relation between the concepts *Converter* and *HVAC* is mapped as *R5*. On the one hand, these concepts and relations are compared with the model fragment, so that the number of occurrences of the concept or relation in the model fragment is the value of the correspondent characteristic in the feature vector.

Therefore, the value of the characteristic *C1* is 1 because there is one pantograph in the model fragment, and the value of the characteristic *R5* is 0 because there is no relation of the type *Converter-HVAC* in the model fragment. In fact, the product model contains a relation of the type *Converter-HVAC*, but the model fragments does not contain this relation. Therefore, the value for *R5* is 0 in the feature vector, because each feature vector only contains the encoding of a model fragment, not the encoding of the whole model. Finally, the concepts in the ontology are also the characteristics to encode the query. The concepts are compared with the query, so that the number of occurrences of the concept in the query is the value of the correspondent characteristic in the feature vector. Therefore, the value of the *C1* is 1 because the concept pantograph appears once in the query.

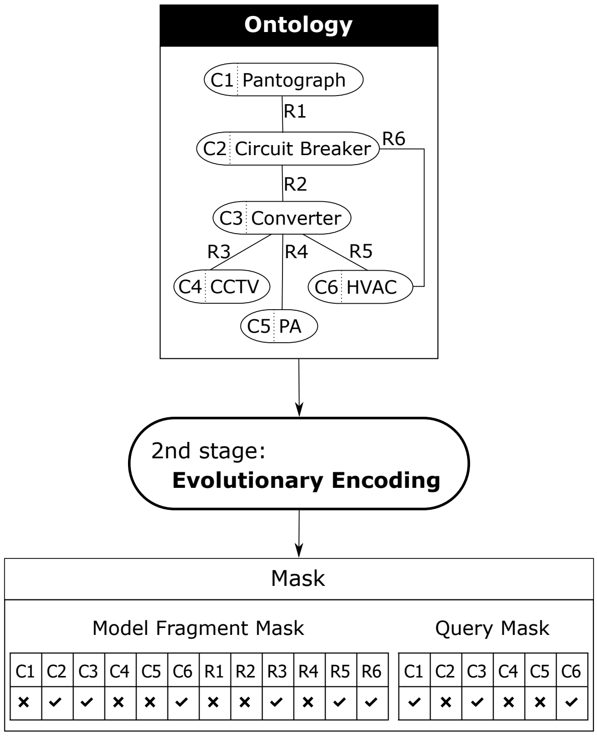
It is important to notice that all the feature vectors will have the same length. No matters if a model fragment has five elements and another model fragment has 50 elements, their feature vectors will have the same number of characteristics. Similarly, the number of characteristics is the same for all the queries, no matter how long the queries are. This is fundamental for most of the ML techniques, such as the Learning to Rank algorithms.

4.3.2 Evolutionary Encoding

In ML, feature selection is the name of the process used to reduce the number of characteristics in feature vectors. This process selects only the most relevant characteristics in the feature vectors, which reduces the time cost and the redundant information [105, 106]. To take advantage of these benefits, the evolutionary encoding stage analyzes different combinations of characteristics in order to identify the most relevant characteristics and discard the others. Specifically, an evolutionary algorithm generates and evolves different masks, where each mask indicates what characteristics are enabled or disabled.

Figure 4.6 shows an example of the evolutionary encoding stage. Taking into account the mask shown in this figure, only six of the twelve characteristics for the model fragments are enabled: *C2*, *C3*, *C6*, *R3*, *R5*, and *R6*. The rest of the characteristics for model fragments are disabled. Similarly, the mask indicates that only three of the characteristics for the queries (i.e. *C1*, *C3*, and *C6*) are enabled.

Therefore, the evolutionary algorithm generates and evolves different masks. The masks are tested over the knowledge base and the one that achieves the best results is the output of the evolutionary encoding stage.



C1	C2	C3	C4	C5	C6	R1	R2	R3	R4	R5	R6
✗	✓	✓	✗	✗	✓	✗	✗	✓	✗	✓	✓

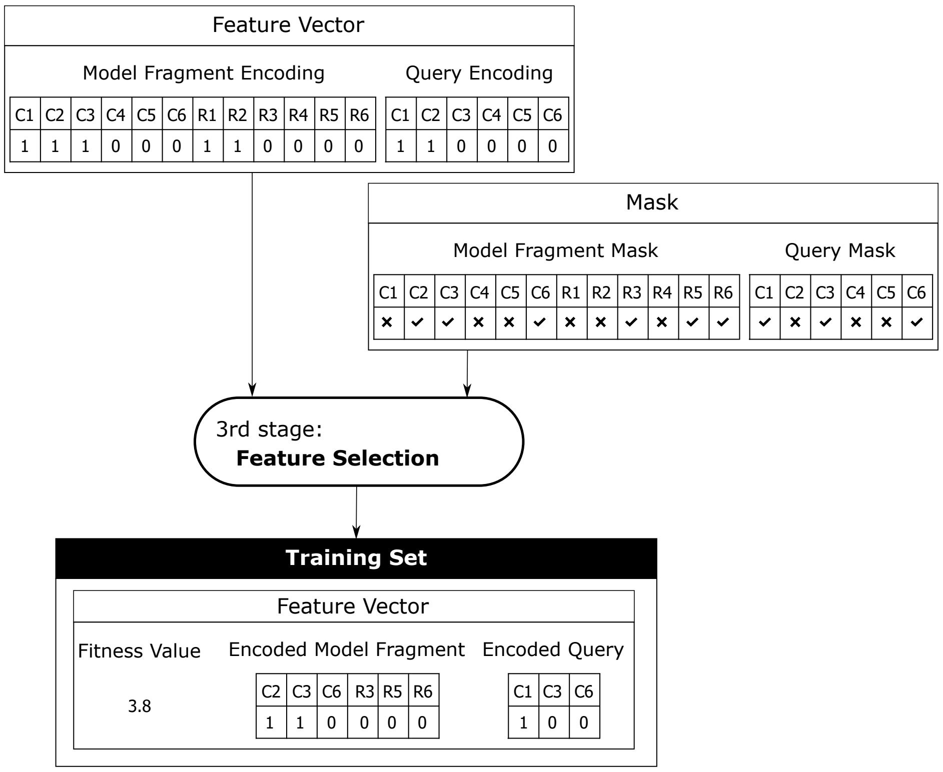
C1	C2	C3	C4	C5	C6
✓	✗	✓	✗	✗	✓

Figure 4.6: Example of the input and output for the evolutionary encoding stage

4.3.3 Feature Selection

In the third stage, the objective is to merge the outputs of the two previous stages. The ontological encoding stage returns a set of feature vectors and the evolutionary encoding stage returns a mask. Therefore, this stage apply the mask on the feature vectors to reduce their number of characteristics. As the Figure 4.7 shows, each disabled characteristic in the mask is discarded in the feature vectors. Therefore, the feature vector are only composed by the characteristics that are enabled in the mask. In the mask of the Figure 4.7, the enabled characteristics are *C2*, *C3*, *C6*, *R3*, *R5*, and *R6* for the model fragments and *C1*, *C3*, and *C6* for the queries. Therefore, the feature vector in the example is simplified using only these characteristics.

The output of the feature selection stage is the training set. The training set contains all the simplified feature vectors; one simplified feature vector for each feature vector in the input of the stage.



C1	C2	C3	C4	C5	C6	R1	R2	R3	R4	R5	R6
1	1	1	0	0	0	1	1	0	0	0	0

C1	C2	C3	C4	C5	C6
1	1	0	0	0	0

C1	C2	C3	C4	C5	C6	R1	R2	R3	R4	R5	R6
✗	✓	✓	✗	✗	✓	✗	✗	✓	✗	✓	✓

C1	C2	C3	C4	C5	C6
✓	✗	✓	✗	✗	✓

3.8

C2	C3	C6	R3	R5	R6
1	1	0	0	0	0

C1	C3	C6
1	0	0

Figure 4.7: Example of feature selection based on a mask

4.4 Summary of the ontological evolutionary encoding

This section provides a summary of the ontological evolutionary encoding, which is described in the previous sections (see Section 4.2 and Section 4.3). The objective of the ontological evolutionary encoding is to characterize the object being observed (i.e. model fragments regarding queries) to be able to apply ML techniques for fragment retrieval on models. To do this, the ontological evolutionary encoding consists of three stages. The first one, ontological encoding, turns the samples of a knowledge base into feature vectors, where each sample contains a model fragment regarding a query. The second one, evolutionary encoding, identifies what characteristics have to be enabled in order to reduce the time cost and the redundant information. The third one, feature selection, applies the mask on the feature vectors to reduce their number of characteristics.

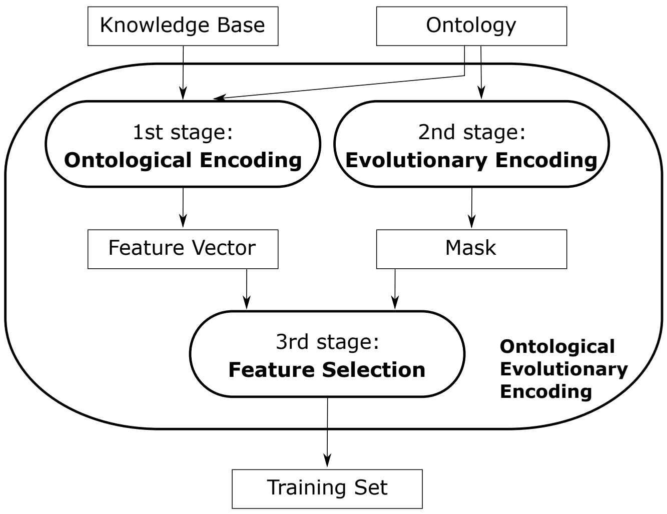


Figure 4.8: Overview of the ontological evolutionary encoding approach.

As output, the ontological evolutionary encoding returns a training set, which contains a set of feature vectors. Each feature vector corresponds to a sample of the knowledge base that has been encoded based on the ontology and simplified using a mask. Figure 4.8 shows an overview of the ontological evolutionary encoding, including examples for the inputs and outputs of its stages.

Chapter 5

FITNESS FUNCTION

Contents

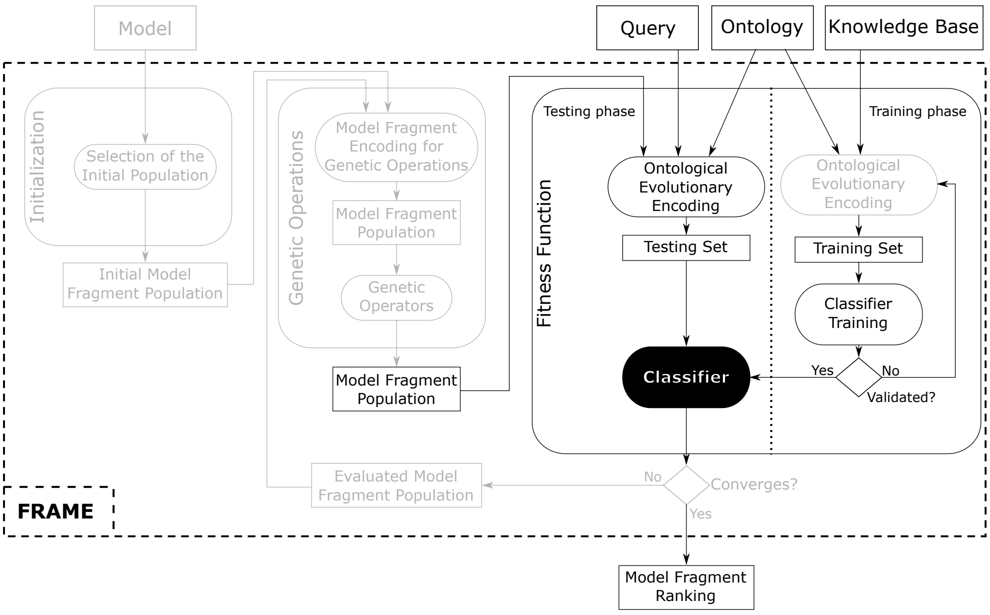
5.1 Overview of the chapter

To determine what model fragment is the best realization of a specific query, the FRAME approach needs to assess the different model fragments from a product model. To do this, this chapter presents a fitness function based on ML techniques. ML empowers us to take advantage of the knowledge and the experience that have been generated in companies for years in order to automatically assess the model fragments. Specifically, using the model fragments that have been manually retrieved in the last years, the ML technique can learn how to automatically assess new model fragments.

Learning to Rank (LtoR) is the name given to a family of ML techniques, which automatically address ranking tasks. Specifically, the LtoR algorithms make possible the construction of a classifier that contains a set of rules to rank objects. The classifier automatically learns these rules by comparing the objects within a knowledge base. Then, since the classifier knows how to rank objects following these learned rules, the classifier can be used to rank new objects. In other words, LtoR algorithms use a knowledge base to train a classifier, which is called training phase. Then, the classifier is used to rank new objects, which is called testing phase [107].

In the fitness function of the FRAME approach, a LtoR algorithm is used to train a classifier. The classifier is trained using the retrieved model fragments in the knowledge base. Therefore, the classifier can learn when a model fragment is a bad or a good realization of a query and what fitness value has to be assigned in each case. Once the classifier has been trained, the classifier is used to assess a model fragment population, which is a set of model fragments from a model. According to its rules, the classifier assesses the model fragments assigning a fitness value to each model fragment. In this case, the learned rules are oriented to assign a higher fitness value to the model fragments that better realize the query. Then, using these fitness values, the model fragments are ranked and the top model fragment is the best realization of the query according to the classifier assessing.

Figure 5.1 shows an overview of the fitness function in the FRAME approach. This figure highlights in black colour the steps and the artifacts of the approach that are described in this chapter. According to this figure, the fitness function has two phases: training phase and testing phase. The training phase is responsible of train the classifier, and the testing phase is responsible of assessing the model fragment population through the classifier. As output, the fitness function returns a model fragment ranking, which contains the model fragments ordered by their fitness values.



Model
Query
Ontology
Knowledge Base
Initialization
Selection of the Initial Population
Initial Model Fragment Population
Genetic Operations
Model Fragment Encoding for Genetic Operations
Model Fragment Population
Genetic Operators
Model Fragment Population
Fitness Function
Testing phase
Training phase
Ontological Evolutionary Encoding
Testing Set
Classifier
Ontological Evolutionary Encoding
Training Set
Classifier Training
Yes
No
Validated?
Evaluated Model Fragment Population
No
Converges?
Yes
FRAME
Model Fragment Ranking

Figure 5.1: Overview of the process to evaluate and rank model fragments

While the testing phase is repeated for each new model fragment population to be assessed, the training phase only is performed the first time. The classifier is trained once, and then, this classifier is used for each new model fragment population. In fact, the classifier is an artifact created in the training phase and it is a step in the testing phase to rank new model fragment populations. For this reason, Figure 5.1 shows the classifier in a black, rounded rectangle to point out its double meaning. The following sections present the training phase (Section 5.2) and the testing phase (Section 5.3) in more detail. Moreover, the last section presents a summary of the fitness function (Section 5.4).

5.2 Training phase

To manually retrieve the model fragment that realizes a query, engineers or modellers have to search in a model all the model elements that are related to the query. To do this, engineers have to decide what model elements are relevant for the query according to some criteria. These criteria are based on the experience and the knowledge acquired by the engineers in previous searches. For example, in a railway domain, an engineer can note that if the retrieved model fragment has a circuit breaker, the associated converter has to be also included in the retrieved model fragment. Therefore, when a query involves a circuit breaker, the engineer always considers the circuit breakers and the associated converters.

Similarly, a LtoR algorithm can compare the retrieved model fragments and related queries known beforehand, in order to imitate the experience and knowledge of domain experts. Specifically, the comparisons allow to learn the criteria or rules defined in the classifier. Therefore, the target of the training phase is to generate the classifier, which in the testing phase will be used to assess new model fragment populations. The right part highlighted in the Figure 5.1 shows the artifacts and steps of the training phase, which are described in more details in the following subsections.

Note that the LtoR algorithm is used not only to automate the assessing of model fragments, but also to find criteria or rules that can be too complex from a human perspective.

5.2.1 Input and output artifacts

The input artifacts of the training phase are the knowledge base and the ontology.

The **knowledge base** contains the model fragments that have been manually retrieved by the engineers and modellers for years. Specifically, the knowledge base consists in a set of samples, where each sample contains a query, a model fragment, and a fitness value. The query uses natural language to describe the requirement, bug, or feature, that was searched by the engineer in a product model. The model fragment consists of the element or the set of elements that the engineer manually retrieved from the model taking into account the description in the query. The fitness value determines how well the model fragment realizes the query. There are more details about the knowledge base and an example in Section 4.2.

The **ontology** contains the main concepts and relations of a domain. Therefore, the ontology contains all the necessary information to encode the knowledge base into feature vectors, following the encoding described in Chapter 4. There are more details about the ontology and an example in Section 4.2.

The other artifacts in the training phase are the training set and the classifier.

The **training set** contains feature vectors, where each feature vector encodes a sample of the knowledge base. Specifically, each feature vector contains the fitness value of the sample and a set of characteristic/value pairs that characterizes the model fragment and the query of the sample. There are more details about the training set and an example in Section 4.2.

The **classifier** is the output of the training phase. A classifier is a rule-set that is learnt from a given training set [108, 109]. The final purpose of this classifier is the assessing of model fragments in the testing phase.

5.2.2 Steps of training phase

The training phase has two steps: ontological evolutionary encoding and classifier training. The first one is applied to turn the knowledge base into a training set. The second one is applied to train a classifier from the training set.

Ontological Evolutionary Encoding

Since most of the ML techniques (e.g. LtoR algorithms) are designed to process feature vectors as inputs [103], the knowledge base has to be encoded. To do this, we propose the ontological evolutionary encoding, which is described in Chapter 4. Specifically, thanks to this encoding, each sample of the knowledge base is encoded into a feature vector. The resulting set of feature vectors is commonly called training set, because this set contains the feature vectors used to train the classifier.

Classifier Training

Once the training set is available, the classifier is trained through a LtoR algorithm. However, the classifier training step involves not only the training, but also the tuning and validation of the classifier. To do this, 80% of the feature vectors in the training set are used to train the classifier, and the rest of the feature vectors (i.e. 20%) are used as a validation set for both the tuning and the validation of the classifier.

The most common tuning methods are grid search and manual tuning [110, 111]. Specifically, the FRAME approach is designed to use a grid search as it is described in [112]. Firstly, a grid search is built to determine the values of the parameters. The parameters depend on the LtoR algorithm used. For example, Rankboost algorithm has three tuning parameters: number of iterations, threshold, and metric. Secondly, it is necessary to select the range of each parameter. For example, the considered values for the number of iterations could be in range [100,500]. Thirdly, the parameters are uniformly sampled in their range. For example, if we take into account only hundreds in range [100,500], the sampled values for the number of iterations will be 100, 200, 300, 400, and 500. Fourthly, all of the combinations of the sampled values for each parameter are used to train and test the classifier. The combination that obtains the best results is used to tune the LtoR algorithm.

Then, the classifier is validated through a cross-validation method. Cross-validation is a statistical method of evaluating and comparing ML techniques by dividing data into two segments: one used to train a classifier, and the other used to validate the classifier [113]. Moreover, to reduce variability, multiple rounds of cross-validation are performed using different partitions, and the results are averaged over the rounds [114]. Among the cross-validation methods, the most popular is k-fold. Specifically, this method consists of randomly dividing the knowledge base into k-independent partitions. Then, $k-1$ of the partitions are used to train the classifier, and this classifier is then used to test the partition that is left out. This procedure is repeated k times, each time leaving out another partition. This produces k estimations of the classifier, allowing assessment of its central tendency and variance [115].

If the central tendency and variance indicate that the classifier cannot properly rank model fragments, it will be necessary to train again the classifier. In this new training, some artifacts of the training phase (e.g. the ontology, the knowledge base, or the ML technique) must be modified in order to improve the classifier. Otherwise, if the central tendency and variance indicate that the classifier can properly rank model fragments, the classifier obtains the go-ahead. Therefore, the classifier is available for the testing phase.

Figure 5.2 shows an example of the application of k-fold method. This method is applied on a knowledge base with twelve samples, using a k equals to four. Therefore, the samples of the knowledge base are randomly divided in four folds, where each fold contains three of the samples. Then, we have to perform four iterations, the first iteration trains a classifier using the *Fold 1*, the *Fold 2*, and *Fold 3*; and the *Fold 4* is used to test the classifier obtaining the first performance results. In the second iteration, the testing is performed using the *Fold 3* and the rest are used to train the classifier. In the third iteration, the testing is performed using the *Fold 2*, and the rest are used to

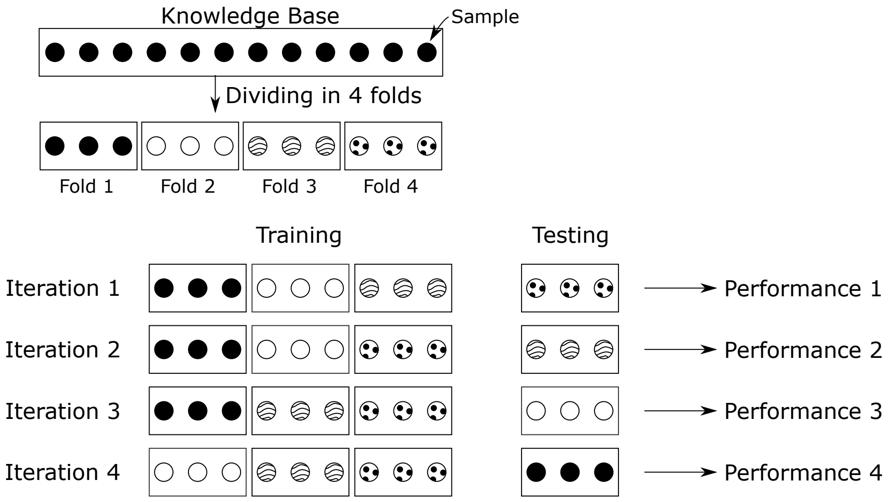


Figure 5.2: Example of k-fold method

train the classifier. Finally, in the fourth iteration, the testing is performed using the *Fold 1*, and the rest are used to train the classifier. Then, the performances of the classifiers trained in each iteration are used to calculate the central tendency and the variance. These help us to determine if a classifier trained using the whole knowledge base will be able or not to properly assess model fragments.

5.3 Testing phase

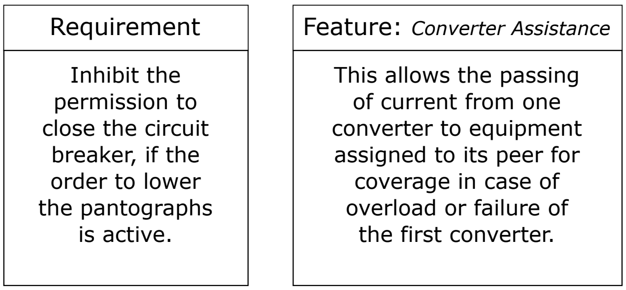
The target of the testing phase is to assess a model fragment population in order to determine which model fragment better realizes a query. To do this, the classifier, that is trained in the training phase, assigns a fitness value to each model fragment in the population. This fitness value corresponds to how well the model fragment realizes the query. Therefore, a high fitness value means that the model fragment is a good realization of the query. In contrast, a low fitness value means that the model fragment does not properly realize the query.

When each model fragment in the population has a fitness value, the fitness values are used to order the model fragments in a ranking. The top model fragment is the best realization of the query according to the ranking. The left part highlighted in the Figure 5.1 shows the artifacts and steps of the testing phase, which are described in more details in the following subsections.

5.3.1 Input and output artifacts

The input artifacts, that are used in the testing phase, are the query, the model fragment population, and the ontology.

The **query** is a natural language description of a requirement, a bug, or a feature depending on the software maintenance task. Traceability Link Recovery focuses on retrieving the traces between requirements and models. Therefore, the query corresponds to a requirement, which is written before development, is client influenced, and is for contracts. Bug Location focuses on locating the model elements affected by a bug. Therefore, the query corresponds to a bug description, which is written during the development, is internal, and is for reporting possible technical problems. Feature Location focuses on locating the model elements that realize a feature. Therefore, the query corresponds to a feature description, which is written when products already exist, is internal, and is for reuse. Therefore, the queries can be written in a different phase of the development, in a different style, and with a different goal in mind. Figure 5.3 shows an example of a requirement and a feature in railway domain, which were used in our works [9, 14] and [13], respectively.



Requirement

Inhibit the permission to close the circuit breaker, if the order to lower the pantographs is active.

Feature: *Converter Assistance*

This allows the passing of current from one converter to equipment assigned to its peer for coverage in case of overload or failure of the first converter.

Figure 5.3: Example of queries: requirement and feature descriptions

The **model fragment population** is a set of model fragments that belong to a model. The model fragments are composed for one or more elements of the model, which is known as density [16]. The elements can be connected between them or not, which is known as dispersion [16]. Moreover, the model fragments can be unique or not in the model, which is known as multiplicity [16].

Figure 5.4 shows two examples of model fragments. The first model fragment contains five model elements: a pantograph, a circuit breaker, a converter, and two connections. In contrast, the second model fragment only contains two model elements: a HVAC (air conditioning system) and a PA (public address system). In the first model fragment, all the model elements are connected between them. In contrast, in the second model fragment, the model elements are not connected between them. Furthermore, the first model fragment is not unique. The first model fragment is composed of the *Pantograph 1*, the *Circuit Breaker 1*, the *Converter 1*, and the connections between them. These model elements (a pantograph, a circuit breaker, a converter, and two connections) can be found in another model fragment of the same model, which is composed of the *Pantograph 2*, the *Circuit Breaker 2*, the *Converter 1*, and the connections between them. In contrast, the second model fragment is unique, because it is not possible to find other model fragment with a HVAC and a PA.

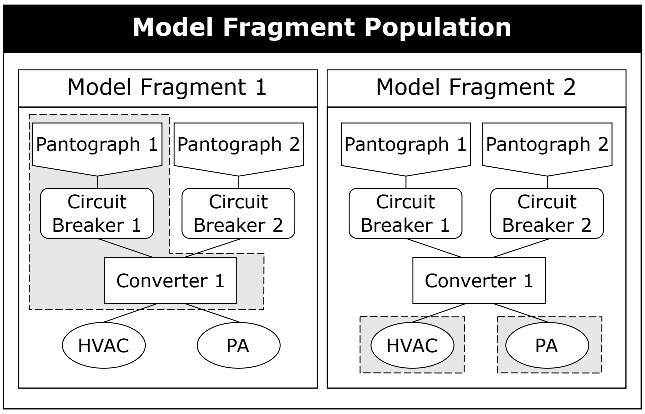


Figure 5.4: Example of two model fragments from the same model

The **ontology** contains the main concepts and relations of a domain. Therefore, the ontology contains all the necessary information to encode the knowledge base into feature vectors, following the encoding described in Chapter 4. There are more details about the ontology and an example in Section 4.2.

The other artifacts in the testing phase are the testing set and the model fragment ranking.

The **testing set** contains feature vectors, where each feature vector encodes a model fragment of the population and the query. Specifically, each feature vector contains a fitness value and a set of characteristic/value pairs that characterizes a model fragment in the population and the query (see Figure 5.5. In fact, the model fragment population and the query are encoded using the ontological evolutionary encoding presented in Chapter 4. However, there are small differences between the encoding of a knowledge base and the encoding of a model fragment population and a query. These differences are described in the first step of the testing phase (see Section 5.3.2).

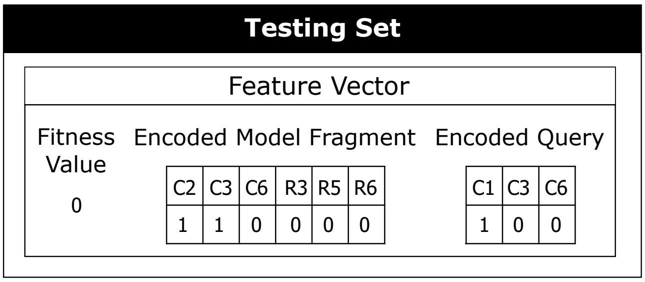


Figure 5.5: Example of a feature vector in the testing set

The **model fragment ranking** is the output of the testing phase. The model fragment ranking contains all the model fragments of the population ordered by their fitness values. These values are determined for the classifier in the testing phase and indicate how well each model fragment realizes the query. Figure 5.6 shows an example of model fragment ranking. In this figure, two model fragments are ordered taking into account their fitness values. The model fragment with the highest value is in the top of ranking.

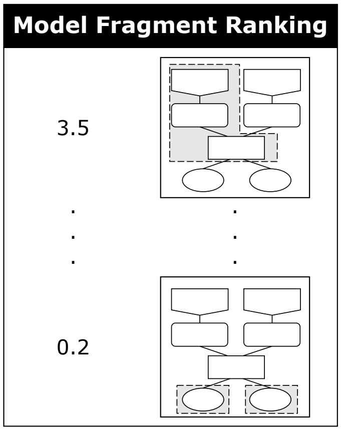


Figure 5.6: Example of a model fragment ranking

5.3.2 *Steps of testing phase*

The testing phase has two steps: ontological evolutionary encoding and Classifier. The first one is applied to turn the model fragment population and the query into a testing set. The second one is applied to rank the testing set through the classifier.

Ontological Evolutionary Encoding

In the training phase, the knowledge base has to be encoded into feature vectors in order to train the classifier through a ML technique. In a similar way, the input artifacts of the testing phase (i.e. the model fragment population and the query) have to be encoded into feature vectors in order to use that classifier. The ontological evolutionary encoding described in Chapter 4 is designed to encode the samples of the knowledge base into feature vectors. Each sample contains a model fragment, a query, and a fitness value. However, among the input artifacts for the testing set, there is only a query and there are not fitness values. Therefore, it may initially seem that the ontological evolutionary encoding cannot be used to encode the input artifacts of the testing phase.

Nevertheless, regarding the query, the same query is considered for each model fragment in the population. The target of the testing phase is to assess which model fragment in the population better realizes the query. Therefore, the query is the same for all the model fragments in the population. For this reason, in the testing phase, each sample contains a model fragment of the population and the same query.

On the other hand, regarding the fitness values, these numerical values have no relevance in the testing phase. In the training phase, these values are fundamental to train the classifier. However, the objective of the testing phase, and of classifier in particular, is to determine these values for the model fragments in the population. Therefore, when the classifier assesses the testing set, it ignores these values and determine what fitness value is appropriated for each model fragment.

Therefore, the model fragment population and the query can be embedded into samples, where each sample contains a model fragment of the population, the same query, and a default fitness value (e.g. 0 or -1). Then, the samples can be encoded into feature vectors following the encoding described in Chapter 4. The obtained feature vectors compose the testing set.

5.3.3 Classifier

In this step, the classifier is used to assess the testing set. Specifically, the classifier assigns a fitness value to each feature vector in the testing set. The assigned fitness values are numerical values greater than 0. In fact, the higher the fitness value, the better the model fragment will be for the query.

Since each feature vector in the testing set corresponds to a model fragment in the population, the fitness values can be ordered in a ranking. The top positions are occupied by the model fragments with the highest fitness values. Therefore, the top model fragment is the best realization of the query according to the fitness values assigned for the classifier. Moreover, the ranking is the output of the fitness function.

5.4 Summary of the fitness function

This section provides a summary of the fitness function, which is described in the previous sections. The objective of the fitness function is to assess model fragments through ML techniques in order to determine which model fragment of a product model better realizes a specific query. To do this, the fitness function consists of two phases: training phase and testing phase.

In the training phase, a classifier is trained by a LtoR algorithm, which belongs to a family of ML techniques for ranking tasks. Thanks to the training, the classifier learns when a model fragment is a bad or a good realization of a query and what fitness value has to be assigned in each case.

In the testing phase, the classifier is used to assess a model fragment population in order to determine which fragment better realizes a specific query. Specifically, the classifier assigns a fitness value to each model fragment in the population. Then, the model fragments are ordered in a ranking according to their fitness values. Based on this ranking, the top model fragment is the best realization of the query.

As output, the fitness function returns the model fragment ranking, which contains the model fragments of the population ordered according to their fitness values. Figure 5.7 shows an overview of the fitness function, including examples for the inputs and outputs of the testing phase. The examples for the inputs and outputs of the training phase (i.e. the knowledge base, the ontology, and the training set) have been already shown in Section 4.1.

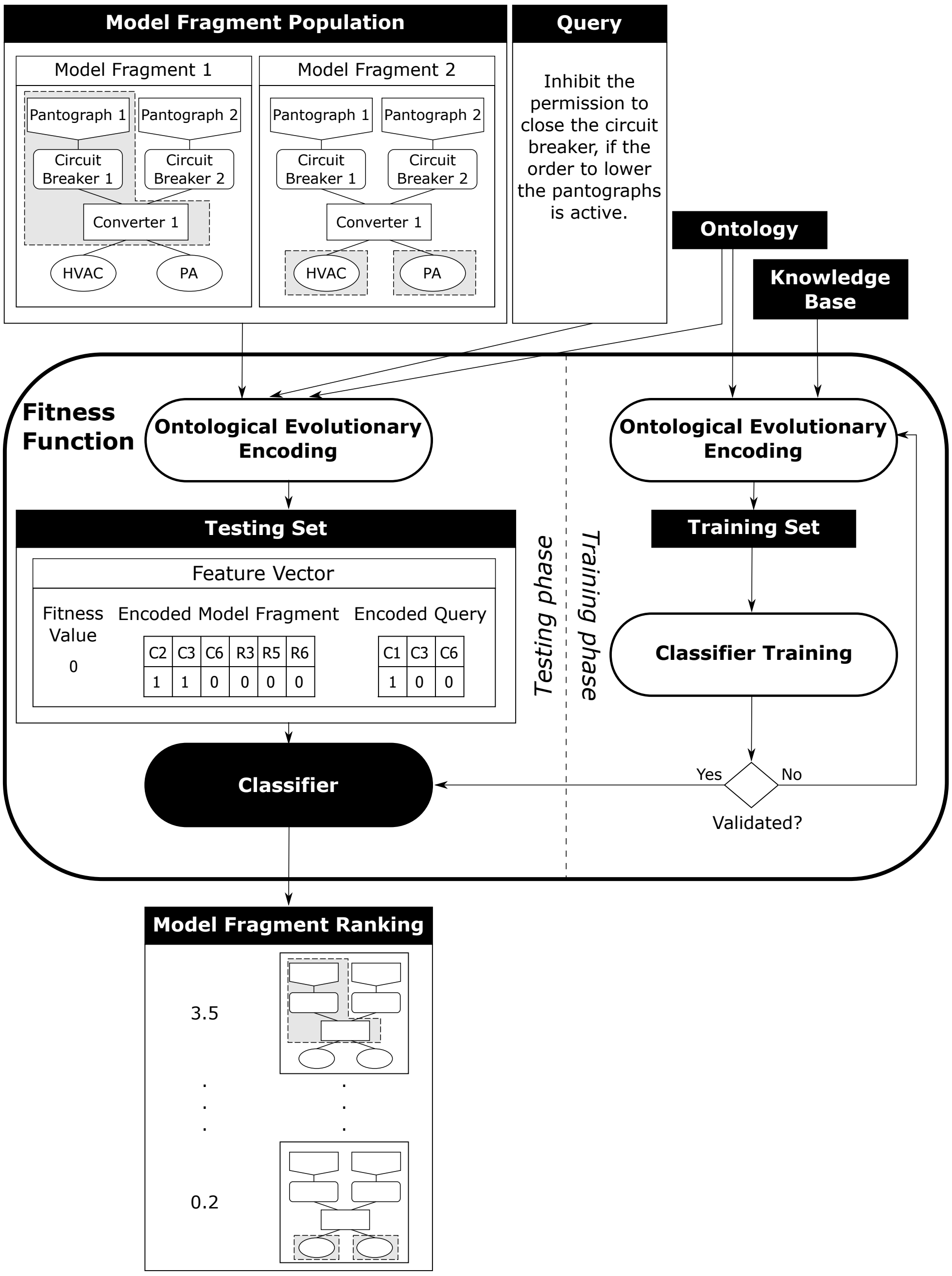


Figure 5.7: Overview of the ontological evolutionary encoding approach.

Chapter 6

Evolutionary Algorithm

Contents

6.1 Overview of the chapter

A high amount of model fragments can be extracted from a product model. Even with an automatic approach, to assess all these model fragments would take months of work. In this chapter, we propose the use of an evolutionary algorithm to guide the extraction of the model fragments from a model. This guide allows to extract model fragments based on fitness values instead of extracting the whole set of model fragments from a model.

Specifically, the evolutionary algorithm extracts an initial model fragment population, which is evolved using genetic operations. Then, the fitness function assesses the model fragments in the population, assigning a fitness value to each model fragment. Then, model fragment population with the assigned fitness values (i.e. the evaluated model fragment population) is used for the genetic operations to extract more model fragments from the model. Taking into account the fitness values, the genetic operations know which model fragments better realize the query. Therefore, the genetic operations can extract similar model fragments to the best ones in the evaluated model fragment population. This guides the extraction towards model fragments that are good realizations of the query.

The genetic operations and the fitness function are repeated until the solution converges to a certain stop condition. Then, the evaluated model fragment population is ordered according to the fitness values in a ranking. The model fragments with the highest fitness values are in the top of the ranking. This ranking (i.e. the model fragment ranking) is the final output of the FRAME approach.

Figure 6.1 shows an overview of the FRAME approach. This figure highlights in black colour the steps and the artifacts of the approach that are described in this chapter. According to this figure, the evolutionary algorithm is divided in three stages, but only this chapter focuses on the two first stages. The first one, the initialization, extracts the initial model fragment population. The second one, the genetic operations, evolves the model fragment populations to extract other model fragments from the model. The third one, the fitness function, assesses the model fragment in the populations to determine which model fragments better realize the query. As output, the evolutionary algorithm returns a model fragment ranking, which contains the model fragments ordered by their fitness values.

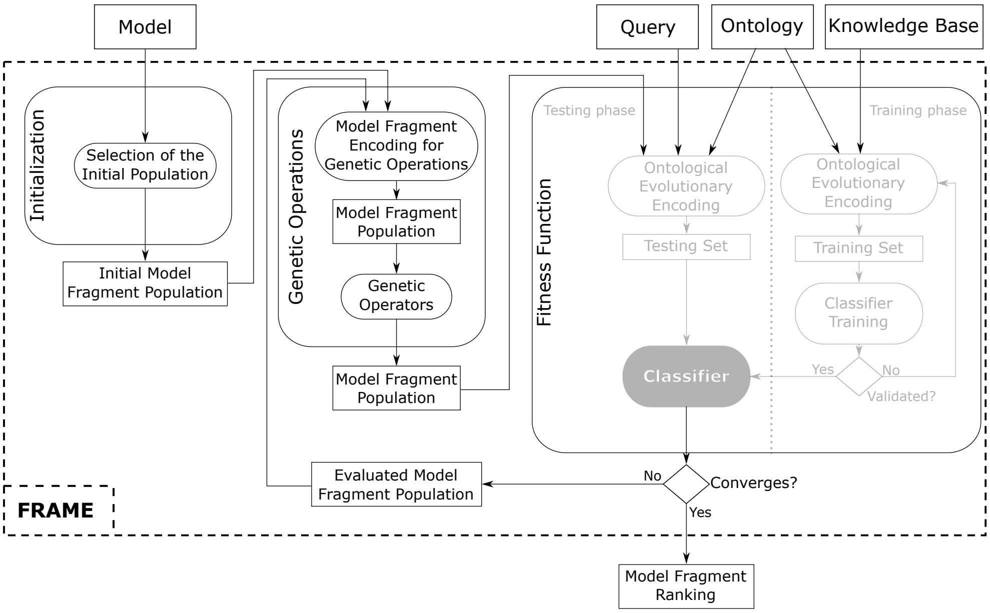

Model
Query
Ontology
Knowledge Base
Initialization
Selection of the Initial Population
Initial Model Fragment Population
Genetic Operations
Model Fragment Encoding for Genetic Operations
Model Fragment Population
Genetic Operators
Model Fragment Population
Fitness Function
Testing phase
Training phase
Ontological Evolutionary Encoding
Testing Set
Ontological Evolutionary Encoding
Training Set
Classifier Training
Classifier
Yes
No
Validated?
Evaluated Model Fragment Population
No
Converges?
Yes
FRAME
Model Fragment Ranking

Figure 6.1: Overview of the evolutionary algorithm in the FRAME approach

The following sections present the input and output artifacts (Section 6.2) and the three stages of the evolutionary algorithm (Section 6.3). Finally, the last section presents an overview of the whole FRAME approach (Section 6.4).

6.2 Input and output artifacts

The input of the evolutionary algorithm consists of a model, a query, an ontology, and a knowledge base (see Figure 6.1).

The **model** is the artifact where the query has been searched. That is, the model is composed of different elements and the aim of the evolutionary algorithm is to extract model fragments that contain one or more of the model elements.

The evolutionary algorithm, and the FRAME approach in general, have been designed to work with models that conform to MOF (the OMG metalanguage for defining modeling languages). In particular, the approach has been evaluated using models, taken from a real-world train, specified using a Domain Specific Language (DSL) that formalizes the train control and management of the products manufactured by our industrial partner. The DSL has the expressiveness required to describe both the interaction between the main pieces of installed equipment, and the non-functional aspects related to regulation.

Figure 6.2 depicts a model example, taken from a real-world train, specified using the DSL used by our industrial partner. Specifically, the example of the figure presents a converter assistance scenario where two pantographs (High Voltage Equipment) collect energy from the overhead wires, and send it to their respective circuit breakers (Contactors), which in turn send it to their independent Voltage Converters. The converters then power their assigned Consumer Equipment: the HVAC on the left (air conditioning system) and the PA (public address system) on the right.

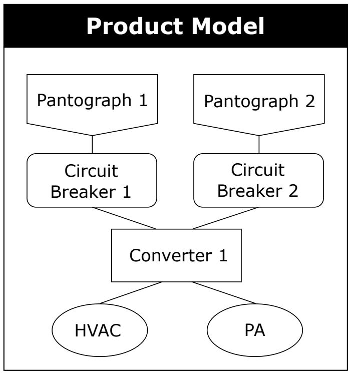


Figure 6.2: Example of a model, taken from a real-world train

The **query** is the artifact for which a model fragment must be retrieved. Specifically, the query is a natural language description of a requirement, a bug, or a feature depending on the software maintenance task. There are more details about the query and an example of a requirement and a feature in Section 5.3.1.

The **ontology** is the artifact for encoding. The ontology contains the main concepts and relations of a domain. Therefore, the ontology contains all the necessary information to encode the model fragment population and the knowledge base into feature vectors, following the encoding described in Chapter 4. There are more details about the ontology and an example in Section 4.2.

The **knowledge base** is the artifact used to learn how to assess the model fragments. Specifically, the knowledge base contains the model fragments that have been manually retrieved by the engineers and modellers for years. The knowledge base consists in a set of samples, where each sample contains a query, a model fragment, and a fitness value. The query uses natural language to describe the requirement, bug, or feature, that was searched by the engineer in a product model. The model fragment consists of the element or the set of elements that the engineer manually retrieved from the model taking into account the description in the query. The fitness value determines how well the model fragment realizes the query. There are more details about the knowledge base and an example in Section 4.2.

The other artifacts of the evolutionary algorithm are the initial model fragment population, the model fragment population, the evaluated model fragment population, and the model fragment ranking.

Both the **initial model fragment population** and the **model fragment population** are sets of model fragments that belong to a model. The initial model fragment population contains the model fragments extracted through the initialization stage of the evolutionary algorithm. On the other hand, the model fragment population contains the model fragments extracted through the genetic operations stage of the evolutionary algorithm. There are more details about the model fragments in the populations and two examples of model fragments in Section 5.3.1.

The **evaluated model fragment population** and the **model fragment ranking** are the artifacts obtained from the fitness function. Specifically, these artifacts contain model fragments associated to fitness values. Each fitness value indicates how well the model fragment realizes the query. The unique difference between these artifacts is that, in the model fragment ranking, the model fragments are ordered by their fitness values. The model fragment with the highest value is in the top of the ranking. There are more details about the model fragment ranking and an example in Section 5.3.1.

6.3 Stages of the evolutionary algorithm

The evolutionary algorithm consists of three stages: initialization, genetic operations, and fitness function.

6.3.1 Initialization

The initialization stage is to extract a population of model fragments from the model, which serves as input for the evolutionary algorithm. The initialization stage has only one step: extraction of the initial population, where the model fragments are extracted.

To extract a model fragment, all the elements of the model are considered. A selection function is used to randomly determine if each model element is or is not part of the model fragment. Therefore, a model element can be present in different model fragments or a model element can be not present in any of the model fragments.

The selection function is repeatedly used to extract several model fragments for the initial population. This population is the output of the initialization stage.

Note that the initialization stage is performed only one time for each query that wants to be searched.

6.3.2 Genetic Operations

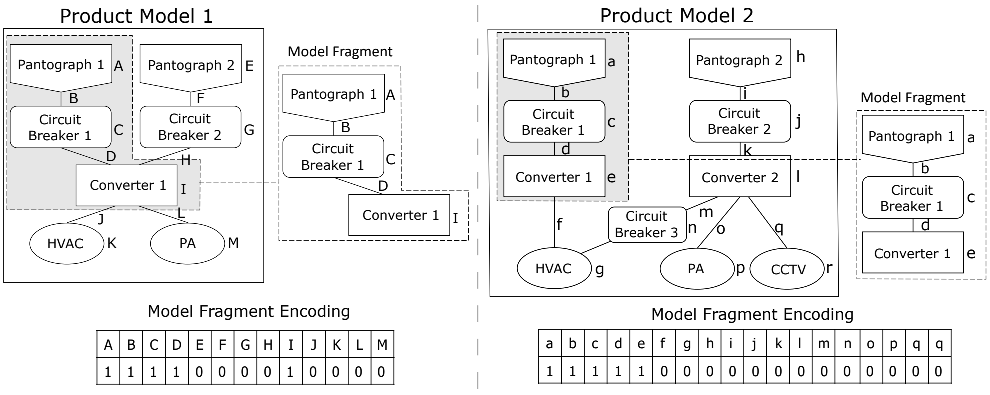
The second stage of the evolutionary algorithm is the genetic operations. In this stage, genetic operations are applied to a model fragment population in order to generate a new model fragment population.

The initial model fragment population contains a small amount of the model fragments in a model. Moreover, the model fragments of the population are extracted by randomly selection of the model elements. Therefore, we cannot ensure that the initial model fragment population contains a model fragment that is a good realization of the query. In fact, the more fragments can be extracted from a model, the lower the probability that the initial population contains a model fragment that realize the query well. Therefore, three genetic operators (selection, crossover, and mutation) are used to generate new model fragment populations, which can contain better model fragments for the query.

To do that, the genetic operations stage has two steps: model fragment encoding for genetic operations and the genetic operators. The following sections provide more details about these two steps.

Model Fragment Encoding for Genetic Operations

Traditionally, in evolutionary algorithms, each possible solution of the problem is encoded as a string of binary values. However, encoding each model fragment as a string of binary values is not straightforward. The authors in [99] propose an encoding where each model fragment is encoded as an individual in relation to the model. In other words, each individual is a set of model elements that are present or absent in a model fragment.



A	B	C	D	E	F	G	H	I	J	K	L	M
1	1	1	1	0	0	0	0	1	0	0	0	0

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	q
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 6.3: Examples of Model Fragment Encoding

Figure 6.3 shows two examples of the representation of model fragments. Each letter labels a model element of the model. Therefore, the individual contains as many positions as model elements in the model and the binary value of these positions depends on the presence or absence of the model elements in the model fragment. If the model element appears in the model fragment, the value will be 1; if the model element does not appear in the model fragment the value will be 0.

Figure 6.3 also shows that the encoding will be different for different models, even though the model fragment to be encoded is the same. Both of the examples in Figure 6.3 represent the same model fragment. However, since they come from different models, their representations are different.

The FRAME approach has two encodings for model fragments: the ontological evolutionary encoding and the model fragment encoding for genetic operations. In the ontological evolutionary encoding, both the model fragments and the query are encoded, the values are integers, and the length is the same for all the encoded model fragments. In the model fragment encoding for genetic operations, the query is not considered, the values are binaries, and the length depends on model elements. These differences highlight the need of different encodings for ML techniques and genetic operations.

Genetic Operators

The generation of new model fragments (based on existing ones) is done by applying a set of three genetic operators, which are adapted to work on model fragments. These genetic operations were introduced for the first time in [116] to carry out the selection of parents, the crossover, and the mutation of model fragments. Figure 6.4 shows an example of the application of the genetic operators. Moreover, each genetic operator is described in more detail below.

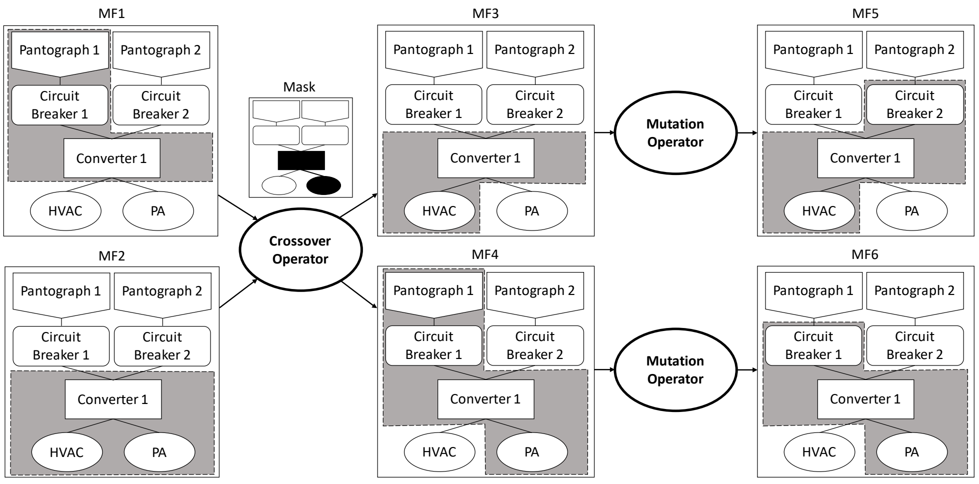


Figure 6.4: Example of genetic operations

- The **selection operator** picks the best candidates from the population as input for the rest of the operators. There are different methods that can be used to perform the selection of the parents. One of the most widespread methods (adopted by our work) is to follow the wheel selection mechanism [117], where each model fragment from the population has a probability of being selected that is proportional to its fitness value. Candidates with high fitness values have higher probabilities of being chosen as parents for the next generation.

- The **crossover operator** enables the creation of two new individuals by combining the genetic material from two model fragments. A randomly generated mask determines how the combination is done, indicating for each element of the model fragments if the offspring should inherit from one model fragment or the other.

Specifically, the mask is created randomly and all of the model elements have a 50% probability of belonging to the mask. To do this, a random number (0 or 1) is generated for each model element. The elements whose value is 1 belong to the mask and the elements whose value is 0 belong to the inverse of the mask. Moreover, a model fragment is a subset of the elements that are present in a model.

Since both model fragments are extracted from the same model, their combination will always return a model fragment that is part of the original product model. As a result of the crossover operation, two individuals are generated: one by directly applying the mask, and the other one by applying the inverse of the mask, as is usually done in genetic algorithms [118].

Figure 6.4 shows an example of the application of the genetic operators. Taking into account the crossover operation, the model fragment MF1 is combined with the model fragment MF2 according to a mask that contains two sets of elements (one regular and one marked in black). To create the first of the new candidate model fragment, we interpret the mask by selecting the blackened elements from the first parent (MF1) and the regular elements from the second parent (MF2). As a result, the new model fragment (MF3) contains the set of elements that are present in the mask in MF1 and the set of elements that are absent in the mask in MF2. In addition, the mask is also interpreted in the opposite way by selecting the blackened elements from MF2 and the regular elements from MF1, thus producing another new and distinct model fragment (MF4).

- The **mutation operator** is used to imitate the mutations that occur randomly in nature when new individuals are born. In other words, new individuals have small differences with their parents that could make them adapt better (or worse) to their living environment. Following this idea, the mutation operator applied to model fragments [116] takes as input a model fragment and mutates it into a new one, which is returned as output.

  Specifically, the mutation operator can perform two kinds of modifications: the addition of elements to the model fragment, or the removal of elements from the model fragment. Since the approach is looking for fragments of the model that realize a specific requirement, the new modified fragment must remain a part of this model. Therefore, the modifications that can be done to the model fragment must be driven by the model, which determines the additions and subtractions of elements that can be applied to the model fragments in the population.

Figure 6.4 shows an example of the application of the genetic operations. Taking into account this example, the mutation operation takes the first offspring produced through the crossover operator and adds one element (the second circuit breaker). Then, the mutation operation takes the second offspring and removes one element (the first pantograph). The resulting model fragments (MF5 and MF6) are new candidates in the population for the realization of the query.

After applying the genetic operators, it may be that not all of the elements of the new individuals are connected. Indeed, the query can be implemented by several model elements that are not directly connected in the model [116]. Therefore, it is necessary to create model fragments of this kind since they could be the ones realizing the query.

6.3.3 Fitness Function

Finally, the last stage of the evolutionary algorithm is the fitness function. The fitness function is based on ML techniques, specifically in a LtoR algorithm. The LtoR algorithm allows the fitness function to learn how to assess model fragments based on a knowledge base. However, to apply the LtoR algorithm, the fitness function has to encode the model fragments using an ontology.

Once the fitness function knows how to assess the model fragments, it can assign a fitness value to each model fragment of the population. The fitness value indicates how well a model fragment realizes the query. As output, the fitness function returns the evaluated model fragment population, which contains the model fragments with their assigned fitness values. The fitness function is presented in more detail in the Chapter 5.

The last two stages of the evolutionary algorithm (genetic operations and fitness function) are repeated until the solution converges to a certain stop condition. Usually, the stop condition can be a time slot, a fixed number of iterations, or a trigger value of the fitness that makes the process finish when reached [116]. Since the stop condition greatly depends on the domain and the problem being solved, it is adjusted depending on the results being output, taking into account when the fitness values are converging and no further improvements are being made by new iterations [116]. When the stop condition is met, the evolutionary algorithm provides a model fragment ranking, which contains the evaluated model fragments ordered by their fitness values.

6.4 Overview of the FRAME approach

The second objective (OB2) of this dissertation is to provide a ML-based approach to automatically perform fragment retrieval in software maintenance tasks on models. To do this, we need to tackle three design problems: the characterization of model fragments, how to assess model fragments regarding a query, and the extraction of model fragments from a model.

The solution for the first design problem is the ontological evolutionary encoding, which is presented in Chapter 4. The solution for the second design problem is the fitness function, which is presented in Chapter 5. Finally, the solution for the third design problem is the evolutionary algorithm, which is presented in the previous sections of this chapter (Chapter 6). Therefore, taking into account these solutions, the design of the FRAME approach is complete. In this section, we present an overview of the whole approach.

Figure 6.5 shows the overview of the approach. The top of figure shows the input artifacts. The center of figure shows the main steps of our approach. The bottom of the figure shows the output artifact. Rounded rectangles represent the different steps of the approach, and straight rectangles represent the inputs and outputs of each of the steps.

The approach has been designed to retrieve the model fragments, which better realize a query in a model. To do this, the approach receives as input the model, the query, an ontology, and a knowledge base (see the top of Figure 6.5). The approach relies on an evolutionary algorithm, which has three stages:

1 **Initialization:** The first stage is to extract a population of model fragments from the model, which serves as input for the evolutionary algorithm. In order to extract the population of model fragments, elements of the model are randomly selected to compose different model fragments. These model fragments are added to initial model fragment population.

2 **Genetic operations:** Second, genetic operations are applied to the model fragment population in order to generate new candidate model fragments for the query. To do this, the model fragments in the population are encoded, and then, evolved using three genetic operators: the selection operator, the crossover operator, and the mutation operator.

3 **Fitness function:** Third, the model fragment population is assessed through the fitness function, which assign a fitness value to each model fragment of the population. To do this, the fitness function has two phases.

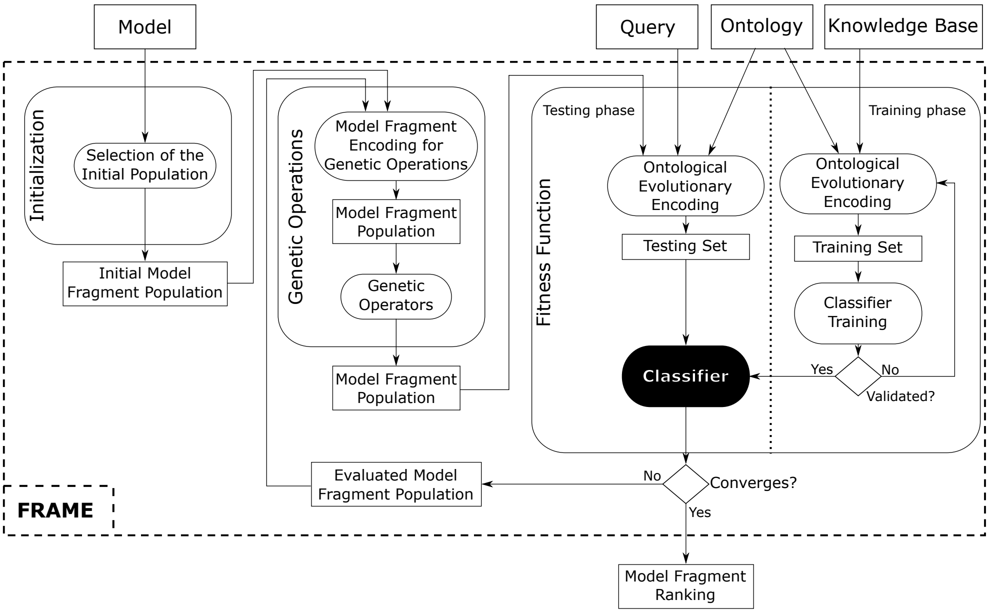

Model
Query
Ontology
Knowledge Base
Initialization
Selection of the Initial Population
Initial Model Fragment Population
Genetic Operations
Model Fragment Encoding for Genetic Operations
Model Fragment Population
Genetic Operators
Model Fragment Population
Fitness Function
Testing phase
Training phase
Ontological Evolutionary Encoding
Testing Set
Ontological Evolutionary Encoding
Training Set
Classifier Training
Classifier
Yes
No
Validated?
No
Converges?
Yes
Evaluated Model Fragment Population
FRAME
Model Fragment Ranking

Figure 6.5: Overview of the FRAME approach

The first phase, the training phase, is performed only in the first iteration of the evolutionary algorithm. In this phase, the knowledge base is encoded into feature vectors through the ontological evolutionary encoding, and then, used to train the classifier through a LtoR algorithm.

The second phase, the testing phase, is performed in all the iterations of the evolutionary algorithm. First, the model fragments of the population and the query are encoded using the ontological evolutionary encoding. Second, the classifier, that has been trained in the training phase, is used to assign the fitness value to each model fragment.

The last two stages of the evolutionary algorithm (genetic operations and fitness function) are repeated until the solution converges to a certain stop condition. However, after the first iteration, only the testing phase of the fitness function is repeated in each new iteration of the evolutionary algorithm provides a model fragment ranking, where model fragments are ordered according to their fitness values. The top positions are occupied by the model fragments with the highest fitness values.

PART IV

Treatment Validation

Never tell me the odds!
Star Wars

Chapter 7

EVALUATION DESIGN

Contents

7.1 Overview of the chapter

This chapter presents the design of the evaluation to validate the FRAME approach proposed in this dissertation. Specifically, our industrial partner provided us the necessary documentation to evaluate our approach for Traceability Link Recovery (TLR), as the software maintenance task. The goal of this evaluation is not only to test out the approach, but also to compare our approach against the approaches that have obtained the best results for this software maintenance task in the literature.

Therefore, the evaluation was designed to address the following three research questions:

Evaluation-RQ1: Is it feasible to retrieve model fragments in industrial domains using the FRAME approach presented so far?

Evaluation-RQ2: What is the performance of FRAME approach on industrial models?

Evaluation-RQ3: Can the FRAME approach outperform significantly the results of the most common approaches for Traceability Link Recovery?

Figure 7.1 shows an overview of the process that was followed to perform this evaluation. The top part of Figure 7.1 shows the input artifacts, which are extracted from the documentation provided by our industrial partner: queries (requirements), models, an ontology, a knowledge base, and an oracle. Each test case is comprised of a requirement, a model, the ontology, and the knowledge base. The oracle is composed of the approved traceability between the requirement and the model of each test case. In other words, for each test case, the oracle contains a model fragment that is the correct solution to trace the query in the model.

Some of the approaches do not require all the input artifacts. For example, the TLR-Linguistic approach does not require the ontology or the knowledge base. In this case, the non-required artifacts are ignored by the approach. However, all the approaches have all the input artifacts at their disposal in the test cases.

Below the input artifacts, Figure 7.1 shows the approaches that are being evaluated and compared. The main techniques or methods used for each

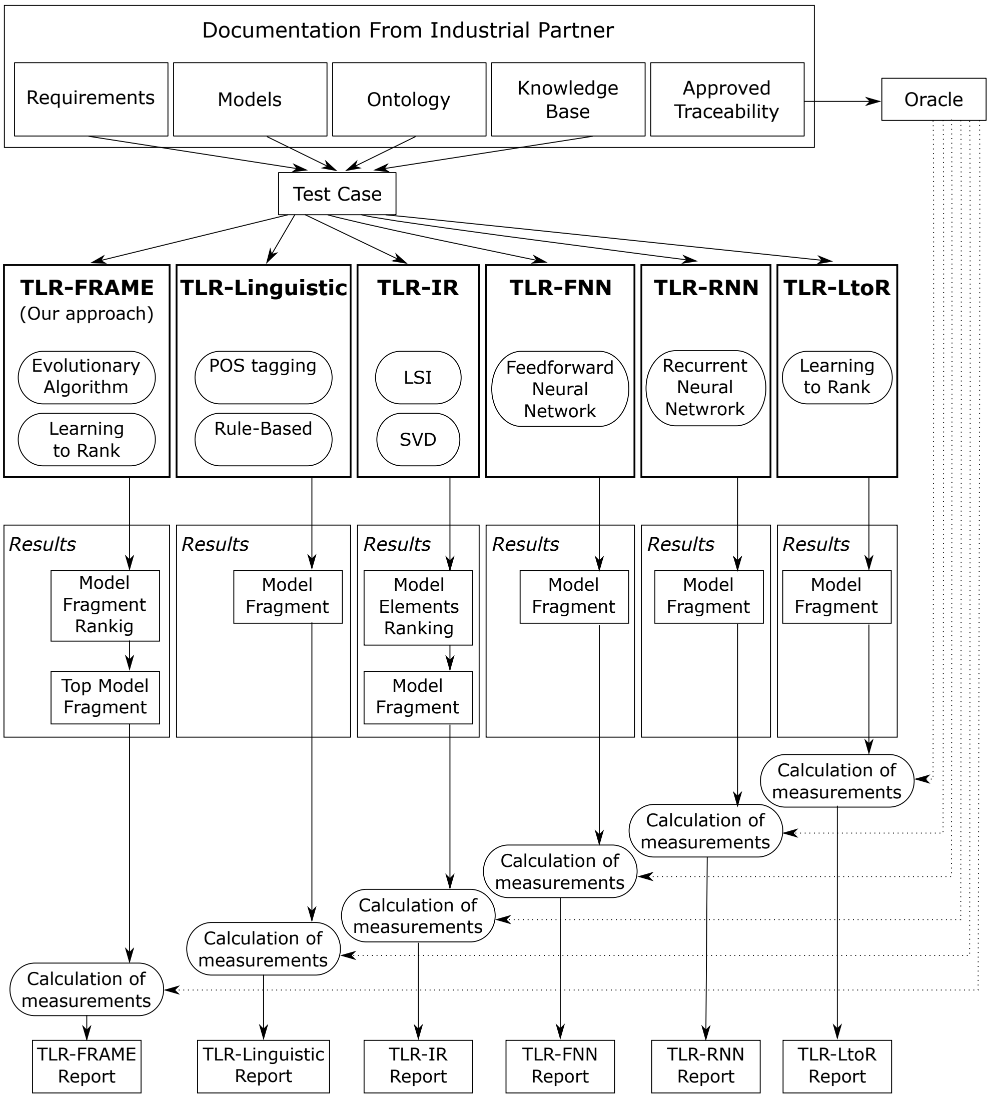


Figure 7.1: Setup of the evaluation

approach are highlighted in the figure. Specifically, the first approach the one proposed in this dissertation (TLR-FRAME). Then , the following two approaches (TLR-Linguistic and TLR-IR) are the traditional approaches that obtain the best results for Traceability Links Recovery on models. Both approaches are based on Natural Language Processing. Then, the following two approaches (TLR-FNN and TLR-RNN) are based on deep learning techniques that have also successfully been applied in Traceability Link Recovery in some recent works. Finally, the last approach (TLR-LtoR) is also based on Learning to Rank as our approach. However, this approach explores the model (search space) by means of brute-force instead of using an evolutionary algorithm.

After the approaches, Figure 7.1 shows the solutions of the approaches. Most of the approaches obtain a model fragment as solution. However, two of the approaches do not obtain directly a model fragment as solution. The FRAME approach obtains a model fragment ranking, so the model fragment in the top of the ranking is selected as final solution of the approach. Moreover, the TLR-IR approach obtains a model element ranking, so the model fragment is composed by the model elements with a degree of similarity equal or greater than to $x = 0.7$ in the ranking. Specifically, this degree of similarity was chosen since this heuristic has yielded good results in other similar works [119, 120].

Therefore, although in some cases is not direct, all the approaches obtain a model fragment as solution of each test case. Then, as Figure 7.1 shows, the model fragments are evaluated through measures reporting the correspondent results.

The following sections describe the real case (Section 7.2), describe the approaches and their setup in this evaluation (Section 7.3), explain how results are measured (Section 7.4), and present the threats to validity (Section 7.5).

7.2 Real Case: Test Cases and Oracle

The real case where we applied our approach was provided by our industrial partner CAF, a worldwide provider of railway solutions. Specifically, our industrial partner provided the necessary documentation to evaluate our approach in a Traceability Link Recovery problem. Traceability Link Recovery on models focuses on recovering the model fragment that better realizes a requirement (query) in a model. Therefore, the provided documentation consists in several requirements to be searched, several models where to search the requirements, the domain ontology and the knowledge base to apply the approach, and the oracle with the correct solutions.

From documentation, 20 test cases were defined. Each test case was composed of a requirement, a product model, the knowledge base, and the ontology. A detailed description of each of them is provided below:

- The **requirements** have about 25 words.
- The **models** have about 650 elements.
- The **knowledge base** includes 103 samples. Specifically, each of these samples contains a requirement, a model fragment that has about 15 elements, and an fitness value. Figure 7.2 shows the distribution of fitness values in the knowledge base, taking into account how many samples of the knowledge base are in each range of fitness values.

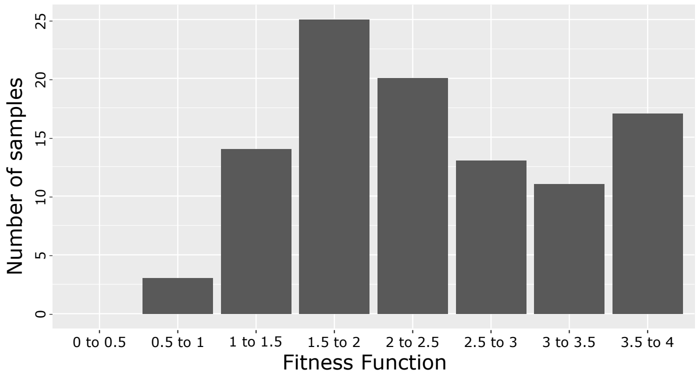


Figure 7.2: Distribution of fitness values in the knowledge base

- The **ontology** contains a total of 54 elements between concepts and relations. Specifically, the ontology contains 24 concepts and 30 relations. Therefore, based on this ontology, the feature vectors contain a total of 78 characteristics: 54 characteristics to encode the model fragments and 24 characteristics to encode the queries. Although, due to the feature selection in the encoding, only around 90% of these features are enabled.

For each test case, we followed the experimental setup described in Figure 7.1. Each test case was run 30 times. As suggested by [121], given the stochastic nature of the TLR-FRAME approach, several repetitions are needed to obtain reliable results. Finally, the solutions were evaluated and compared to the oracle. The oracle contains the approved traceability. In other words, the oracle contains the model fragments, which are manually retrieved by the domain experts and are the correct solutions for each test case. Since the case study contains 20 test cases, the oracle contains the 20 model fragments for these test cases, one for each test case.

7.3 Approaches under Evaluation

This section provides a description and the setup of each approach that was evaluated. In total, six approaches were considered in the evaluation. The first one was the one proposed in this dissertation: (1) the FRAME approach.

Winkler et al. [122] classify several approaches that have been created over the past 15 years that try to optimize the automatic identification of traces. Based on this classification, we selected the two approaches that obtain the best results for traceability links between requirements and models: (2) a rule-based approach that deduces traces by applying rules (TLR-Linguistic) [123]; and (3) an information retrieval approach that can detect candidate traceability links through Information Retrieval (TLR-IR) [124, 125].

Deep learning techniques have also successfully been applied in Traceability Link Recovery in some recent works [59]. Therefore, we decided to compare our approach with two approaches that apply deep learning: (4) the first one is based on a Feedforward Neural Network (TLR-FNN); and (5) the second one is based on a Recurrent Neural Network (TLR-RNN).

Finally, to check the need for the evolutionary algorithm in our approach, TLR-FRAME is also compared to (6) TLR-LtoR, which explores the search space by means of brute-force. Therefore, the model fragments are generated from the model and evaluated through LtoR, but the results obtained from the LtoR process are not used to guide the generation of new model fragments. Since there is no guide to explore the model, the search for the model fragment that realizes a specific requirement is performed by brute-force.

Note that since both requirements and models use the natural language, Natural Language Processing (NLP) techniques are used to process them before applying any approach. In fact, NLP has a direct and beneficial impact on the results of some approaches (e.g. the TLR-IR approach). Specifically, the queries and the models are processed by a combination of NLP techniques defined in [104], which consists of tokenizing, lowercasing, removal of duplicate keywords, syntactical analysis, lemmatization, and stopword removal.

7.3.1 TLR-FRAME: FRAME approach

The TLR-FRAME approach has been presented in the chapters 4, 5, and 6 of this dissertation. The approach is based on ML techniques, specifically in a LtoR algorithm, to determine what model fragments better realize a requirement (query). The model fragments are extracted from a model using an evolutionary algorithm.

To setup the approach for the evaluation, four technical details are addressed: the stop condition, the hyperparameters for the evolutionary algorithm, the LtoR algorithm with its tuning parameters, and the cross-validation method.

The stop condition greatly depends on the domain and the problem being solved. In general, there are two atomic performance measures for evolutionary algorithms: one regarding solution quality, and one regarding algorithm speed or search effort. In this paper, we focus on the solution quality (i.e., obtaining a solution that is more similar to the one from the oracle in terms of precision and recall). After running some prior tests to determine the number of iterations to converge (and adding a margin to ensure convergence), we allocated a fixed amount of iterations (200 iterations) to stop the execution.

For the evolutionary algorithm, we tuned the population size, the crossover probability, and the mutation probability. We have chosen the values 100, 0.9, and 0.1, respectively. These were selected based on the parameters that are commonly used in the literature [126] and the results of some preliminary tuning experiments.

The selection of the LtoR algorithm depends on several aspects, such as the size of the knowledge base. RankBoost [127] belongs to the family of LtoR and is well known for its efficiency and effectiveness in different domains [128, 129]. Moreover, Rankboost can benefit from a small knowledge base together with a small number of characteristics in the encoding to reduce the overfitting problem [130, 131]. Since this condition is satisfied by our real case, TLR-FRAME is based on Rankboost. Moreover, Rankboost has three parameters: number of iterations, threshold, and metric. These parameters were tuned using a grid-search with the values 200, 10, and ERR10, respectively.

Finally, our approach uses cross-validation to validate the classifier in the training phase of the fitness function. Specifically, in the evaluation, the cross-validation method used was k-fold with a k value equals to 4.

> The implementation for our approach is available at `http://bitbucket.org/svitusj/flame`. We have also made the dataset and the implementation for the other five approaches available in the same location.

> We used the Eclipse Modeling Framework to manipulate the models and Common Variability Language to manage the model fragments. The genetic operations were built upon the Watchmaker Framework for Evolutionary Computation [132]. Furthermore, RankBoost was implemented using the RankLib library [133].

7.3.2 TLR-Linguistic: Linguistic Rule-Based approach

Spanoudakis et al. [123] present a linguistic rule-based approach to support the automatic generation of traceability links between requirements and models. Specifically, the traceability links are generated following two stages:

Stage 1: a Parts-of-Speech (POS) tagging technique [134] is applied on the requirements that are defined using natural language.

Stage 2: the traceability links between the requirements and the models are generated through the *requirement-to-object-model* rules.

```
RTOM_RULE Rule-1:
EXISTS
  SEQUENCE(<x1/{NN1, NN2}>,<x2/{VBZ, VBR}>,<x3/{JJ}>) in Requirement;
  <x4/CLASS>, <x5/ATTRIBUTE> in Model
SUCH THAT
  ATTRIBUTE_OF(<x5>,<x4>) and CONTAINS(NAME(<x5>), <x3>) and (CONTAINS(NAME(<x4>), <x1>)
ACTION GENERATE
  OVERLAPS(Requirement, <x5>)
RTOM_RULE_END
```

Figure 7.3: Example of *requirement-to-object-model* rules

The *requirement-to-object-model* (RTOM) rules are specified by investigating grammatical patterns in requirements. Figure 7.3 shows an example of a RTOM rule. This rule establishes a relation between a requirement (third line) and an attribute in a model (fourth line), if the following conditions are satisfied:

- The requirement contains a noun (<`x1/{NN1, NN2}`>), followed by the verb *to be* in the present form (<`x2/{VBZ, VBR}`>), and then, there is an adjective (<`x3/{JJ}`>).
- The model contains at least a class that has one attribute (`ATTRIBUTE_OF(`<`x5`>`,`<`x4`>`)`). The name of the attribute is equal to the adjective in the requirement (`CONTAINS(`<`x5`>`,`<`x3`>`)`) and the name of class is equal to the noun in the requirement (`CONTAINS(`<`x4`>`,`<`x1`>`)`).

When these conditions are satisfied, the rule establishes a relation (`GENERATE OVERLAPS(Requirement,`<`x5`>`)`) between the requirement and the class in the model. Therefore, taking into account several rules, the TLR-Linguistic approach can identify the model elements that are related to the requirement. These model elements compose the model fragment, that is the output of the approach.

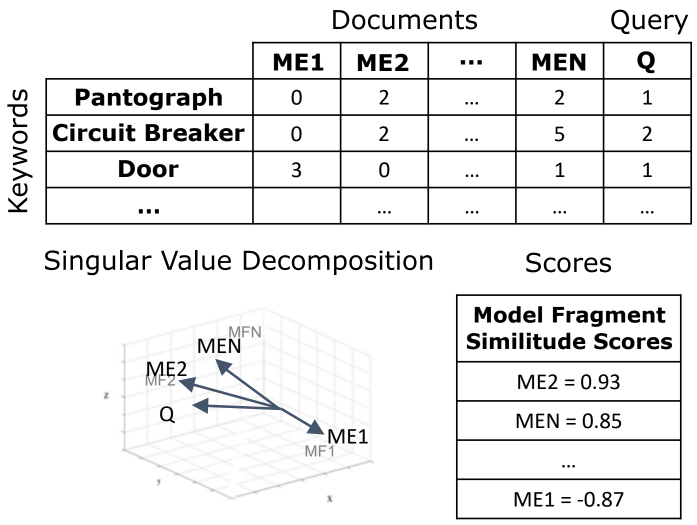
In [123], there are two different types of traceability rules: RTOM for traceability relations between requirements and model elements, and inter-requirement rules for traceability relations between different parts of a requirement statement. In total, the authors propose 26 rules for two domains: a software-intensive TV system created by Philips, and a university course management system. Since our approach is focused only on the traceability between requirements and model elements, this approach only tackles the RTOM traceability rules for our domain. Therefore, based on the guides and the examples of rules that are provided by [123], a domain expert who was not involved in the research generated an initial set of rules for our domain. In addition, to mitigate the dependence on a single domain expert, a second expert who also was not involved in the research extended the set of rules. In the end, the extended set contains nine RTOM rules, which is similar to the number proposed by [123]. However, there is no significant difference between the results obtained using the initial set and the results obtained using the extended set. Specifically, the results described in this dissertation correspond to the extended set, which are a bit better than those obtained from the initial set. Nonetheless, in both cases, the results are not as good as the ones obtained with our approach.

The Stanford POS Tagger [135] was utilized for the development of the TLR-Linguistic approach.

7.3.3 TLR-IR: Information Retrieval approach

Information Retrieval (IR) [136, 137, 138] is a sub-field of computer science that deals with the automated storage and retrieval of documents. IR techniques have been successfully used to retrieve traceability links between different kinds of software artifacts in different contexts [139, 140, 141, 142, 143]. Specifically, in [124] and [125], De Lucia et al. use Latent Semantic Indexing (LSI) to recover traceability links between requirements and different kinds of software artifacts, including models in the form of use-case diagrams, among others. We use LSI to recover traceability links between requirements and models as one of the approaches for our evaluation.

Latent Semantic Indexing (LSI) [144] is an automatic mathematical/statistical technique that analyzes relationships between queries and documents (bodies of text). LSI constructs vector representations of both a user query and a corpus of text documents by encoding them as a *term-by-document co-occurrence matrix* and analyzes the relationships between those vectors to get a similarity ranking between the query and the documents (see Figure 7.4).



Keywords	ME1	ME2	...	MEN	Q
Pantograph	0	2	...	2	1
Circuit Breaker	0	2	...	5	2
Door	3	0	...	1	1
...		...	...	...	...

Model Fragment Similitude Scores
ME2 = 0.93
MEN = 0.85
...
ME1 = -0.87

Figure 7.4: Example of Traceability Link Recovery using Latent Semantic Indexing

In Traceability Link Recovery, the query corresponds to the requirement and each document is a natural language representation of a model element extracted using the technique in [145]. The top of Figure 7.4 shows an example of *term-by-document co-occurrence matrix*, with values associated to our real case. Each row in the matrix (keywords) stands for each of the words that compose the query and the documents (e.g. pantograph or door). Each column in the matrix stands for a document and the final column stands for the query. Each cell in the matrix contains the frequency with which the term (keyword) of its row appears in the document denoted by its column. For instance, in Figure 7.4, the term ‘pantograph’ appears twice in the document of the second model element (ME2) and once in the query.

Then, 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 [144]. The bottom of Figure 7.4 shows a three-dimensional graph of the Singular Value Decomposition technique. For legibility reasons, only a small set of the columns is represented. To measure the degree of similarity between vectors, the cosine between the query vector and the document vectors is calculated. Cosine values that are closer to 1 denote a higher degree of similarity, and cosine values that are closer to -1 denote a lower degree of similarity. Similarity increases as vectors point in the same general direction (as more terms are shared between documents). With this measurement, the model elements are ordered according to their degree of similarity to the requirement (see on the bottom left part of Figure 7.4). This ranking of model elements is returned as output of the TLR-IR approach.

The LSI technique used within the TLR-IR approach was implemented using the Efficient Java Matrix Library (EJML [146]).

7.3.4 TLR-FNN: Feedforward Neural Network approach

Feedforward Neural Networks (FNNs) represent a traditional neural network structure and lay the foundation for many other structures [147]. Data flow always moves one direction, from input layer to hidden layer, then to output layer; it never goes backwards. Figure 7.5 shows the structure of a FNN where the FNN receives a vector of I input signals, $z = (z_1, z_2, ..., z_I)$. The neurons of the hidden layer assign to each input signal, Z_i, its respective weight, v_i, to strengthen or deplete the input signal. Weighted inputs are accumulated at each neuron and then an activation function determines the output (or firing strength) of each neuron, o. In fact, the strength of the output is further influenced by a threshold value, which is also referred to as the bias; thus, the activation function receives both the input signal and the bias to determine the output of each neuron [148].

In Traceability Link Recovery, the input signals of FNN corresponds to the query and one of the model elements. As output, FNN returns a binary value that indicates if the model element should be or not included in the model fragment. Then, the following model element is evaluated through FNN, and the process is repeated for each element in the model. The model elements, that returns 1 as its binary value, are included in the model fragment, which is the output of the approach. In [14], there is a more detailed explanation about the application of neural networks on models for Traceability Link Recovery.

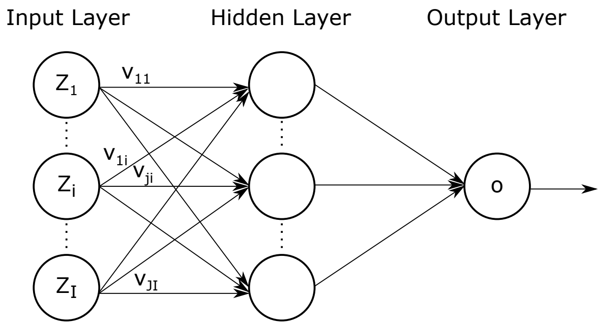


Figure 7.5: Feedforward Neural Network

While Figure 7.5 shows only one hidden layer, a FNN can have more than one hidden layer. However, it has been proved that FNNs with monotonically increasing differentiable functions can approximate any continuous function with one layer, provided that the hidden layer has enough hidden neurons [149]. Specifically, the network architecture of the FNN defined for the evaluation is a dense layer that is followed by the final softmax layer. Moreover, we performed a hyperparameter optimization based on the random search optimization provided by the Deep Learning for Java library. For all of the layers, the hyperparameter optimization resulted in an initial learning rate of 0.0035, and the Gaussian distribution recommended in [150] for weight initialization. In addition, for the dense layer, the hyperparameter optimization resulted in a layer size of 128 and the randomized rectified linear unit (RRELU) as the activation function.

The neural networks in TLR-FNN and TLR-RNN were developed and tuned by means of the Deep Learning for Java library [151].

7.3.5 TLR-RNN: Recurrent Neural Network approach

Since the number of parameters in a fully connected FNN can grow extremely large as the width and depth of the network increases, researchers have proposed other neural network structures targeting different types of practical problems. Recurrent Neural Networks (RNNs) are particularly well suited for processing sequential data such as text and audio. While FNNs have no feedback connections to previous layers, RNNs have these feedback connections to model the temporal characteristics of the problem being learned [148]. Moreover, RNNs have successfully been applied in Traceability Link Recovery in some recent works [59].

Although RNNs are specifically designed to process sequential data, RNNs have showed great results in some cases of non-sequential input information, for instance, image captioning [152] or prediction of hospital readmission [153]. In these works, even if the input data is not in the form of sequences, they can make classifiers able to learn so that they process data in sequential order only [153]. In our case, even if the models are not sequential data, we can order the model elements so that a classifier trained by a RNN benefits from the sequential order of the model elements.

Imagine that, among the model elements, the model contains a pantograph, a circuit breaker, and the connection between them (pantogragph-circuit breaker). First, we evaluated the pantograph through RNN. Even if we knew that pantograph is related to the requirement, we could not determine if the other two model elements are related to the requirement. Therefore, we evaluated the connection through RNN. If we knew that the pantograph and the connection (pantogragph-circuit breaker) are related to the requirement, it would be certainly reasonable to assume that the circuit breaker is related to the requirement. Therefore, the sequential order of the model elements may be exploited by RNN.

Figure 7.6 shows the structure of an Elman RNN, which is a RNN based on the extension of a FNN. As illustrated in Figure 7.6, data flow moves from an input layer to a hidden layer, but there is a new layer, named context layer, that makes a copy of the hidden layer. This context layer serves as an extension of the input layer, feeding signals that represent previous network states to the hidden layer. Therefore, the input vector is $z = (z_1, ..., z_I I, z_{I+1}, ..., z_{I+J})$, where the first I signals are the actual inputs of the network and the J signals are the context units [148].

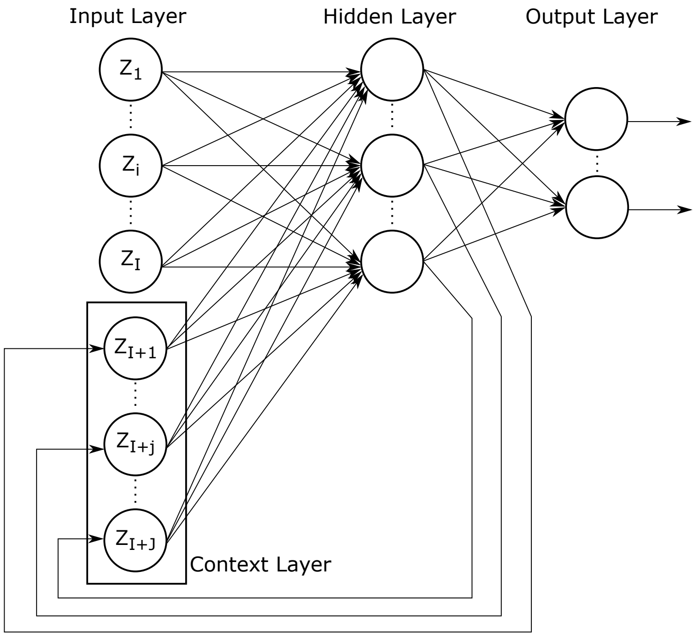


Figure 7.6: Elman Simple Recurrent Neural Network

As in the TLR-FNN approach, the input signals of RNN corresponds to the query and one of the model elements. As output, FNN returns a binary value that indicates if the model element should be or not included in the model fragment. This process is repeated for all the model elements, obtaining a model fragment as output. Specifically, the network architecture of the RNN implemented is a Long Short Term Memory layer followed by the final softmax layer. Moreover, we performed a hyperparameter optimization based on the random search optimization provided by the Deep Learning for Java library. For all of the layers, the hyperparameter optimization resulted in an initial learning rate of 0.02 and the Normal distribution described in [154] for the weight initialization. In addition, for the LSTM layer, the hyperparameter optimization resulted in a layer size of 223 and the standard sigmoid activation function as the activation function.

7.3.6 TLR-LtoR: Learning to Rank approach

Taking into account this approach, we want to determine if the better results of TLR-FRAME are due to the combination of the evolutionary algorithm and LtoR, or there is no need to combine the two to get these results. For this purpose, this approach is based only on LtoR, and the model fragments that are used as input for the LtoR process are extracted randomly through a standard random search.

We used this algorithm as outlined in Algorithm 2 (available in [10]). The algorithm starts with a random initial model fragment, as the best fragment. A new random model fragment is then assessed using LtoR. Then, the values provided by LtoR for both fragments, the best one and the new one, are compared and the model fragment with the greatest value is selected as the best one. The search then goes back to the second step, extracting and assessing a new model fragment, and this loop is repeated until a stop condition is met.

Therefore, this approach does not take advantage of evolving model fragments to guide the exploration of the models, as our approach does thanks to the evolutionary algorithm. Since there is no a guide to extract the models, the search for the model fragment that realizes a specific requirement is performed by brute-force.

Since TLR-LtoR is also based on LtoR such as TLR-FRAME, training and testing are also required. Therefore, this approach follows the same steps of the fitness function of the TLR-FRAME (Chapter 5). The ontological evolutionary encoding encodes the knowledge base as feature vectors, to train a classifier from these feature vectors. Then, this classifier is used to assess the model fragment, which are extracted through the standard random search. When the stop condition is met, the model fragment with the highest fitness value is returned as the output of the approach.

The LtoR algorithm (Rankboost) and the cross-validation method (k-fold) are the same as for the TLR-FRAME approach. However, the stop condition is different from TLR-FRAME in order so that the comparison between them is fair. For the TLR-FRAME approach, the stop condition was set up to perform 200 iterations of the evolutionary algorithm, where each iteration evaluated 120 model fragments. Therefore, for each test case, the approach evaluated a total of 24000 model fragments. However, TLR-LtoR approach only evaluates one model fragment for each iteration, so the stop condition was set up to perform 24000 iterations in order to evaluate the same number of model fragments.

7.4 Comparison and Measure

For each test case, each approach obtains a model fragment. This model fragment realizes the requirement to a greater or lesser extent. Therefore, to evaluate how well the model fragment realizes the requirement, this model fragment is compared against the correct solution. Since the oracle contains the correct solutions for each test case, we can compare the obtained solution to the correct solution. This comparison is performed through a confusion matrix.

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

- True Positive (TP): values that are predicted as true (in the obtained solution) and are true in the real scenario (the correct solution).
- False Positive (FP): values that are predicted as true (in the obtained solution) but are false in the real scenario (the correct solution).
- True Negative (TN): values that are predicted as false (in the obtained solution) and are false in the real scenario (the correct solution).
- False Negative (FN): values that are predicted as false (in the obtained solution) but are true in the real scenario (the correct solution).

Then, some performance measurements are derived from the values in the confusion matrix. Specifically, we create a report that includes four performance measurements (precision, recall, the F-measure, and the Matthews Correlation Coefficient) for the test case.

Precision and recall are commonly used in information retrieval for evaluating retrieval (classification) performance [155, 156, 157] In fact, the two traditional approaches under evaluation (TLR-Linguistic and TLR-IR) were evaluated in [123] and in [124, 125] respectively, obtaining the best results for recall and precision in Traceability Links Recovery on models.

Precision measures the proportion of elements from the obtained solution that are correct according to the correct solution and is defined as follows:

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

Recall measures the proportion of elements of the correct solution that are correctly retrieved by the obtained solution and is defined as follows:

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

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

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2TP + FP + FN}$$

However, precision, recall, and F-measure are typically used only to evaluate the skill of information retrieval systems, where there is only interest in the positive findings (i.e. the quality of the retrieved results) [157]. Indeed, none of these performance measurements correctly handle negative examples (TN). In the case of fragment retrieval, we do care about true negatives as the retrieval of model elements that are non-related to the query comes at en additional effort and cost by the engineers. The **MCC** is a correlation coefficient between the observed and predicted binary classifications that takes into account all of the observed values (TP, TN, FP, FN) and is defined as follows:

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

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

This process was repeated for all test cases and for all the approaches. Whenever an approach was tested out from a test cases, the obtained solution was compared through a confusion matrix and the previous measures (Recall, Precision, F-measure, and MCC) where used to evaluate the results.

7.5 Threats to validity

Finally, we use the classification of threats to validity of [158] to acknowledge the limitations of the evaluation. Following the [158] definitions for the experiment processes, the treatments are the different approaches: TLR-FRAME, TLR-Linguistic, TLR-IR, TLR-FNN, TLR-RNN, and TLR-LtoR. All the treatments were applied by a Lenovo with a processor Intel(R) Core(TM) i5-3210M @2.5GHz with 4GB of RAM and running Windows 10 Pro N 64 bits as the hosting Operative System (subject). Moreover, all the treatments were tested using all the test cases (objects).

Therefore, taking into account the treatments, the subject, and the objects, Table 7.1 presents the threats that can be applicable to our evaluation. In this table, the threats (column 2) are grouped by type (column 1) and defined briefly (column 3). In addition, the last column (column 4) of the table shows how the threats have been dealt with.

Table 7.1: Threats to Validity

Type of threat	**Threat**	**Due to**	**How we have dealt with it**
Conclusion Validity	Fishing	Researchers may influence the result by looking for a specific outcome	We avoided this threat by using all of the test cases for all of the approaches; none of the test cases were removed for any reason whatsoever.

Table 7.1: Threats to Validity

Type of threat	Threat	Due to	How we have dealt with it
	Reliability of measures	When you measure a phenomenon twice, the outcome shall be the same	We reduced this threat by using four measures: precision, recall, F-measure, and MCC, which are widely accepted in the software engineering research community. Moreover, as suggested by [121], several repetitions have been performed to obtain reliable results. Specifically, each test case was run 30 times for each approach.
	Lower statistical power	Sample size is not enough	We reduced this threat by using 20 different test cases and statistically analyzing the results.
	Reliability of treatment implementation	The implementation is not similar between different persons, applying the treatment, or between different occasions	We avoided this threat by tuning the parameters to maximize the performance of all of the approaches and to perform a fair evaluation.
Internal Validity	Instrumentation	Effect caused by the artifacts used for experiment execution	We avoided this threat by balancing the knowledge base, which is in the test cases. The knowledge base contains samples with high and low fitness values. We reduced this threat by validating the ontology, which is in the test cases. Different domain experts check that the ontology is well designed, so that it does not negatively affect to the evaluation.

Table 7.1: Threats to Validity

Type of threat	Threat	Due to	How we have dealt with it
	Influence of the domain	This threat appears when the outcomes depend on a specific domain	We reduced this threat by designing and developing the approach independently of the domain. Nevertheless, the experiment and its results should be replicated in other domains before assuring their generalization.
External Validity	Interaction of setting and treatment	This is the effect of not having the experimental setting or material representative of industrial practice	We avoided this threat by using the last version of the Eclipse Modeling Framework to perform the implementation. We reduced this threat by using formats that are frequently leveraged to specify all kinds of different software, for example MOF.
Construct validity	Interaction of testing and treatment	Subjects apply the metrics to the treatments	The experiment suffers from this threat since, the computer is responsible to apply the metrics. However, the metrics are automatically applied through the same program for the results of all the approaches. Therefore, this threat can be considered not applicable.
	Mono-operation bias	A single treatment or case can lead to bias	We avoided this threat by applying five different approaches to compare our approach. We avoided this threat by using 20 different test cases.

Table 7.1: Threats to Validity

Type of threat	Threat	Due to	How we have dealt with it
	Author bias	People that define the artifacts can subjectively influence the obtainment of the results that they are looking for	We avoided this threat using the documentation provided by our industrial partner. The test cases were prepared from this documentation by a domain expert who was not involved in the experiment and the research.

Chapter 8

RESULTS OF THE EVALUATION

Contents

8.1 Overview of the chapter

This chapter presents the results of the evaluation. Specifically, this chapter presents the reported results for each approach, the statistical analysis, the answers to the research questions, and the discussion.

8.2 Reported results

In Table 8.1, we outline the results, which are aggregated for each of the approaches. Each row shows the Precision, Recall, F-measure, and MCC values reported for each approach.

Table 8.1: Mean Values and Standard Deviations for Precision, Recall, F-Measure, and Matthews Correlation Coefficient (MCC) for the approaches

	Precision	Recall	F-Measure	MCC
TLR-FRAME	59.91 ± 33.39	78.95 ± 15.16	62.50 ± 27.76	0.64
TLR-Linguistic	37.38 ± 16.18	48.61 ± 19.78	40.41 ± 16.19	0.40
TLR-IR	18.09 ± 25.55	53.45 ± 38.70	21.69 ± 23.95	0.21
TLR-FNN	8.20 ± 0.10	100 ± 0.00	14.06 ± 0.14	-0.84
TLR-RNN	8.37 ± 0.09	100 ± 0.00	14.34 ± 0.14	-0.77
TLR-LtoR	13.01 ± 26.08	11.85 ± 18.24	10.27 ± 17.57	0.07

As the table shows, our FRAME approach achieves the best results for most performance indicators, providing a mean precision value of 59.91%, a recall value of 78.95%, a combined F-measure value of 62.50%, and a MCC value of 0.64. In contrast, the TLR-Linguistic approach, the TLR-IR approach, and the TLR-LtoR approach present worse results in all of the measurements: the TLR-Linguistic approach attains 37.38% precision, 48.61% recall, 40.41% F-measure, and 0.40 MCC; the TLR-IR approach achieves 18.09% precision, 53.45% recall, 21.69% F-measure, and 0.21 MCC; and the TLR-LtoR approach attains 13.01% precision, 11.85% recall, 10.27% F-measure, and 0.07 MCC. On the other hand, both the TLR-FNN approach and the TLR-RNN approach achieve the best results for recall, but they present the worst results for the rest of the indicators: the TLR-FNN attains 8.20% precision, 100% recall, 14.06% F-measure, and -0.84 MCC; and the TLR-RNN baseline achieves 8.37% precision, 100% recall, 14.34% F-measure, and -0.77 MCC.

8.3 Statistical Analysis

To properly compare the results of the different approaches, the data resulting from the empirical analysis was analyzed using statistical methods. Specifically, we first run a statistical test to state that there is no difference between the approaches (the null hypothesis H_0), or on the contrary, to state that the approaches differ (the alternative hypothesis H_1). Then, since the null hypothesis is rejected, the approaches are individually compared. An additional post hoc analysis is performed to determine if there are significant differences among the results of two specific approaches. Finally, we measured the *effect size* in order to assess whether an approach is statistically better than another and to assess the magnitude of the improvement.

8.3.1 Statistical Test

Since our data does not follow a normal distribution in general, our analysis required the use of nonparametric techniques. There are several tests for analyzing this kind of data; however, the Quade test is the most powerful one when working with real data [159]. In addition, according to Conover [160], the Quade test is the one that has shown the best results for a low number of approaches.

Table 8.2: Quade test statistic and $p - Values$

	Precision	Recall
p-Value	1.7×10^{-10}	2.20×10^{-16}
Statistic	14.41	35.27

Table 8.2 shows the Quade test statistic and $p - Values$ for recall and precision. The null hypothesis can be rejected taking into account the probability value, $p - Value$. The $p - Value$ obtains values between 0 and 1. The lower the $p - Value$ of a test, the more likely that the null hypothesis is false. It is accepted by the research community that a $p - Value$ under 0.05 is sta-

tistically significant [161], and so the hypothesis H_0 can be considered false. Therefore, since the $p - Values$ are smaller than 0.05, we rejected the null hypothesis. Consequently, we can state that there are differences among the six approaches.

8.3.2 Post Hoc Analysis

This kind of analysis performs a pair-wise comparison among the results of each approach, determining whether statistically significant differences exist among the results of a specific pair of approaches.

Table 8.3: Holm's Post Hoc $p - Values$

	Precision	Recall
TLR-FRAME vs TLR-Linguistic	8.4×10^{-03}	3.3×10^{-06}
TLR-FRAME vs TLR-IR	7.3×10^{-05}	0.041
TLR-FRAME vs TLR-FNN	1.9×10^{-07}	8.3×10^{-06}
TLR-FRAME vs TLR-RNN	1.9×10^{-07}	8.3×10^{-06}
TLR-FRAME vs TLR-LtoR	3.8×10^{-06}	2.7×10^{-07}
TLR-Linguistic vs TLR-IR	6.1×10^{-04}	0.49
TLR-Linguistic vs TLR-FNN	3.4×10^{-07}	1.0×10^{-07}
TLR-Linguistic vs TLR-RNN	3.4×10^{-07}	1.0×10^{-07}
TLR-Linguistic vs TLR-LtoR	2.5×10^{-03}	1.9×10^{-05}
TLR-IR vs TLR-FNN	0.04	4.2×10^{-06}
TLR-IR vs TLR-RNN	0.04	4.2×10^{-06}
TLR-IR vs TLR-LtoR	0.27	1.1×10^{-03}
TLR-FNN vs TLR-RNN	1.1×10^{-03}	0.0
TLR-FNN vs TLR-LtoR	0.97	2.9×10^{-08}
TLR-RNN vs TLR-LtoR	0.97	2.9×10^{-08}

Table 8.3 shows the $p-Values$ of Holm's post hoc analysis for each specific pair of approaches. Almost of all the $p-Values$ shown in this table are smaller than 0.05, except for some cases: the precision comparison between TLR-IR and TLR-LtoR, the precision comparison between TLR-FNN and TLR-LtoR, the precision comparison between TLR-RNN and TLR-LtoR, and the recall comparison between TLR-Linguistic and TLR-IR. Therefore, significant differences for one of the performance measurements were obtained in all of the comparisons.

8.3.3 Effect Size

Statistically significant differences can be obtained even if they are so small as to be of no practical value [161]. It is then important to assess whether an approach is statistically better than another and to assess the magnitude of the improvement. *Effect size* measures are needed to analyze this.

For a non-parametric effect size measure, we used Vargha and Delaney's $\hat{A}_{12}$ [162]. $\hat{A}_{12}$ measures the probability that running one approach yields higher values than running another approach. If the two approaches are equivalent, then $\hat{A}_{12}$ will be 0.5.

For example, $\hat{A}_{12} = 0.7$ means that we would obtain better results in 70% of the runs with the first of the pair of approaches that have been compared, and $\hat{A}_{12} = 0.3$ means that we would obtain better results in 70% of the runs with the second of the pair of approaches that have been compared. Thus, we have an $\hat{A}_{12}$ value for every pair of approaches.

Table 8.4 shows the values of the effect size statistics between every pair of approaches.

TLR-FRAME vs TLR-Linguistic: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FRAME will obtain better results than TLR-Linguistic in 70% of the cases for precision, and better recall values in 83% of the cases.

TLR-FRAME vs TLR-IR: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FRAME will obtain better results than TLR-IR in 85% of the cases for precision, and better recall values in 68% of the cases.

TLR-FRAME vs TLR-FNN: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FRAME will obtain better results in 93% of the

Table 8.4: $\hat{A}_{12}$ statistic for each pair of approaches

	Precision	Recall
TLR-FRAME vs TLR-Linguistic	0.70	0.83
TLR-FRAME vs TLR-IR	0.85	0.68
TLR-FRAME vs TLR-FNN	0.93	0.15
TLR-FRAME vs TLR-RNN	0.93	0.15
TLR-FRAME vs TLR-LtoR	0.89	0.97
TLR-Linguistic vs TLR-IR	0.81	0.45
TLR-Linguistic vs TLR-FNN	0.93	0.0
TLR-Linguistic vs TLR-RNN	0.93	0.0
TLR-Linguistic vs TLR-LtoR	0.86	0.91
TLR-IR vs TLR-FNN	0.55	0.13
TLR-IR vs TLR-RNN	0.54	0.13
TLR-IR vs TLR-LtoR	0.65	0.78
TLR-FNN vs TLR-RNN	0.48	0.5
TLR-FNN vs TLR-LtoR	0.67	1
TLR-RNN vs TLR-LtoR	0.68	1

cases for precision, while TLR-FNN will obtain better recall values in 85% of the case.

TLR-FRAME vs TLR-RNN: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FRAME will obtain better results in 93% of the cases for precision, while TLR-RNN will obtain better recall values in 85% of the case.

TLR-FRAME vs TLR-LtoR: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FRAME will obtain better results than TLR-LtoR in 89% of the cases for precision, and better recall values in 97% of the cases.

TLR-Linguistic vs TLR-IR: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-Linguistic will obtain better results in 81% of the cases for precision, while TLR-IR will obtain better recall values in 55% of the case.

TLR-Linguistic vs TLR-FNN: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-Linguistic will obtain better results in 93% of the cases for precision, while TLR-FNN will obtain better recall values in 100% of the case.

TLR-Linguistic vs TLR-RNN: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-Linguistic will obtain better results in 93% of the cases for precision, while TLR-RNN will obtain better recall values in 100% of the case.

TLR-Linguistic vs TLR-LtoR: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-Linguistic will obtain better results than TLR-LtoR in 86% of the cases for precision, and better recall values in 91% of the cases.

TLR-IR vs TLR-FNN: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-IR will obtain better results in 55% of the cases for precision, while TLR-FNN will obtain better recall values in 87% of the case.

TLR-IR vs TLR-RNN: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-IR will obtain better results in 54% of the cases for precision, while TLR-RNN will obtain better recall values in 87% of the case.

TLR-IR vs TLR-LtoR: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-IR will obtain better results than TLR-LtoR in 65% of the cases for precision, and better recall values in 78% of the cases.

TLR-FNN vs TLR-RNN: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FNN will obtain better results than TLR-RNN in 52% of the cases for precision, and better recall values in 50% of the cases.

TLR-FNN vs TLR-LtoR: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-FNN will obtain better results than TLR-LtoR in 67% of the cases for precision, and better recall values in 100% of the cases.

TLR-RNN vs TLR-LtoR: The $\hat{A}_{12}$ measure value indicates that, of the two approaches, TLR-RNN will obtain better results than TLR-LtoR in 68% of the cases for precision, and better recall values in 100% of the cases.

The obtained $\hat{A}_{12}$ values show that TLR-FRAME is superior to all of the approaches for precision. Moreover, TLR-FRAME is also superior to TLR-Linguistic, TLR-IR, and TLR-LtoR on recall, meaning that TLR-FRAME will obtain better results than these three approaches in most of the cases. Overall, these measurements confirm that, for recall and precision, TLR-FRAME outperforms the approaches (TLR-IR and TLR-Linguistic) that obtain the best results for TLR between requirements and models. Moreover, these measurements confirm that, for precision, TLR-FRAME outperforms the ML approaches (TLR-FNN and TLR-FNN) that have successfully been applied recently in Traceability Link Recovery. Finally, these measurements confirm that, for recall and precision, TLR-FRAME outperforms the approach (TLR-LtoR) that explores the search space by means of brute-force.

8.4 Research Questions for the Evaluation

In response to Evaluation-RQ1, we can affirmatively answer to this research question. It is feasible to retrieve model fragments in industrial domains using the FRAME approach presented in this dissertation. The FRAME approach was successfully used to retrieve the model fragments related to several requirements in an industrial domain like CAF.

In response to Evaluation-RQ2, the performance of FRAME approach on the industrial models of CAF is determined by the results of the evaluation. The FRAME approach achieved a mean precision value of 59.91%, a recall value of 78.95%, a combined F-measure value of 62.50%, and a MCC value of 0.64.

In response to Evaluation-RQ3, the FRAME approach can significantly outperform the results of the most common approaches for Traceability Link

Recovery. Taking into account the statistical analysis, the FRAME approach outperforms the most common approaches (TLR-IR and TLR-Linguistic) for recall and precision. Moreover, for precision, the FRAME approach outperforms the ML approaches (TLR-FNN and TLR-FNN) that have successfully been applied recently in Traceability Link Recovery. Furthermore, for recall and precision, the FRAME approach outperforms the TLR-LtoR approach, which indicates that the evolutionary algorithm is useful and necessary for our approach.

8.5 Discussion

In this section, we discuss what prerequisites are needed by each approach, what properties affect the results and limit the approaches. We also discuss why TLR-FRAME is less sensitive to tacit knowledge and vocabulary mismatch than the other approaches. These advantages lead to the better results of TLR-FRAME.

8.5.1 Prerequisites and Properties

Both our approach and the other approaches need some prerequisites to be applied. If one of their prerequisites is not satisfied, the approach would not be used in that domain. Table 8.5 shows what artifacts are needed to apply each approach.

Table 8.5: Required artifacts for each approach

		Approaches					
		TLR-Linguistic	TLR-IR	TLR-FNN	TLR-RNN	TLR-LtoR	TLR-FRAME
Artifacts	Models	X	X	X	X	X	X
	Requirements	X	X	X	X	X	X
	Knowledge Base			X	X	X	X
	Ontology			X	X	X	X
	Rules	X					

Table 8.5 shows that all of the approaches need models and requirements. Specifically, the models where requirements have to be located must conform to MOF (the OMG metalanguage for defining modeling languages) and that requirements must be provided using natural language. Moreover, all of the approaches that are based on Machine Learning (TLR-FRAME, TLR-LtoR,

TLR-FNN, and TLR-RNN) need a knowledge base to train and an ontology to encode the models and requirements. Finally, the TLR-Linguistic approach needs rules to identify relations between model elements and requirement words. The rules have to be defined following the guides and examples in [123].

Therefore, even though the training in TLR-FRAME is beneficial in avoiding to a large extent issues such as tacit knowledge and vocabulary mismatch, it is necessary to have access to a knowledge base and an ontology to perform the training. In industrial domains, especially long-living ones, where requirements and models have been stored for years, a knowledge base may be easily available. Also, thanks to the wide experience of the employees in companies of this kind, the main concepts and relations could be identified by experts in the domain. However, in other scenarios, such as when only the first product has been developed, TLR-FRAME cannot be applied.

In addition, even though we had all the necessary artifacts to apply our approach, the results may not be as good as possible. In fact, some properties of the artifacts have an impact on the results. For example, if there is not enough information in the knowledge base, TLR-FRAME would not train properly, so the results would be worse than expected. Table 8.6 shows the properties that we have identified in this work and that had an impact on the obtained results.

Table 8.6: Artifacts whose properties have an impact on the results

		Properties				
		Homogeneity	Completeness	Heterogeneity	Size	Volume
Artifacts	Models	X			X	
	Requirements	X	X			
	Knowledge Base		X	X	X	
	Ontology		X		X	
	Rules		X			X

The following paragraphs provide more details about the properties identified in Table 8.6:

- **Models** may be developed by several engineers and at different times, so the terms used to describe model elements may be different (e.g., pantograph and panto are two different terms used in our models to refer to the same concept: *pantograph*). Therefore, the first model property that affects to the results is homogeneity. The second one is the size, which has an impact on the result based on the understanding of the models [163].

- **Requirements** may be defined by different engineers and at different times, so the homogeneity of the requirements, like the homogeneity of the models, has an impact on the results. The results are also affected by the completeness of the requirements. Often, when requirements are written, part of the domain knowledge related to the requirements is not embodied in them because tacit knowledge about the domain is assumed to be known by all of the domain experts. Therefore, the requirements are more or less complete in accordance with how many assumptions are made by the engineers. In the end, requirements may lose part of the information that is required because of these assumptions.

- **Knowledge Base** contains the information necessary to train the classifier, so this information must be enough to train it. If the knowledge base only contains the information to recover the traces between one requirement and one model, the classifier may not learn how to recover the traces for other requirements or models. Therefore, including heterogeneity samples of traces in the knowledge base provides more complete information for the training. In addition, some ML techniques require a larger knowledge base than others to provide suitable results, so the technique must be selected based on the available knowledge base.

- **Ontology** is composed of the main concepts and relations of a domain. Therefore, if a relevant concept or relation is not present in the ontology, the encoding for the fitness function will not take it into account and the training may be incomplete, leading to worse results. For this reason, the first property to keep in mind for the ontology is completeness. Moreover, if the ontology contains unnecessary concepts or relations, the number of characteristics for the encoding would be greater and a great number of characteristics in the training step leads to overfitting. Therefore, we must also take into account the size of the ontology.

- **Rules** are defined by humans through the manual comparison of the models and requirements. Therefore, the completeness of the rules depends on how well engineers understand the models and requirements and how complex these models and requirements are. The volume of rules also affects the results. If only one rule is defined, the approach only recovers one type of model element, so the approach may need several rules.

However, a large number of rules does not guarantee the best results. Therefore, both completeness and volume must be taken in account.

8.5.2 Advantages of TLR-FRAME

This section discusses why TLR-FRAME achieves better results than the baselines regarding three aspects: tacit knowledge, vocabulary mismatch, and available documentation.

Tacit Knowledge

Often, when requirements are written, part of the domain knowledge related to the requirements is not embodied in them. The tacit knowledge about the domain is assumed to be known by all of the domain experts, so it is never formalized in writing. This behavior has been reported in previous works [164, 165]. For example, given the requirement: *At all stations, the doors are automatically opened*, the engineers understand that the doors have to be opened in all of the stations, without being requested by a passenger. However, this requirement also embodies tacit knowledge that is not written but is obvious to the domain engineers: *The train has doors on both sides, but only the doors on the side of the platform will be opened, while the doors on the side of the tracks will remain closed*, and *all of the doors on one side will be opened, except the driver's door in the cabin.*

The tacit knowledge is not reflected in the text of the requirements. This tacit knowledge is shared among the engineers that write the requirements and the engineers that read the requirements. Therefore, both the text of the requirements and tacit knowledge are used to build the models. As a result, the model contains elements that are related to text of the requirement, but the model also contains elements that are related to the tacit knowledge. However, since part of the knowledge is not reflected in the text of the requirement, recovering the most relevant model fragment for a requirement is complex.

Both TLR-IR and TLR-Linguistic depend, to a large extent, on the text of the requirement. TLR-IR evaluates the similarity between the requirement and the model fragment according to the co-occurrences of terms between the two. TLR-Linguistic evaluates the similarity between the requirement and the model fragment according to patterns that relate the terms in the requirement with the elements in the model fragment. In both cases, the lack of terms that is caused by the tacit knowledge makes it impossible to locate the elements from the model that are relevant to the requirement.

In contrast, TLR-FRAME is less sensitive to tacit knowledge due to training. In the training, the requirements of the knowledge base are linked to the model fragments of the knowledge base. Even though the text of requirements is inaccurate due to tacit knowledge, the linked model fragments are complete. Consequently, the classifier is not only trained from the text of the requirements, but also from the elements of the model fragments. Therefore, the classifier learns that certain elements of models are relevant to certain requirements even though these elements are not described properly in the text of the requirements. As TLR-FRAME depends, to a lesser extent, on the text of the requirement than TLR-IR and TLR-Linguistic, when the requirements have a lack of terms due to tacit knowledge, the results that are obtained through TLR-FRAME are better than the results obtained through TLR-IR and TLR-Linguistic.

Vocabulary Mismatch

Vocabulary mismatch is caused by the use of different terms to reference the same concept in the requirement and the model. In industrial environments, sometimes the engineer who is in charge of writing the requirement is not the same engineer assigned to building the model. Moreover, both the requirement and the model may be manipulated by different engineers.

Even though TLR-IR, TLR-Linguistic, and even TLR-FRAME, may use Natural Language Processing (NLP) to homogenize the terms between requirements and models, vocabulary mismatch continues to be an issue that must be taken into account. Since the in-house terms that are used in a specific domain or company are not known synonyms, these in-house terms may not be included in NLP, causing vocabulary mismatch. For example, the terms *PLC* and *system* may be recognized as synonyms, but the terms *PLC* and *COSMOS* are definitely not known to be synonyms because *COSMOS* is an in-house term that is used exclusively by our industrial partner to refer to *PLC*.

As in the tacit knowledge issue, TLR-IR and TLR-Linguistic are seriously affected by vocabulary mismatch because both of them depend, to a large extent, on the text of the requirements. If the terms that are used in the requirements and the terms that are used in the models are not known synonyms, they cannot be related, and therefore the requirement cannot be correctly related to the elements of the model. Therefore, the lack of awareness that is caused by vocabulary mismatch makes it impossible to locate the elements from the model that are relevant to the requirement.

In contrast, TLR-FRAME is less sensitive to vocabulary mismatch for the same reason described for the tacit knowledge issue. The evaluation of TLR-FRAME depends on the information provided by training. If the information that is extracted through the training indicates that a term of the requirement is related to a term of an element in the model, the classifier learns that both terms are related to each other even when they are not considered synonyms. Therefore, TLR-FRAME depends, to a lesser extent, on the synonyms than TLR-IR and TLR-Linguistic, which leads to our approach having better results than the these approaches.

Available Documentation

Since TLR-FNN and TLR-RNN are trained using the same knowledge base than TLR-FRAME, they should also be less sensitive to tacit knowledge and vocabulary mismatch. However, our knowledge base may be unsuitable for properly training a Neural Network. For example, in [59], the training set is composed of 45% of the 769,366 artifacts, so this training set contains about 423,151 feature vectors. However, our training set is composed of the encoding of the knowledge base that has 103 samples whose model fragments have around 15 elements. Therefore, since the ending is performed at the model-element level, the training set contains about 1545 (103 x 15) feature vectors.

Some works analyze the impact of the number of samples on the performance of the neural networks. The authors in [166, 167] suggest the use of a minimum of 10–30p samples for training, where p is the number of features vectors used. However, this rule is often universally enforced in remote sensing without questioning its relevance to the complexity of the specific problem [168]. In fact, in some domains, the best result are obtained with 2p or 4p samples for training [169, 168]. Therefore, a small knowledge base may be insufficient and a large knowledge base may introduce noise.

On the other hand, the knowledge base may also be affected by the vaporization problem [170]. In fact, some industrial companies do not store enough information to create a knowledge base with the necessary completeness and size. However, these domains also need to recover the traceability links, and our approach can be successfully used even if the knowledge base is small, as our evaluation proves.

Since TLR-LtoR is based on LtoR as TLR-FRAME also is, we might expect that TLR-LtoR will not have the problems described. However, TLR-LtoR obtained the worst results because the search space was too big, so the exploration of this search space randomly required many more iterations. Therefore, by evaluating the same number of model fragments using the two approaches, the TLR-FRAME obtained the best results thanks to the combination of the LtoR, which provides a successful evaluation of the model fragments, and the evolutionary algorithm, which allows the search space to be explored in an effective way.

PART V

CONCLUSION

Everything that has a beginning has an end.
THE MATRIX REVOLUTIONS

Chapter 9

Conclusion and Ongoing Research

Contents

9.1 Overview of the chapter

This chapter recapitulates the results presented so far and concludes the dissertation. First, we provides answers to the research questions formulated in the first chapter (Section 9.2). Then, the ongoing research is described (Section 9.3). Finally, we conclude the dissertation (Section 9.4).

9.2 Research Questions

Throughout this dissertation, we have been working toward answers for the research questions associated to the objectives defined in the first chapter (see Chapter 1).

9.2.1 Results of the Objective 1

The first objective (OB1) consists of reviewing the works related to the application of ML techniques for software maintenance tasks on models. Along Chapter 3, we answered the research question associated to OB1 through a preliminary systematic review (SR).

RQ1: What other approaches for software maintenance tasks on models are there?

- **SR-RQ1:** What kind of software artifacts is the most common target for Traceability Link Recovery, Bug Localization, or Feature Location?
- **SR-RQ2:** What are the most common ML techniques for Traceability Link Recovery, Bug Localization, Feature Location?

Answer to RQ1: To address RQ1, we analyzed the state-of-the-art through a preliminary systematic review. To do so, we used the guidelines proposed in [47, 48, 49]. The search shows that most research works, that apply ML techniques for software maintenance tasks, target source code instead of models. However, some works tackle Feature Location on models applying ML techniques [90, 91, 98, 99, 10]. One of these works applies decision and classification trees [98]. The remaining works apply genetic algorithms, which is the most common ML technique for Feature Location.

9.2.2 Results of the Objective 2

The second objective (OB2) of this dissertation consists of providing a ML-based approach to automatically perform fragment retrieval in software maintenance tasks on models, which is called FRAME. Along Chapters 4, 5, and 6, we answered the set of research questions associated to OB2:

RQ2: How to apply ML techniques on models?

Answer to RQ2: To address RQ2, we encode the model fragments and the queries into feature vectors. To do so, we design an ontological evolutionary encoding that turn the model fragments and the queries into the characteristic-value pairs of the feature vectors. Each characteristic correspond to a concept or a relation in an ontology, and each value is computed as the frequency of the concept or relation in a model fragment or a query. Results show that the ontological evolutionary encoding can be used to characterize the model fragments and the queries from a industrial domain like CAF. This allows to apply ML techniques for fragment retrieval on models.

RQ3: How to assess model fragments through ML techniques?

Answer to RQ3: To address RQ3, the fitness function has been designed so a ML technique trains a classifier able to assess new model fragments regarding a query. To do so, we focus on LtoR algorithms that are ML techniques oriented to rank objects. In the fitness function, a LtoR algorithm trains a classifier using the model fragments that have been manually retrieved for several queries. Comparing those model fragments and queries, the classifier learns if a model fragment is or not a good realization of a query. Then, the classifier can assess new model fragments based on the previous learning. Specifically, the classifier assigns a fitness value to each new model fragment. This fitness value indicates how well the model fragment realizes the query. Therefore, the new model fragments can be ranked according to their fitness values, in order to find the model fragment that better realizes the query. Results show that ML techniques can be applied to address the fragment retrieval in software maintenance tasks, as in the Traceability Link Recovery task tackled for our industrial partner. In addition, in [13], we tackled a different maintenance task (Feature Location) applying a preliminary version of the fitness function and we obtained successful results.

RQ4: How to extract model fragments from a model?

Answer to RQ4: To answer RQ4, we use an evolutionary algorithm to effectively extract model fragments from a model. To do so, the evolutionary algorithm is guided by the fitness function. The fitness function assigns to each model fragment a fitness value, which indicates how well a model fragment realizes the query. Then, these fitness values are exploited by the genetics operators to select, crossover, or mutate the best candidates for the query. Then, the evolved model fragments are assessed through the fitness function. Both the fitness function and the genetics operations are repeated until a stop condition is satisfied. When the stop condition is met, the evolutionary algorithm returns a set of model fragment ranked according to their fitness values. Therefore, thanks to the evolutionary algorithm, we do not need to assess all the model fragments from a model. Only the best candidates for the query are assessed. Results show the evolutionary algorithm can be used to address the extraction of model fragments from a model. In addition, in the evaluation of the approach, we compare our approach with the evolutionary algorithm and the approach using brute-force to extract the model fragments. Results show that the evolutionary algorithm allows to explore the search space in an effective way, significantly outperforming the results of the approach without the evolutionary algorithm.

9.2.3 Results of the Objective 3

The third and last objective (OB3) of validating the contribution of this research in an industrial context. Along Chapter 8, we answered the research question associated to OB3 through the evaluation of the FRAME approach in a real case.

RQ5: What results does the designed approach achieve in comparison to other approaches for software maintenance tasks on models?

Evaluation-RQ1: Is it feasible to retrieve model fragments in industrial domains using the FRAME approach presented so far?

Evaluation-RQ2: What is the performance of FRAME approach on industrial models?

Evaluation-RQ3: Can the FRAME approach outperform significantly the results of the most common approaches for Traceability Link Recovery?

Answer to RQ5: To answer RQ5, we performed an evaluation of the approach designed for OB2 (FRAME) and compared it with other five approaches. To do so, our industrial partner provided us with a real case, for Traceability Link Recovery in the train domain. We designed the evaluation taking into account the real case, the approaches and their settings, the measurements for the results, and the threats to validity. The results of the evaluation show that it is feasible to successfully retrieve model fragments in industrial domains using the FRAME approach. In fact, the FRAME achieves the best results for most performance indicators, providing a mean precision value of 59.91%, a recall value of 78.95%, a combined F-measure value of 62.50%, and a MCC value of 0.64. Furthermore, the statistical analysis shows that the FRAME is superior to all of the other approaches for precision and is superior to three of the five approaches for recall. In addition, we included a discussion about prerequisites for each approach, relevant properties for the results of the approaches, and three domain aspects identified during the evaluation: tacit knowledge, vocabulary mismatch, and available documentation.

9.3 Ongoing Research

The contributions presented in this dissertation are the results of an ongoing work that is currently being developed further. Specifically, the FRAME approach is being currently applied to other software maintenance tasks, such as Feature Location or Bug Localization. Moreover, some aspects of the FRAME approach are being currently further researched to increase the performance of the approach. This section presents some open research questions and the ongoing work that is being done to address them.

9.3.1 Feature Location and Bug Localization

In this dissertation, the presented approach (FRAME) is applied for fragment retrieval in Traceability Link Recovery, so the approach recovers the model fragment that better realizes a requirement. Similarly, the FRAME is designed to locate a model fragment for a feature or to locate a model fragment for a bug. However, the performance of the approach for these software maintenance tasks still remains to be seen.

In [13], the first results for Feature Location are promising. However, in that work, we used a preliminary version of the FRAME approach. Therefore, we still have to ensure that this promising results are achieved for the final approach. Moreover, given the differences between requirements, bugs, and features the ongoing work also has to consider to compare the results of the approach for Traceability Link Recovery, Bug Localization, and Feature Location, in order to identify the advantages and limitations of the approach for these software maintenance tasks.

In addition, the current evaluation has only taken into consideration the performance of the proposed approach. However, we hope to evaluate the benefits of the approach from the engineers' perspective (e.g. satisfaction, time reduction, or ease of understanding) regarding the three software maintenance tasks: Traceability Links Recovery, Bug Localization, and Feature Location.

9.3.2 Ontological Evolutionary Encoding: Granularity

In this dissertation, we proposed the ontological evolutionary encoding to encode model fragments and queries into feature vectors. Specifically, the encoding uses the concepts and the relations of a domain ontology as characteristics in the feature vectors. ER'17 provides evidence that by using the ontological evolutionary encoding, ML techniques are applicable to Software Engineering tasks such as traceability link recovery. In fact, the performance of the encoding in this work is determined by the recovery of traces between the requirements and the models with an average value of 90.47% in recall and 75.19% in precision. However, in some cases, this encoding is not enough.

Imagine that a query requires to open the doors on the left side of the train. The ontology identifies the term *Door* as concept, so it will be used to encode the model fragments and the queries. However, the term *Right* is discarded, because it is not a concept or a relation in the ontology. Therefore, the approach has no way to differentiate a door on the left side to a door on the right side, so any model element that is a *Door* can be considered correct. This has an impact on the performance of the approach.

To solve it, we are currently working on the extension of the ontology using properties associated to the concepts. In fact, in [14], the ontology was extended with 14 properties to mitigate the problem. However, taking into account all the concepts, the properties, and the relations of a domain could be drawback for ML, where a great number of characteristics leads to overfitting. Therefore, the granularity of the ontological evolutionary encoding (concept level or property level) requires a further research.

9.3.3 Fitness Function: Deep Learning

The fitness function in the FRAME approach was designed to apply a LtoR algorithm. LtoR algorithms were selected for our design mainly for two reasons:

1. The model fragments can be ranked, because LtoR algorithms assign integer values. For example, support vector machine (SVM) is a ML technique more known than LtoR algorithms, but this technique returns binary values. Therefore, the fitness value for each model fragment would be zero or one. If we have several values with one, we cannot know what model fragment better realizes the query.

2. The LtoR algorithm have obtained good results even in cases where the knowledge base contains few samples. In industrial domains, especially long-living ones, where queries and models have been stored for years, a knowledge base may be easily available. However, to manually retrieve model fragments consume high amounts of time and effort. Therefore, the knowledge base cannot be as big as we wanted. A knowledge base with thousands or hundreds of samples may not be available. For this reason, we decided to apply the ML techniques that mitigate this situation.

However, in view of the results obtained, we are currently working on the application of deep learning in the fitness function. Specifically, we want to know if a feedforward neural network can properly assess the model fragments in the fitness function. The feedforward neural networks allow to assign integer values, so our research question focuses on the size of the knowledge base. What should be the size of the knowledge base for the neural network to obtain good results?

9.3.4 Domain Ontology: Metamodel

In this dissertation, the ontological evolutionary encoding is based on a domain ontology to turn the model fragments and queries into feature vectors. However, this ontology is not always available. In industrial domains, especially long-living ones, thanks to the wide experience of the employees, the main concepts and relations could be identified by experts in the domain. However, another possibility to be explored is the use of the metamodel to characterize the model fragments and the queries, instead of a domain ontology.

A metamodel is a description of the language's abstract syntax since it defines: (i) a set of constructs selected for the purpose of performing a specific (set of) task(s) and, (ii) a set of well-formedness rules for combining these constructs in order to create grammatically valid models in the language [171]. Therefore, we are currently exploring the possibility of using the constructs and the rules of the metamodel to characterize the model fragments and the queries, instead of using the concepts and the relations of a domain ontology.

9.3.5 Knowledge Base: Size and Model Fragments

The knowledge base is one of the main artifacts to successfully apply ML techniques. In fact, the influence of the knowledge base on the ML-based approaches is one of the points discussed in Section 8.5. The neural networks seem to require a bigger knowledge base than the FRAME approach.

On the other hand, in [16], we identify three properties of the model fragments: density, dispersion, and multiplicity. Since the knowledge base contains model fragments, we believe that these properties may have an impact on the training of the classifier. For example, we wonder if the classifier could properly assess big model fragments (high density), when it is trained only with small model fragments (low density).

Therefore, we are currently working towards a deeper research of the knowledge base. A research that tackles not only the size of the knowledge base, but also the model fragments that comprise it.

In addition, we will also tackle the Reinforcement Learning as part of this study or as an additional research. Specifically, we would like to analyze the advantages and drawbacks of including the solutions provided by FRAME in the knowledge base. This would require a re-training of the classifier and have an impact in the size and the distribution of the knowledge base.

9.4 Concluding Remark

As a concluding remark, although there are some open research questions, the work presented in this dissertation has provided a step forward in terms of addressing the issue of software maintenance tasks on models through ML techniques. In particular, the work presented in this dissertation: (1) review the works related to the application of ML techniques for software maintenance tasks on models; (2) provide a ML-based approach to automatically perform fragment retrieval in software maintenance tasks on models; and (3) validate the proposed approach in an industrial context. Moreover, in regard with the impact of our work:

Workshops: Our work was first introduced in a scientific workshop (specifically in 5th International Workshop on Reverse Variability Engineering [13]).

Conferences: Our work has been presented at scientific venues (specifically in the 36th International Conference on Conceptual Modeling [9], in the 25th International Conference on Cooperative Information Systems [15], and in the 21st International Conference on Model Driven Engineering Languages and Systems [16]).

Journals: Our work has been published in an international journal (specifically in Journal of Systems and Software [14]).

Research Projects: has been contributed to local, national, and international research projects such as ACIF (Local Spanish grant), VARIAMOS, ALPS, and DataME (Spanish national research project), and REVaMP2 (an international ITEA 3 Call 2 project).

Industrial Scenarios: has been evaluated in an industrial scenario: CAF (a worldwide provider of railway solutions).

Bibliography

[1] Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk, and Andrea De Lucia. On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In *2010 IEEE 18th International Conference on Program Comprehension*, pages 68–71. IEEE, 2010.

[2] Anas Mahmoud, Nan Niu, and Songhua Xu. A Semantic Relatedness Approach for Traceability Link Recovery. In *2012 20th IEEE international conference on program comprehension (ICPC)*, pages 183–192. IEEE, 2012.

[3] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Feature Location in Source Code: a Taxonomy and Survey. *Journal of software: Evolution and Process*, 25(1):53–95, 2013.

[4] Julia Rubin and Marsha Chechik. A Survey of Feature Location Techniques. In *Domain Engineering*, pages 29–58. Springer, 2013.

[5] Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, and Vaclav Rajlich. Feature Location via Information Retrieval Based Filtering of a Single Scenario Execution Trace. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, pages 234–243. ACM, 2007.

[6] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017.

[7] Francisca Pérez, Raúl Lapeña, Jaime Font, and Carlos Cetina. Fragment Retrieval on Models for Model Maintenance: Applying a Multi-objective Perspective to an Industrial Case Study. *Information and Software Technology*, 103:188–201, 2018.

[8] Francisca Pérez, Jaime Font, Lorena Arcega, and Carlos Cetina. Collaborative feature location in models through automatic query expansion. *Automated Software Engineering*, 26(1):161–202, 2019.

[9] Ana C Marcén, Francisca Pérez, and Carlos Cetina. Ontological Evolutionary Encoding to Bridge Machine Learning and Conceptual Models:

Approach and Industrial Evaluation. In *International Conference on Conceptual Modeling*, pages 491–505. Springer, 2017.

[10] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. Achieving feature location in families of models through the use of search-based software engineering. *IEEE Transactions on Evolutionary Computation*, 22(3):363–377, 2017.

[11] Lorena Arcega, Jaime Font, Øystein Haugen, and Carlos Cetina. An approach for bug localization in models using two levels: model and metamodel. *Software and Systems Modeling*, 18(6):3551–3576, 2019.

[12] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.

[13] Ana C Marcén, Jaime Font, Óscar Pastor, and Carlos Cetina. Towards Feature Location in Models through a Learning to Rank Approach. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*, pages 57–64, 2017.

[14] Ana C Marcén, Raúl Lapeña, Óscar Pastor, and Carlos Cetina. Traceability Link Recovery between Requirements and Models using an Evolutionary Algorithm Guided by a Learning to Rank Algorithm: Train Control and Management Case. *Journal of Systems and Software*, page 110519, 2020.

[15] Francisca Pérez, Ana Cristina Marcén, Raúl Lapeña, and Carlos Cetina. Introducing Collaboration for Locating Features in Models: Approach and Industrial Evaluation. In *On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I*, volume 10573 of *Lecture Notes in Computer Science*, pages 114–131. Springer, 2017.

[16] Manuel Ballarín, Ana C Marcén, Vicente Pelechano, and Carlos Cetina. Measures to report the Location Problem of Model Fragment Location. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 189–199, New York, USA, 2018. ACM.

[17] Ilhem Boussaïd, Patrick Siarry, and Mohamed Ahmed-Nacer. A survey on search-based model-driven engineering. *Automated Software Engineering*, 24(2):233–294, 2017.

[18] Oscar Pastor and Juan Carlos Molina. *Model-driven architecture in practice: a software production environment based on conceptual modeling.* Springer Science & Business Media, 2007.

[19] Stephen J Mellor, Tony Clark, and Takao Futagami. Model-driven development: guest editors' introduction. *IEEE software*, 20(5):14–18, 2003.

[20] Jochen Ludewig. Models in software engineering–an introduction. *Software and Systems Modeling*, 2(1):5–14, 2003.

[21] Jean-Marie Favre. Megamodelling and etymology. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[22] Thomas Kühne. Matters of (meta-) modeling. *Software & Systems Modeling*, 5(4):369–385, 2006.

[23] Alberto Rodrigues Da Silva. Model-driven engineering: A survey supported by the unified conceptual model. *Computer Languages, Systems & Structures*, 43:139–155, 2015.

[24] David W Embley, Stephen W Liddle, and Oscar Pastor. Conceptual-model programming: a manifesto. In *Handbook of Conceptual Modeling*, pages 3–16. Springer, 2011.

[25] Aditya Agrawal, Tihamer Levendovszky, Jon Sprinkle, Feng Shi, and Gabor Karsai. Generative programming via graph transformations in the model-driven architecture. In *Workshop on Generative Techniques in the Context of Model Driven Architecture, OOPSLA*, 2002.

[26] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices*, 35(6):26–36, 2000.

[27] Douglas T Ross. Origins of the apt language for automatically programmed tools. In *History of Programming Languages*, pages 279–338. 1978.

[28] J.W. Backus. The Syntax and Semantics of the Proposed International Algebraic Language of the Zurich ACM-GAMM Conference. In *Proceedings of the International Conference on Information Processing, UNESCO, Paris, 1959.*, page 125–132, 1960.

[29] Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[30] Jose E Rivera, Francisco Duran, and Antonio Vallecillo. A graphical approach for modeling time-dependent behavior of dsls. In *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 51–55. IEEE, 2009.

[31] José M Conejero, Eduardo Figueiredo, Alessandro Garcia, Juan Hernández, and Elena Jurado. On the relationship of concern metrics and requirements maintainability. *Information and Software Technology*, 54(2):212–238, 2012.

[32] Richard J Turver and Malcolm Munro. An early impact analysis technique for software maintenance. *Journal of Software Maintenance: Research and Practice*, 6(1):35–52, 1994.

[33] Robert L Glass. Frequently forgotten fundamental facts about software engineering. *IEEE software*, (3):112–110, 2001.

[34] Andrea De Lucia, Rocco Oliveto, Francesco Zurolo, and Massimiliano Di Penta. Improving comprehensibility of source code via traceability information: a controlled experiment. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pages 317–326. IEEE, 2006.

[35] Orlena CZ Gotel and CW Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101. IEEE, 1994.

[36] Laura Moreno, Wathsala Bandara, Sonia Haiduc, and Andrian Marcus. On the relationship between the vocabulary of bug reports and source code. In *2013 IEEE International Conference on Software Maintenance*, pages 452–455. IEEE, 2013.

[37] Nasir Ali, Aminata Sabane, Yann-Gael Gueheneuc, and Giuliano Antoniol. Improving bug location using binary class relationships. In *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*, pages 174–183. IEEE, 2012.

[38] Bogdan Dit, Latifa Guerrouj, Denys Poshyvanyk, and Giuliano Antoniol. Can better identifier splitting techniques help feature location? In *2011 IEEE 19th International Conference on Program Comprehension*, pages 11–20. IEEE, 2011.

[39] Kwanwoo Lee, Kyo C Kang, and Jaejoon Lee. Concepts and Guidelines of Feature Modeling for Product Line Software Engineering. In

International Conference on Software Reuse, pages 62–77, Verlag Berlin Heidelberg, 2002. Springer.

[40] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, DTIC Document, 1990.

[41] Wesley KG Assunção, Roberto E Lopez-Herrejon, Lukas Linsbauer, Silvia R Vergilio, and Alexander Egyed. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering*, 22(6):2972–3016, 2017.

[42] Jabier Martinez, Nicolas Ordoñez, Xhevahire Tërnava, Tewfik Ziadi, Jairo Aponte, Eduardo Figueiredo, and Marco Tulio Valente. Feature location benchmark with argouml spl. In *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*, pages 257–263, 2018.

[43] Iqbal Muhammad and Zhu Yan. Supervised machine learning approaches: A survey. *ICTACT Journal on Soft Computing*, 5(3), 2015.

[44] Lei Wang. Discovering phase transitions with unsupervised learning. *Physical Review B*, 94(19):195105, 2016.

[45] Shalika Walker, Waqas Khan, Katarina Katic, Wim Maassen, and Wim Zeiler. Accuracy of different machine learning algorithms and added-value of predicting aggregated-level energy performance of commercial buildings. *Energy and Buildings*, 209:109705, 2020.

[46] Girish Chandrashekar and Ferat Sahin. A Survey on Feature Selection Methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[47] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. 2007.

[48] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, pages 1–10, 2008.

[49] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[50] C. Mills and S. Haiduc. A machine learning approach for determining the validity of traceability links. pages 121–123, 2017. cited By 5.

[51] C. Mills, J. Escobar-Avila, and S. Haiduc. Automatic traceability maintenance via machine learning classification. pages 369–380, 2018. cited By 2.

[52] C. Mills. Automating traceability link recovery through classification. volume Part F130154, pages 1068–1070, 2017. cited By 2.

[53] D. Falessi, M. Di Penta, G. Canfora, and G. Cantone. Estimating the number of remaining links in traceability recovery. *Empirical Software Engineering*, 22(3):996–1027, 2017. cited By 9.

[54] A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshynanyk, and A. De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. pages 522–531, 2013. cited By 166.

[55] A. Ghannem, M.S. Hamdi, M. Kessentini, and H.H. Ammar. Search-based requirements traceability recovery. *Lecture Notes in Networks and Systems*, 15:156–171, 2018. cited By 0.

[56] J. David, M. Koegel, H. Naughton, and J. Helming. Traceability rearmed. volume 1, pages 340–348, 2009. cited By 4.

[57] Chuan Duan and Jane Cleland-Huang. Clustering Support for Automated Tracing. In *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, 2007.

[58] Yu Zhao, Tarannum S Zaman, Tingting Yu, and Jane Huffman Hayes. Using Deep Learning to Improve the Accuracy of Requirements to Code Traceability. *Grand Challenges of Traceability: The Next Ten Years*, page 22, 2017.

[59] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. Semantically Enhanced Software Traceability using Deep Learning Techniques. In *Software Engineering (ICSE), 2017 IEEE/ACM 39th International Conference on*, pages 3–14. IEEE, 2017.

[60] D. Distante and S. Faralli. A two-phase bug localization approach based on multi-layer perceptrons and distributional features. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11619 LNCS:518–532, 2019. cited By 0.

[61] M. Rath, D. Lo, and P. Mäder. Analyzing requirements and traceability information to improve bug localization. pages 442–453, 2018. cited By 8.

[62] R. Malhotra, S. Aggarwal, R. Girdhar, and R. Chugh. Bug localization in software using nsga-ii. pages 428–433, 2018. cited By 0.

[63] A.H. Moin and M. Khansari. Bug localization using revision log analysis and open bug repository text categorization. *IFIP Advances in Information and Communication Technology*, 319 AICT:188–199, 2010. cited By 5.

[64] A.N. Lam, A.T. Nguyen, H.A. Nguyen, and T.N. Nguyen. Bug localization with combination of deep learning and information retrieval. pages 218–229, 2017. cited By 39.

[65] Y. Xiao, J. Keung, Q. Mi, and K.E. Bennin. Bug localization with semantic and structural features using convolutional neural network and cascade forest. volume Part F137700, 2018. cited By 2.

[66] A.N. Lam, A.T. Nguyen, H.A. Nguyen, and T.N. Nguyen. Combining deep learning with information retrieval to localize buggy files for bug reports. pages 476–481, 2016. cited By 73.

[67] S. Wang, D. Lo, and J. Lawall. Compositional vector space models for improved bug localization. pages 171–180, 2014. cited By 38.

[68] G. Liu, Y. Lu, K. Shi, J. Chang, and X. Wei. Convolutional neural networks-based locating relevant buggy code files for bug reports affected by data imbalance. *IEEE Access*, 7:131304–131316, 2019. cited By 0.

[69] H. Ruan, B. Chen, X. Peng, and W. Zhao. Deeplink: Recovering issue-commit links based on deep learning. *Journal of Systems and Software*, 158, 2019. cited By 1.

[70] Z. Mousavian, M. Vahidi-Asl, and S. Parsa. Finding software fault relevant subgraphs a new graph mining approach for software debugging. pages 000908–000911, 2011. cited By 0.

[71] Y. Xiao, J. Keung, Q. Mi, and K.E. Bennin. Improving bug localization with an enhanced convolutional neural network. volume 2017-December, pages 338–347, 2018. cited By 10.

[72] Y. Xiao and J. Keung. Improving bug localization with character-level convolutional neural network and recurrent neural network. volume 2018-December, pages 703–704, 2018. cited By 2.

[73] Y. Xiao, J. Keung, K.E. Bennin, and Q. Mi. Improving bug localization with word embedding and enhanced convolutional neural networks. *Information and Software Technology*, 105:17–29, 2019. cited By 4.

[74] N. Safdari, H. Alrubaye, W. Aljedaani, B. Baez Baez, A. Distasi, and M.W. Mkaouer. Learning to rank faulty source files for dependent bug reports. volume 10989, 2019. cited By 0.

[75] Y. Xiao, J. Keung, K.E. Bennin, and Q. Mi. Machine translation-based bug localization technique for bridging lexical gap. *Information and Software Technology*, 99:58–61, 2018. cited By 9.

[76] S. Polisetty, A. Miranskyy, and A. Başar. On usefulness of the deep-learning-based bug localization models to practitioners. pages 16–25, 2019. cited By 0.

[77] Mai Alduailij and Mona Al-Duailej. Performance evaluation of information retrieval models in bug localization on the method level. In *2015 International Conference on Collaboration Technologies and Systems (CTS)*, pages 305–313. IEEE, 2015.

[78] A. Dutta, R. Sahay, P. Mitra, and R. Mall. Predicate proximity in failure: An mlp based fault localization approach. volume 2019-October, pages 936–941, 2019. cited By 1.

[79] Claire Le Goues, Westley Weimer, and Stephanie Forrest. Representations and operators for improving evolutionary software repair. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, pages 959–966, 2012.

[80] Z. Mousavian, M. Vahidi-Asl, and S. Parsa. Scalable graph analyzing approach for software fault-localization. pages 15–21, 2011. cited By 3.

[81] Y. Uneno, O. Mizuno, and E.-H. Choi. Using a distributed representation of words in localizing relevant files for bug reports. pages 183–190, 2016. cited By 8.

[82] L. Jiang and Z. Su. Context-aware statistical debugging: From bug predictors to faulty control flow paths. pages 184–193, 2007. cited By 77.

[83] I.-X. Chen, C.-Z. Yang, H. Jaygarl, and P.-J. Wu. Information retrieval on bug locations by learning co-located bug report clusters. pages 801–802, 2008. cited By 3.

[84] Xin Ye, Razvan Bunescu, and Chang Liu. Learning to Rank Relevant Files for Bug Reports using Domain Knowledge. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 689–699. ACM, 2014.

[85] Tien-Duy B. Le, David Lo, Claire Le Goues, and Lars Grunske. A Learning-to-Rank Based Fault Localization Approach using Likely Invariants. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pages 177–188. ACM, 2016.

[86] A.S.A. AL-Jumaili, H.K. Tayyeh, and R.J. Kadhem. A backpropagation neural network for splitting identifiers. *Journal of Computer Science*, 14(10):1412–1419, 2018. cited By 0.

[87] M. Abdelkader, M. Malki, and S.M. Benslimane. A heuristic approach to locate candidate web service in legacy software. *International Journal of Computer Applications in Technology*, 47(2-3):152–161, 2013. cited By 4.

[88] N. Balaji, S. Lakshmi, M. Anand, M. Anbarasan, and P. Mathiyalagan. An efficient scheme for secure feature location using data fusion and data mining in internet of things environment. *Software - Practice and Experience*, 2020. cited By 0.

[89] B. Dit. Configuring and assembling information retrieval based solutions for software engineering tasks. pages 641–646, 2017. cited By 3.

[90] J. Font, L. Arcega, Ø. Haugen, and C. Cetina. Feature location in model-based software product lines through a genetic algorithm. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9679:39–54, 2016. cited By 15.

[91] J. Font, L. Arcega, Ø. Haugen, and C. Cetina. Feature location in models through a genetic algorithm driven by information retrieval techniques. pages 272–282, 2016. cited By 16.

[92] C. Ziftci and I. Krüger. Feature location using data mining on existing test-cases. pages 155–164, 2012. cited By 2.

[93] A.C.E. Silva and M.D.A. Maia. Improving feature location accuracy via paragraph vector tuning. *Information and Software Technology*, 116, 2019. cited By 0.

[94] M. Abdelkader, M. Mimoun, and S.M. Benslimane. Locating candidate web service in legacy software: A search based approach. 2012. cited By 1.

[95] H. Abukwaik, A. Burger, B.K. Andam, and T. Berger. Semi-automated feature traceability with embedded annotations. pages 529–533, 2018. cited By 8.

[96] S.W. Hwang, Y.S. Lee, and Y.K. Nam. System for extracting domain topic using link analysis and searching for relevant features. *Journal of Ambient Intelligence and Humanized Computing*, 2018. cited By 1; Article in Press.

[97] A.S.B. Rani and A.R.N.B. Kamal. Text mining to concept mining: Leads feature location in software system. 2018. cited By 0.

[98] C. Mills. Towards the automatic classification of traceability links. pages 1018–1021, 2017. cited By 3.

[99] Carlos Cetina, Jaime Font, Lorena Arcega, and Francisca Pérez. Improving Feature Location in Long-Living Model-Based Product Families Designed with Sustainability Goals. *Journal of Systems and Software*, 134:261–278, 2017.

[100] Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. Feature model extraction from large collections of informal product descriptions. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 290–300, 2013.

[101] Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, and Christophe Dony. Feature location in a collection of product variants: Combining information retrieval and hierarchical clustering. In *SEKE: Software Engineering and Knowledge Engineering*, pages 426–430, 2014.

[102] Sana Ben Nasr, Guillaume Bécan, Mathieu Acher, João Bosco Ferreira Filho, Nicolas Sannier, Benoit Baudry, and Jean-Marc Davril. Automated extraction of product comparison matrices from informal product descriptions. *Journal of Systems and Software*, 124:82–103, 2017.

[103] Monica Bianchini, Marco Maggini, and Lakhmi C Jain. *Handbook on neural information processing.* Springer, 2013.

[104] Raúl Lapeña, Jaime Font, Óscar Pastor, and Carlos Cetina. Analyzing the Impact of Natural Language Processing over Feature Location in Models. *ACM SIGPLAN Notices*, 52(12):63–76, 2017.

[105] Peyman Beyranvand, Cavit Fatih Kucuktezcan, Zehra Cataltepe, and Veysel Murat Istemihan Genc. A Novel Feature Selection Method for the Dynamic Security Assessment of Power Systems Based on Multi-Layer Perceptrons. *International Journal of Intelligent Systems and Applications in Engineering*, 6(1):53–58, 2018.

[106] Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. Feature selection in machine learning: A new perspective. *Neurocomputing*, 300:70–79, 2018.

[107] Zhi-Jun Lu, Qian Xiang, Yong-mei Wu, and Jun Gu. Application of Support Vector Machine and Genetic Algorithm Optimization for Quality Prediction within Complex Industrial Process. In *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*, pages 98–103. IEEE, 2015.

[108] Asaf Shabtai, Robert Moskovitch, Yuval Elovici, and Chanan Glezer. Detection of Malicious Code by Applying Machine Learning Classifiers on Static Features: A State-of-the-Art Survey. *information security technical report*, 14(1):16–29, 2009.

[109] Haitham Ahmed Jamil. Feature selection and machine learning classification for live p2p traffic. 2019.

[110] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[111] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[112] M Furkan Kıraç, Barış Aktemur, and Hasan Sözer. VISOR: A Fast Image Processing Pipeline with Scaling and Translation Invariance for Test Oracle Automation of Visual Output Systems. *Journal of Systems and Software*, 136:266–277, 2018.

[113] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-Validation. In *Encyclopedia of database systems*, pages 532–538. Springer, 2009.

[114] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. A General Software Defect-Proneness Prediction Framework. *IEEE Transactions on Software Engineering*, 37(3):356–370, 2011.

[115] Alexandre H Hirzel, Gwenaëlle Le Lay, Véronique Helfer, Christophe Randin, and Antoine Guisan. Evaluating the Ability of Habitat Suitability Models to Predict Species Presences. *ecological modelling*, 199(2):142–152, 2006.

[116] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. Feature Location in Models Through a Genetic Algorithm Driven by Information Retrieval Techniques. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, MODELS '16, pages 272–282. ACM, 2016.

[117] Michael Affenzeller, Stephan M. Winkler, Stefan Wagner, and Andreas Beham. *Genetic Algorithms and Genetic Programming - Modern Concepts and Practical Applications.* CRC Press, 2009.

[118] Lawrence Davis. Handbook of Genetic Algorithms. 1991.

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

[120] Hamzeh Eyal Salman, Abdelhak Seriai, and Christophe Dony. 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*, pages 426–430, 2014.

[121] Andrea Arcuri and Gordon Fraser. Parameter Tuning or Default Values? An Empirical Investigation in Search-Based Software Engineering. *Empirical Software Engineering*, 18(3):594–623, 2013.

[122] Stefan Winkler and Jens Pilgrim. A Survey of Traceability in Requirements Engineering and Model-Driven Development. *Software and Systems Modeling (SoSyM)*, 9(4):529–565, 2010.

[123] George Spanoudakis, Andrea Zisman, Elena Pérez-Minana, and Paul Krause. Rule-Based Generation of Requirements Traceability Relations. *Journal of Systems and Software*, 72(2):105–127, 2004.

[124] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Enhancing an Artefact Management System with Traceability Recovery Features. In *Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 306–315. IEEE, 2004.

[125] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering Traceability Links in Software Artifact Management Systems using Information Retrieval Methods. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 16(4):13, 2007.

[126] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar. Scalable Product Line Configuration: A Straw to Break the Camel's Back. In *IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*, pages 465–474, 2013.

[127] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. An Efficient Boosting Algorithm for Combining Preferences. *Journal of machine learning research*, 4(Nov):933–969, 2003.

[128] Sérgio D Canuto, Fabiano M Belém, Jussara M Almeida, and Marcos A Gonçalves. A Comparative Study of Learning-to-Rank Techniques for Tag Recommendation. *Journal of Information and Data Management*, 4(3):453, 2013.

[129] Zherui Cao, Yuan Tian, Tien-Duy B Le, and David Lo. Rule-Based Specification Mining Leveraging Learning to Rank. *Automated Software Engineering*, pages 1–30, 2018.

[130] Zhi-Hua Zhou and Ji Feng. Deep Forest: Towards an Alternative to Deep Neural Networks. *arXiv preprint arXiv:1702.08835*, 2017.

[131] Jialei Wang, Peilin Zhao, Steven CH Hoi, and Rong Jin. Online Feature Selection and its Applications. *IEEE Transactions on Knowledge and Data Engineering*, 26(3):698–710, 2014.

[132] D. Dyer. The Watchmaker Framework for Evolutionary Computation (Evolutionary/Genetic Algorithms for Java). `http://watchmaker.uncommons.org/`, 2016. [Online; accessed 7-April-2016].

[133] Van Dang. The Lemur Project - Wiki - RankLib. `http://sourceforge.net/p/lemur/wiki/RankLib/`, 2013. [Online; accessed April-2017].

[134] Geoffrey Leech, Roger Garside, and Michael Bryant. CLAWS4: the Tagging of the British National Corpus. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, pages 622–628. Association for Computational Linguistics, 1994.

[135] The Stanford Natural Language Processing Group. `https://nlp.stanford.edu/software/tagger.shtml`, 2017. [Online; accessed 18-May-2017].

[136] William B Frakes and Ricardo Baeza-Yates. Information Retrieval: Data Structures and Algorithms. 1992.

[137] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to Information Retrieval*, volume 1. Cambridge University Press, 2008.

[138] Gerard Salton and Michael J McGill. Introduction to Modern Information Retrieval. 1986.

[139] De Lucia et al. Information Retrieval Models for Recovering Traceability Links between Code and Documentation. In *Proceedings of the International Conference on Software Maintenance*, pages 40–49. IEEE, 2000.

[140] Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk, and Andrea De Lucia. On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. In *18th International Conference on Program Comprehension*, pages 68–71. IEEE, 2010.

[141] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering Traceability Links between Code and Documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[142] Andrian Marcus and Jonathan I Maletic. Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing. In *Proceedings of the 25th International Conference on Software Engineering*, pages 125–135. IEEE, 2003.

[143] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Can Information Retrieval Techniques effectively support Traceability Link Recovery? In *14th IEEE International Conference on Program Comprehension*, pages 307–316. IEEE, 2006.

[144] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An Introduction to Latent Semantic Analysis. *Discourse Processes*, 25(2-3):259–284, 1998.

[145] Farid Meziane, Nikos Athanasakis, and Sophia Ananiadou. Generating Natural Language Specifications from UML Class Diagrams. *Requirements Engineering*, 13(1):1–18, 2008.

[146] Peter Abeles. Efficient Java Matrix Library. `http://ejml.org/`, 2017. [Online; accessed 12-April-2017].

[147] Simon Haykin. *Neural Networks: a Comprehensive Foundation.* Prentice Hall PTR, 1994.

[148] Andries P. Engelbrecht. *Computational Intelligence: An Introduction.* Wiley Publishing, 2nd edition, 2007.

[149] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer Feedforward Networks are Universal Approximators. *Neural networks*, 2(5):359–366, 1989.

[150] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing Neural Networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[151] D Team et al. Deeplearning4j: Open-source Distributed Deep Learning for the JVM. *Apache Software Foundation License*, 2, 2016.

[152] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. Deep Captioning with Multimodal Recurrent Neural Networks (M-RNN). *arXiv preprint arXiv:1412.6632*, 2014.

[153] Chahes Chopra, Shivam Sinha, Shubham Jaroli, Anupam Shukla, and Saumil Maheshwari. Recurrent Neural Networks with Non-Sequential Data to Predict Hospital Readmission of Diabetic Patients. In *Proceedings of the 2017 International Conference on Computational Biology and Bioinformatics*, pages 18–23. ACM, 2017.

[154] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-level Performance on Imagenet Classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[155] David D Lewis. Representation quality in text classification: An introduction and experiment. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*, 1990.

[156] David D Lewis. Evaluating text categorization i. In *Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991*, 1991.

[157] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[158] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering.* Springer Science & Business Media, 2012.

[159] Salvador García, Alberto Fernández, Julián Luengo, and Francisco Herrera. 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, 2010.

[160] WJ Conover. Practical Nonparametric Statistics, 3rd edn Wiley. *New York*, pages 250–257, 1999.

[161] Andrea Arcuri and Lionel Briand. A Hitchhiker's Guide to Statistical Tests for Assessing Randomized Algorithms in Software Engineering. *Software Testing, Verification and Reliability*, 24(3):219–250, 2014.

[162] András Vargha and Harold D Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[163] Harald Störrle. On the Impact of Layout Quality to Understanding UML Diagrams: Size Matters. In *International Conference on Model Driven Engineering Languages and Systems*, pages 518–534. Springer, 2014.

[164] Ioana Rus and Mikael Lindvall. Knowledge Management in Software Engineering. *IEEE Software*, 19(3):26, 2002.

[165] Andrew Stone and Peter Sawyer. Using Pre-Requirements Tracing to Investigate Requirements based on Tacit Knowledge. In *ICSOFT (1)*, pages 139–144, 2006.

[166] Paul M Mather and Magaly Koch. *Computer Processing of Remotely-Sensed Images: an Introduction.* John Wiley & Sons, 2011.

[167] Jim Piper. Variability and Bias in Experimentally Measured Classifier Error Rates. *Pattern Recognition Letters*, 13(10):685–692, 1992.

[168] Thomas G Van Niel, Tim R McVicar, and Bisun Datt. On the Relationship between Training Sample Size and Data Dimensionality: Monte Carlo Analysis of Broadband Multi-Temporal Classification. *Remote Sensing of Environment*, 98(4):468–480, 2005.

[169] Steven Walczak and Narciso Cerpa. Heuristic Principles for the Design of Artificial Neural Networks. *Information and software technology*, 41(2):107–117, 1999.

[170] Jan Salvador van der Ven, Anton GJ Jansen, Jos AG Nijhuis, and Jan Bosch. Design Decisions: The Bridge between Rationale and Architecture. In *Rationale management in software engineering*, pages 329–348. Springer, 2006.

[171] Giancarlo Guizzardi. On ontology, ontologies, conceptualizations, modeling languages, and (meta) models. *Frontiers in artificial intelligence and applications*, 155:18, 2007.

PART VI

PUBLICATIONS

Who knows? Have patience. Go where you must go, and hope!

THE LORD OF THE RINGS

Chapter 10

PUBLICATIONS

Contents

10.1 REVE'17 Paper (First page)

Title: Towards Feature Location in Models through a Learning to Rank Approach.

Authors: Ana Cristina Marcén, Jaime Font, Oscar Pastor, and Carlos Cetina.

Proceedings: Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, Volume B

Location: Sevilla, Spain - September 25-29, 2017

Publisher: ACM

Pages: 57–64

DOI: https://doi.org/10.1145/3109729.3109734

Towards Feature Location in Models through a Learning to Rank Approach

Ana C. Marcén[*]
Jaime Font
acmarcen@usj.es
jfont@usj.es
SVIT Research Group, Universidad San Jorge
Autovía A-23 Zaragoza-Huesca Km.299
Zaragoza, Spain 50830

Óscar Pastor
opastor@pros.upv.es
Centro de Investigación en Métodos de Producción de Software,
Universitat Politècnica de València
Camino de Vera, s/n
Valencia, Spain 46022

Carlos Cetina
ccetina@usj.es
SVIT Research Group, Universidad San Jorge
Autovía A-23 Zaragoza-Huesca Km.299
Zaragoza, Spain 50830

ABSTRACT

In this work, we propose a feature location approach to discover software artifacts that implement the feature functionality in a model. Given a model and a feature description, model fragments extracted from the model and the feature description are encoded based on a domain ontology. Then, a Learning to Rank algorithm is used to train a classifier that is based on the model fragments and feature description encoded. Finally, the classifier assesses the similarity between a population of model fragments and the target feature being located to find the set of most suitable feature realizations. We have evaluated the approach with an industrial case study, locating features with mean precision and recall values of around 73.75% and 73.31%, respectively (the sanity check obtains less than 35%).

CCS CONCEPTS

• **Information systems → Learning to rank**; • **Software and its engineering → Software product lines**;

ACM Reference format:
Ana C. Marcén, Jaime Font, Óscar Pastor, and Carlos Cetina. 2017. Towards Feature Location in Models through a Learning to Rank Approach. In *Proceedings of SPLC '17, Sevilla, Spain, September 25-29, 2017,* 8 pages.
https://doi.org/10.1145/3109729.3109734

1 INTRODUCTION

Feature location is known as the process of finding the set of software artifacts that realize a particular functionality of software system. No maintenance activity can be completed without locating in the first place the software artifact (e.g., code) that is relevant to the specific functionality [10]. Since Feature Location is one of the main activities performed during software evolution [14] and up to an 80% of a system's lifetime is spent on the maintenance and evolution of the system [21], there is a great demand for Feature Location approaches that can help developers to find relevant software artifacts in a family of software products.

Learning to Rank is known as a family of Machine Learning algorithms that automatically address ranking tasks [22]. The topic has gained interest in recent years [9], and Learning to Rank has been applied in a lot of fields [7] like document retrieval, collaborative filtering, expert finding, anti web spam, sentiment analysis, product rating, and feature location.

However, most of the research on Feature Location through Learning to Rank has been directed towards the location of features in source code artifacts [5, 10, 33], neglecting other software artifacts such as models. Therefore, there is a dearth of Feature Location approaches that research how to apply Learning to Rank in order to locate the model elements that realize a feature.

In this work we propose LRFL-M (Learning to Rank for Feature Location in Models), which is an Feature Location approach that locates features in models through Leaning to Rank. The approach is based on Learning to Rank to assess the similarity between a feature description and the model fragments that could be the realizations of this feature. Given feature descriptions and model fragments known beforehand, the LRFL-M approach encodes them based on a domain ontology. Then, the classifier is trained based on the feature descriptions and the model fragments encoded. Finally, the similarity between a population of model fragments and the target feature being located are assessed through the classifier in order to find the set of most suitable feature realizations. Therefore, a rank allows knowing what model fragments best realize the target feature as output.

The presented approach was evaluated in CAF, a worldwide provider of railway solutions. Their trains can be found all over the world in different forms (regular trains, subway, light rail, monorail, etc.). The application of the approach shows that the mean values of precision and recall are 73.75% and 73.31%, respectively, while the sanity check is around 46% less than the presented approach.

The contribution of this paper is twofold. First, we show how to encode model elements and feature descriptions by means of a

[*] Also with Centro de Investigación en Métodos de Producción de Software, Universitat Politècnica de València.

10.2 ER'17 Paper (First page)

Title: Ontological Evolutionary Encoding to Bridge Machine Learning and Conceptual Models: Approach and Industrial Evaluation.

Authors: Ana Cristina Marcén, Francisca Pérez, and Carlos Cetina.

Proceedings: Conceptual Modeling 36th International Conference, ER 2017, Valencia, Spain, November 6–9, 2017, Proceedings

Location: Valencia, Spain - November 6-9, 2017

Publisher: Springer

Pages: 491–505

DOI: https://doi.org/10.1007/978-3-319-69904-2_37

Ontological Evolutionary Encoding to Bridge Machine Learning and Conceptual Models: Approach and Industrial Evaluation

Ana C. Marcén[1 2], Francisca Pérez[2], and Carlos Cetina[2]

[1] Centro de Investigación en Métodos de Producción de Software
Universitat Politècnica de València Camino de Vera, s/n, 46022 Valencia, Spain
[2] SVIT Research Group, Universidad San Jorge
Autovía A-23 Zaragoza-Huesca Km.299, 50830, Zaragoza, Spain
`{acmarcen,mfperez,ccetina}@usj.es`

Abstract. In this work, we propose an evolutionary ontological encoding approach to enable Machine Learning techniques to be used to perform Software Engineering tasks in models. The approach is based on a domain ontology to encode a model and on an Evolutionary Algorithm to optimize the encoding. As a result, the encoded model that is returned by the approach can then be used by Machine Learning techniques to perform Software Engineering tasks such as concept location, traceability link retrieval, reuse, impact analysis, etc. We have evaluated the approach with an industrial case study to recover the traceability link between the requirements and the models through a Machine Learning technique (RankBoost). Our results in terms of recall, precision, and the combination of both (F-measure) show that our approach outperforms the baseline (Latent Semantic Indexing). We also performed a statistical analysis to assess the magnitude of the improvement.

Keywords: Machine Learning, Traceability Link Recovery, Evolutionary Computation, Model Driven Engineering

1 Introduction

Machine Learning (ML) is known as the branch of artificial intelligence that gathers statistical, probabilistic, and optimization algorithms, which learn empirically. ML has a wide range of applications, including search engines, medical diagnosis, text and handwriting recognition, image screening, load forecasting, marketing and sales diagnosis, etc. Even though the research on ML has been applied in Software Engineering tasks that target source code artifacts [7, 33], other software artifacts such as conceptual models have been neglected.

Most of the ML techniques are designed to process feature vectors as inputs [8]. Feature vectors are known as the ordered enumeration of features that characterize the object being observed [10]. Therefore, to apply ML techniques in models, the first challenge consists in identifying the features from models and selecting the most suitable ones to encode the models in feature vectors.

10.3 CoopIS'17 Paper (First page)

Title: Introducing Collaboration for Locating Features in Models: Approach and Industrial Evaluation

Authors: Francisca Pérez, Ana Cristina Marcén, Raúl Lapeña, and Carlos Cetina.

Proceedings: On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part I

Location: Rhodes, Greece - October 23-27, 2017

Publisher: Springer

Pages: 114–131

DOI: https://doi.org/10.1007/978-3-319-69462-7_9

Introducing Collaboration for Locating Features in Models: Approach and Industrial Evaluation

Francisca Pérez( ), Ana C. Marcén, Raúl Lapeña, and Carlos Cetina

SVIT Research Group, Universidad San Jorge,
Autovía A-23 Zaragoza-Huesca Km. 299, 50830 Zaragoza, Spain
{mfperez,acmarcen,rlapena,ccetina}@usj.es

Abstract. Feature Location (FL) is one of the most important tasks in software maintenance and evolution. However, current works on FL neglected the collaboration of different domain experts. This collaboration is especially important in long-living industrial domains where a single domain expert may lack the required knowledge to fully locate a feature, so the collaboration among different domain experts could alleviate this lack of knowledge. In this work, we address collaboration among different domain experts by automatically reformulating their feature descriptions. With our approach, we extend existing FL approaches based on Information Retrieval and Linguistic rules to locate features in models. We evaluate our approach in a real-world case study from our industrial partner, which is a worldwide leader in train manufacturing. We analyze the impact of our approach in terms of recall, precision, and F-Measure. Moreover, we perform a statistical analysis to show that the impact of the results is significant. Our results show that our approach for collaboration boosts the quality of the results of FL.

Keywords: Collaborative information retrieval · Feature location · Query expansion · Model driven engineering

1 Introduction

Nowadays, work environments are characterized by an emphasis on collaborative team work [9]. Many empirical studies identified collaborative information seeking and retrieval as everyday work patterns in order to solve a shared information need and to benefit from the diverse expertise and experience of the team members [13].

Despite the importance of collaboration, Feature Location (FL) approaches neglected collaboration among different domain experts to find the set of software artifacts (e.g., code or models) that realize a specific feature. Even though collaboration is a useful and often necessary component of complex projects in industrial contexts when the task at hand is difficult or cannot be carried out by one individual [36].

To cope with this lack, the contribution of this paper is the introduction of collaboration for locating a target feature in models from different domain

10.4 MODELS'18 Paper (First page)

Title: Measures to report the Location Problem of Model Fragment Location

Authors: Manuel Ballarín, Ana Cristina Marcén, Vicente Pelechano, and Carlos Cetina.

Proceedings: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018

Location: Copenhagen, Denmark - October 14-19, 2018

Publisher: ACM

Pages: 189–199

DOI: https://doi.org/10.1145/3239372.3239397

Measures to report the Location Problem of Model Fragment Location

Manuel Ballarín
SVIT Research Group
Universidad San Jorge
Zaragoza, Spain
mballarin@usj.es

Ana C. Marcén
SVIT Research Group
Universidad San Jorge
Zaragoza, Spain
acmarcen@usj.es

Vicente Pelechano
pele@pros.upv.es
Centro de Investigación en Métodos de Producción de Software
Universitat Politècnica de València
Valencia, Spain

Carlos Cetina
SVIT Research Group
Universidad San Jorge
Zaragoza, Spain
ccetina@usj.es

ABSTRACT

Model Fragment Location (MFL) aims at identifying model elements that are relevant to a requirement, feature, or bug. Many MFL approaches have been introduced in the last few years to address the identification of the model elements that correspond to a specific functionality. However, there is a lack of detail when the measurements about the search space (models) and the measurements about the solution to be found (model fragment) are reported. Generally, the only reported measure is the model size. In this paper, we propose using five measurements (size, volume, density, multiplicity, and dispersion) to report the location problems. These measurements are the result of analyzing 1,308 MFLs in a family of industrial models over the last four years. Using two MFL approaches, we emphasize the importance of these measurements in order to compare results. Our work not only proposes improving the reporting of the location problem, but it also provides real measurements of location problems that are useful to other researchers in the design of synthetic location problems.

CCS CONCEPTS

• **Information systems → Presentation of retrieval results**; • **Software and its engineering → Model-driven software engineering**;

KEYWORDS

Model Fragment Location, Feature Location, Traceability Link Recovery, Bug Location

ACM Reference Format:
Manuel Ballarín, Ana C. Marcén, Vicente Pelechano, and Carlos Cetina. 2018. Measures to report the Location Problem of Model Fragment Location . In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3239372.3239397

1 INTRODUCTION

From the timeless traceability activity [21] to recent research efforts on Feature Location [16], [6], [7] and Bug Location [2], Model Fragment Location (MFL) has been gaining momentum. Overall, these MFL approaches address the identification of the model elements that are relevant to a requirement, feature, or bug.

Current MFL approaches have leveraged Information Retrieval, Linguistic techniques, and Search-based techniques to achieve the location of relevant model fragments. These approaches provide the algorithms and the parameters used to tune them in detail. Nonetheless, there is a lack of detail when the measurements about the search space (models) and the measurements about the solution (model fragment) are reported. Generally, the only reported measure is the model size. However, in most of the cases, the model-size values are not comparable among different works since different models are measured in different ways.

In this paper, we propose using five measurements (size, volume, density, multiplicity, and dispersion) to report the location problems during MFL. On the one hand, size and volume measure the search space. On the other hand, density, multiplicity, and dispersion measure the solution to be located. Our proposed measures are the result of analyzing 1,308 MFLs performed over the last four years in models of the industrial dimensions of CAF [1].

Properly reporting the location problem is important because otherwise it is not possible to compare the performance of different approaches with each other. It is not the same challenge to locate a large model fragment in a small model than to locate a small and scattered model fragment over several large models. We illustrate

[1]http://www.caf.net/en

10.5 JSS'20 Paper (First page)

Title: Traceability Link Recovery between Requirements and Models using an Evolutionary Algorithm Guided by a Learning to Rank Algorithm: Train Control and Management Case

Authors: Ana Cristina Marcén, Raúl Lapeña, Oscar Pastor, and Carlos Cetina.

Journal: Journal of Systems and Software

Date: May, 2020

DOI: https://doi.org/10.1016/j.jss.2020.110519

Traceability Link Recovery between Requirements and Models using an Evolutionary Algorithm Guided by a Learning to Rank Algorithm: Train Control and Management Case

Ana C. Marcén[a,b], Raúl Lapeña[a], Óscar Pastor[b], Carlos Cetina[a]

{*acmarcen,rlapena,ccetina*}*@usj.es, opastor@pros.upv.es*

[a]*SVIT Research Group*
Universidad San Jorge
[b]*Centro de Investigación en Métodos de Producción de Software*
Universitat Politècnica de València

Abstract

Traceability Link Recovery (TLR) has been a topic of interest for many years within the software engineering community. In recent years, TLR has been attracting more attention, becoming the subject of both fundamental and applied research. However, there still exists a large gap between the actual needs of industry on one hand and the solutions published through academic research on the other.

In this work, we propose a novel approach, named Evolutionary Learning to Rank for Traceability Link Recovery (TLR-ELtoR). TLR-ELtoR recovers traceability links between a requirement and a model through the combination of evolutionary computation and machine learning techniques, generating as a result a ranking of model fragments that can realize the requirement.

TLR-ELtoR was evaluated in a real-world case study in the railway domain, comparing its outcomes with five TLR approaches (Information Retrieval, Linguistic Rule-based, Feedforward Neural Network, Recurrent Neural Network, and Learning to Rank). The results show that TLR-ELtoR achieved the best results for most performance indicators, providing a mean precision value of 59.91%, a recall value of 78.95%, a combined F-measure of 62.50%, and a MCC value of 0.64. The statistical analysis of the results assesses the magnitude of the improvement, and the discussion presents why TLR-ELtoR achieves better results than the baselines.

Preprint submitted to Journal of Systems and Software *January 13, 2020*