Improving Feature Location in Long-Living Model-Based Product Families Designed with Sustainability Goals

Carlos Cetina[a,*], Jaime Font[a,b], Lorena Arcega[a,b], Francisca Pérez[a]

[a] *SVIT Research Group, Universidad San Jorge.*
Autovía A-23 Zaragoza-Huesca Km.299, 50830, Villanueva de Gállego (Zaragoza), Spain
[b] *Department of Informatics, University of Oslo.*
Postboks 1080 Blindern, 0316 Oslo, Norway

Abstract

The benefits of Software Product Lines (SPL) are very appealing: software development becomes better, faster, and cheaper. Unfortunately, these benefits come at the expense of a migration from a family of products to a SPL. Feature Location could be useful in achieving the transition to SPLs. This work presents our FeLLaCaM approach for Feature Location. Our approach calculates similarity to a description of the feature to locate, occurrences where the candidate features remain unchanged, and changes performed to the candidate features throughout the retrospective of the product family. We evaluated our approach in two long-living industrial domains: a model-based family of firmwares for induction hobs that was developed over more than 15 years, and a model-based family of PLC software to control trains that was developed over more than 25 years. In our evaluation, we compare our FeLLaCaM approach with two other approaches for Feature Location: (1) FLL (Feature Location through Latent Semantic Analysis) and (2) FLC (Feature Location through Comparisons). We measure the performance of FeLLaCaM, FLL, and FLC in terms of recall, precision, Matthews Correlation Coefficient, and Area Under the Receiver Operating Characteristics curve. The results show that FeLLaCaM outperforms FLL and FLC.

Keywords: Feature Location, Long-Living Software Systems, Architecture Sustainability, Software Product Lines, Model-Driven Engineering

1. Introduction

Software Product Lines (SPLs) [1] are capable of generating new products with the benefit of enabling a systematic reuse of the features that have already been developed in a family of products. The benefits of SPLs are very appealing: software development becomes better, faster, and cheaper when SPLs are used [2]. Unfortunately, these benefits come at the expense of having to perform a migration from a family of products to a SPL. This migration requires deep knowledge about the product family, and is not straightforward.

Researchers are investigating various techniques [3] to locate features in a product family, which could be useful in the transition to SPLs. Overall, these techniques extract features by means of a query (which describes the feature to be located) and the main assets of the product family (e.g., implementation code or specification models). The idea is that words in the query encode domain

*Corresponding author. Tel.: +34 976060100
Email addresses: ccetina@usj.es (Carlos Cetina), jfont@usj.es (Jaime Font), larcega@usj.es (Lorena Arcega), mfperez@usj.es (Francisca Pérez)

knowledge, and that features are implemented using a similar set of words throughout the assets of the product family. However, the utility of these techniques has yet to be fully validated in industrial scenarios, especially in industrial environments where families of products are developed over decades. When software products have been operating for more than 15 years, they are known as long-living [4].

Through this work, we present our FeLLaCaM approach for Feature Location. Given a long-living product family, our FeLLaCaM approach does not only take into account the query that describes the feature to be located, but it also takes into account the occurrences where the candidate features remain unchanged as well as the changes performed to the candidate features throughout product model retrospective. Our idea is to take advantage of the sustainability design [4, 5] (that long-living software systems should have) to address the challenge of Feature Location. If these already long-living product families are fully successful from a suitability point of view, their feature realizations supported changes during its life-cycle while remaining intact [6].

Our approach is materialized as an evolutionary algorithm, guided by the feature description, feature commonality, and feature modifications. The similarity of a candidate feature to the feature description is calculated by means of Latent Semantic Analysis [7]. The feature commonality and the feature modifications throughout the product family are calculated by means of Conceptualized Model Patterns [8]. The output of the approach is a set of solutions that can materialize the target feature.

We evaluated our approach in two long-living industrial domains, with both the model-based product family of BSH and the model-based product family of CAF:

- The BSH group (www.bsh-group.com) produces a family of firmwares for their induction hobs (sold under the brands of Bosch and Siemens). BSH has been developing this family of software products over more than 15 years.

- CAF (www.caf.net/en) produces a family of PLC software to control the trains they manufacture. CAF has been developing this family of software products over more than 25 years.

In our evaluation, we compare our FeLLaCaM approach with two other approaches for Feature Location: (1) FLL (Feature Location through Latent Semantic Analysis) and (2) FLC (Feature Location through Comparisons). FLL is a version of FeLLaCaM which is guided only by feature descriptions, as are other current Feature Location approaches. FLC is our implementation of the algorithms to locate features presented in [9]. Similar to other works [10, 11, 9, 12, 13, 14], FLC performs model comparison to locate the features. We apply FeLLaCaM, FLL, and FLC to the product families of BSH and CAF. Although neither BSH nor CAF are immune to the problem of knowledge vaporization [15], they provided us with documentation about some of their feature realization decisions [16]. For each one of the 96 features in BSH and the 121 features in CAF, the documentation provided a feature description and the approved feature realization. Taking the feature descriptions and the product families as input, we measure the performance of FeLLaCaM, FLL, and FLC in terms of recall, precision, Matthews Correlation Coefficient (MCC), and Area Under the Receiver Operating Characteristics curve (AUC) using the approved features as oracle.

The results show that FeLLaCaM and FLL achieve similar recall results (about 81%), while FLC obtains values around 57.96%. However, in terms of precision, FeLLaCaM (about 78.61%) outperforms both FLL (about 20.18%) and FLC (about 43.09%). The same occurs with MCC, FeLLaCaM obtains a value about 0.73, while FLL and FLC obtain values around 0.35. The AUC measurement also confirms that FeLLaCaM (about 0.87) provides better results than FLL (about 0.79) and FLC (about 0.7). In the face of already long-living SW systems (designed for sustainability) our FeLLaCaM approach improved the Feature Location.

The remainder of the paper is structured as follows: Section 2 provides a basic background of the formalization of the architecture and variability of the products used by our industrial partners. These are used in our running example. Section 3 presents

an overview of our FeLLaCaM approach. Section 4 describes the representation that is used to encode model fragments. Section 5 describes the generation of model fragments. Section 6 presents how the suitability of each model fragment is determined for the problem. Section 7 presents an overview of our implementation. Section 8 evaluates FeLLaCaM in two industrial domains and presents the results. Section 9 discusses the results. Section 10 describes the threats to validity. Section 11 summarizes the related work. Finally, Section 12 states the relevant conclusions.

2. Background

This section presents the Domain-Specific Language (DSL) used by our industrial partner BSH to specify the architecture of their Induction Hobs (IH) and then generate the firmware from the models, the IHDSL. IHDSL will be used throughout the rest of the paper to present a running example. Then, the Common Variability Language (CVL) is presented. CVL is the language used by our FeLLaCaM approach to formalize the location of the features as reusable model fragments.

2.1. The Induction Hobs Domain-Specific Language (IHDSL)

The newest IHs feature full cooking surfaces, where dynamic heating areas are automatically generated and activated or deactivated depending on the shape, size, and position of the cookware placed on top. In addition, there has been an increase in the type of feedback provided to the user while cooking, such as the exact temperature of the cookware, the temperature of the food being cooked, or even real-time measurements of the actual consumption of the IH. All of these changes are made possible at the expense of increasing the software complexity.

The Domain-Specific Language used by our industrial partner to specify the Induction Hobs (IHDSL) is composed of 46 meta-classes, 74 references among them, and more than 180 properties. However, in order to increase legibility and due to intellectual property rights concerns, we show a simplified subset of the IHDSL at the top of Figure 1.

Inverters are in charge of transforming the input electric supply to match the specific requirements of the IH. Then, the energy is transferred to the inductors through the channels. There can be several alternative channels, which enable different heating strategies depending on the cookware placed on top of the IH at run-time. The path followed by the energy through the channels is controlled by the power manager. Inductors are the elements where the energy is transformed into an electromagnetic field. Inductors can be organized into groups to heat larger cookware while sharing the user interface controllers.

2.2. The Common Variability Language applied to IHs

To formalize the variability among the products of the SPL, we need a variability specification that captures which model fragments are used by each of the products that can be built from the SPL. The presented FeLLaCaM approach uses the Common Variability Language (CVL) [17] due to its expressiveness to properly formalize the feature realizations in terms of model fragments. CVL defines variants of a base model conforming to MOF [18] (Meta-Object Facility, the Object Management Group metalanguage for defining modeling languages) by replacing variable parts of the base model with alternative model replacements found in a library.

The base model is a model described by a given DSL (here, IHDSL) that serves as the base for different variants defined over it. In CVL, the elements of the base model that are subject to variations are the placement fragments (hereafter placements). A placement can be any element or set of elements that is subject to variation. To define alternatives for a placement, we use a replacement library, which is a model that is described in the same DSL as the base model that will serve as a base to define alternatives for a placement. Each one of the alternatives for a placement is a replacement fragment (hereafter replacement). Similarly to placements, a replacement can be any element or set of elements that can be used as variation for a placement.

The bottom of Figure 1 shows an example of variability specification of IH through CVL. In the product realization layer, two placements are defined

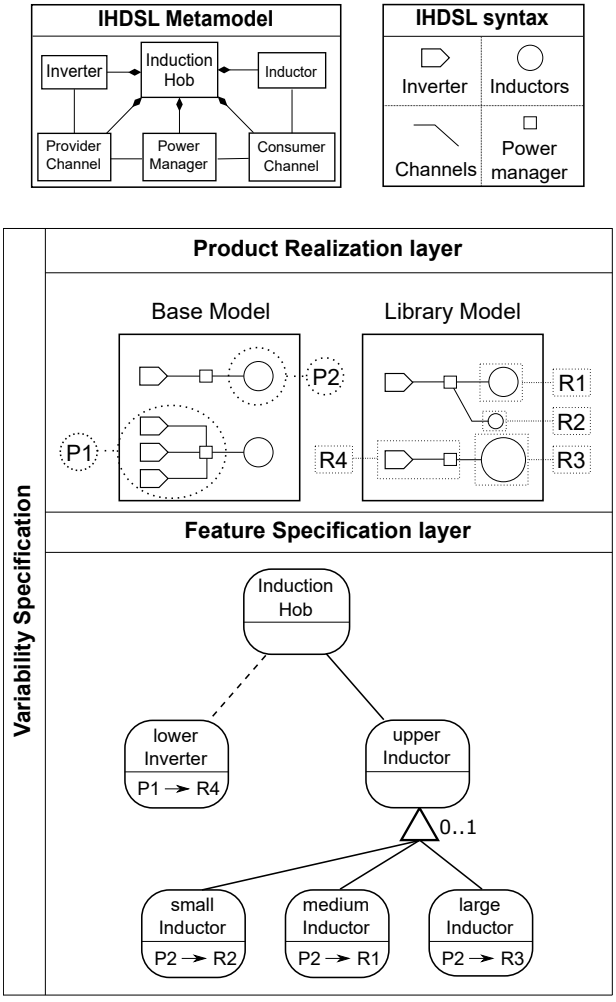

Figure 1: CVL applied to IHDSL

over an IH base model (P1 and P2). Then, four replacements are defined over an IH library model (R1, R2, R3, and R4). In the feature specification layer, a Feature Model is defined that formalizes the variability among the IH based on the placements and replacements. For instance, P1 can only be substituted by R4 (which is optional), but P2 can be replaced by R1, R2, or R3.

Note that each placement and replacement have a signature, which is a set of references (boundaries) going from and towards that placement or replacement. A placement can only be replaced by replacements with matching signatures. That is, the signature of the placement and the signature of the replacement have to be the same. For instance, the P2 signature has a reference from a power manager (outside the placement) to an inductor (inside the placement), while the R4 signature is a reference from a power manager (inside the replacement) to an inductor (outside the replacement). P2 cannot be substituted by R4 since their signatures do not match. The signature of R1 is a reference from a power manager (outside the replacement) to an inductor (inside the replacement); therefore P1 can be substituted by R1 as their signatures match.

Throughout the rest of the paper, we will use the term Feature Location as 'the process of obtaining the particular model elements (as a model fragment) that realize a particular feature defined by a given textual description'.

3. Overview of our FeLLaCaM Approach

This section presents the proposed FeLLaCaM approach for Feature Location in long-living model-based product families. The objective of the approach is to provide the model fragment from a given product family that realizes a specific feature being requested by the user. To do this, the approach receives as input a product family (including all the revisions for each of the product models present in the family) and relies on an evolutionary algorithm that iterates a population of model fragments and evolves them using genetic operations. The evolutionary algorithm is guided by a fitness operation that takes into account the feature description, the feature commonality, and the feature modifications. As output, the approach provides a list of model fragments that might be realizing the feature.

The top of Figure 2 shows an example of input to our approach.

- A set of **product model retrospectives** (hereafter the term retrospective will be used to refer to the product model and the set of past revisions for that particular product model created over the time) that contains the target feature.

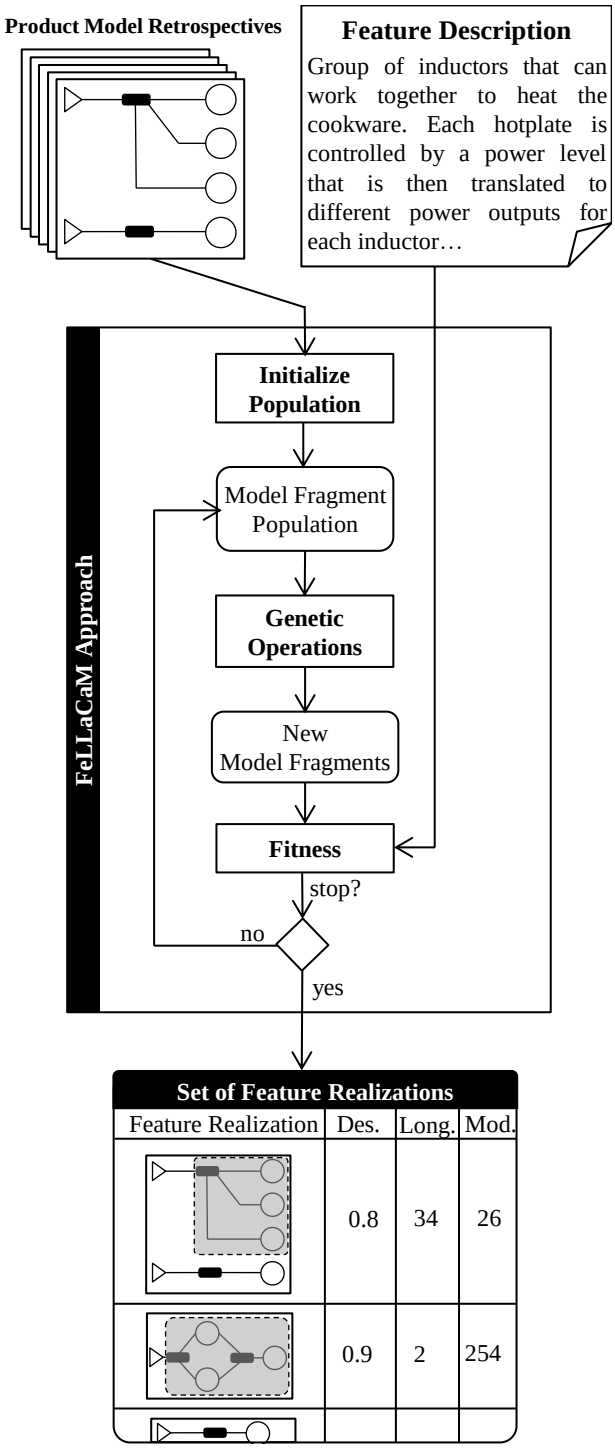

Figure 2: Overview of FeLLaCaM

The engineer selects a subset of product models from the entire family of products that include the feature to be located.

- A **feature description** of the target feature, using natural language. Typically these descriptions can come from textual documentation of the products, comments in the code, bug reports, or oral descriptions from the engineers. Therefore, the query will include some domain-specific terms that are similar to those used when specifying the product models. The knowledge of the engineers about the domain and the product models will be useful in selecting the textual description from the sources available.

The center of Figure 2 shows a simplified representation of the main steps of our approach.

- The **initialize population** step calculates an initial population of model fragments from the input set of product models. This initial population of model fragments is randomly extracted from the product models.

- The **genetic operations** generate the new generation of model fragments. First, a selection operation selects the model fragments that will be used as parents of the new model fragments. The fitness values are used to ensure that the best model fragments are chosen as parents. Then, a crossover operation mixes the model elements of the two parents into a new model fragment. Finally, a mutation operation introduces variations in the new model fragment (by adding or removing model elements) in hopes that the new model fragment achieves better fitness values than its parents.

- The **fitness** step assigns values that assess how good each model fragment is in the following terms:

  – Feature description. The more terms shared between the feature description and the properties of the model fragment, the higher this fitness value.

– Feature commonality. The more occurrences of the model fragment across the product model retrospective, the higher this fitness value.
– Feature modifications. The more modifications performed on the model fragment across the product model retrospective, the lower this fitness value.

The output of FeLLaCaM (see the bottom of Figure 2) is a set of model fragments that realize the target feature. The ranking can be ordered following different criteria, such as the similarity to the feature description, feature commonality, and feature modifications.

In overall, the aim of the approach is to find the feature realization that matches the feature description given by the user. To do so, the approach performs a search (among the different model fragments, obtained applying mutation and crossover operations, that could be realizations for the feature description) guided by a fitness function. To do so the fitness function will assign values based on the similarity with the textual description, the commonality of that feature realization candidate and the modifications overcome through the previous revisions of that product model.

The following sections describe the representation used to encode model fragments in our FeLLaCaM approach. Furthermore, these sections also describe the genetic operations of FeLLaCaM to generate new model fragments and how the fitness of each model fragment is determined in terms of similarity to the feature description, feature commonality, and feature modifications.

4. Model Fragment Encoding of the FeLLaCaM Approach

The possible solutions of our proposed approach are model fragments that realize the target feature. Traditionally, evolutionary algorithms encode each possible solution of the problem as a string of binary values. Each position of the string has two possible values: 0 or 1.

However, encoding each model fragment as a string of binary values is not straightforward. Each individual of our proposed approach will be a model fragment that is defined in one of the product models. In other words, each individual is a set of model elements that are present in one of the product models.

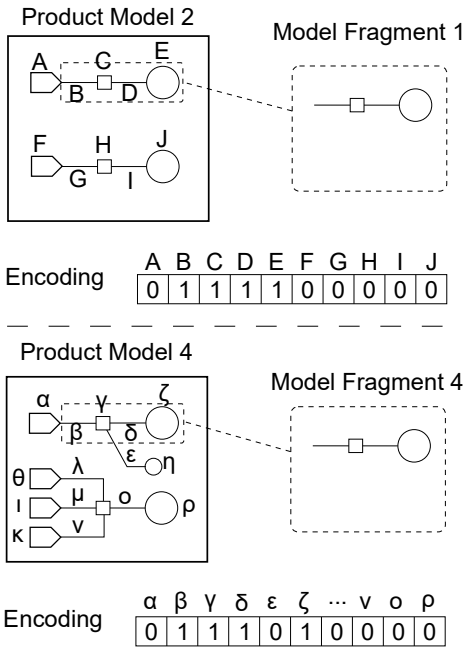

Figure 3: Representation of Model Fragments

Figure 3 shows two examples of our representation of model fragments. We denote each model element of the product model with a letter. In the example of the upper part of Figure 3, the letters A and F correspond to inverters, the letters B, D, G, and I correspond to channels, and the letters E and J correspond to inductors. Therefore, the string of binary values that represents the model fragment from this product model has the positions that correspond to each letter with a value of 0 or 1. 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.

Each model fragment representation depends on

the product model that it came from. Both of the examples in Figure 3 represent the same model fragment, but they come from different product models. Throughout the rest of the paper, we will refer to each individual as a model fragment that is part of a product model.

5. Genetic Operations of the FeLLaCaM Approach

The generation of model fragments is performed by applying genetic operators that we have adapted to work on model fragments. In other words, new fragments based on the existing ones are generated through the use of two genetic operators: the crossover operator and the mutation operator.

5.1. Crossover

In our encoding, there are two elements that can be mapped across the different individuals: the model fragment and the referenced product model. Therefore, our crossover operation will take the model fragment from the first parent and the product model from the second parent, generating a new individual that contains elements from both parents and thus preserving the basic mechanics of the crossover operation.

To achieve the above, our crossover operation is based on model comparisons. Figure 4 shows an example of the application of the crossover operation on model fragments. First, we select the model fragment from the first parent. Then, we select the product model from the second parent. The model fragment (from first parent) is then compared with the product model (from the second parent). If the comparison finds the model fragment in the product model, the operation creates a new individual with the model fragment taken from the first parent but referencing the product model from the second parent. In the case that the comparison does not find a similar element, the crossover will return the first parent unchanged.

This operation enables the search space to be expanded to a different product model, i.e., both model fragments (the one from the first parent and the one from the new individual) will be the same. However, since each of them is referencing a different product model, they will mutate differently and provide different individuals in further generations.

5.2. Mutation

Figure 5 shows an example of our mutation for model fragments. Each model fragment is associated to a product model, and the model fragment mutates in the context of its associated product model. In other words, the model fragment will gain or drop some elements, but the resulting model fragment will still be part of the referenced product model. The mutation possibilities of a given model fragment are driven by its associated product model.

To perform the mutation, the type of mutation that will occur (either the addition or removal of elements) is decided randomly:

- **Subtractive Mutation:** This kind of mutation randomly removes some elements from the model fragment. The only constraint is that the elements be selected from the edges of the model fragment (they are connected with a single element). Therefore, the resulting model fragment is still connected (we can navigate from any element to any other element through the connections between the elements) and it is not split into several isolated groups of elements. Since the resulting model fragment is a subset of the original model fragment and the original was present in the referenced product model, the resulting product model will always be present in the referenced product model.

- **Additive Mutation:** This kind of mutation randomly adds some elements to the model fragment. The only constraint is that the resulting model fragment be part of the referenced product model. To achieve this, the boundaries of the model fragment with the rest of the product model are identified and then a random element from the boundary is added to the resulting model fragment. By doing so, the mutated model fragment will be part of the referenced product model.

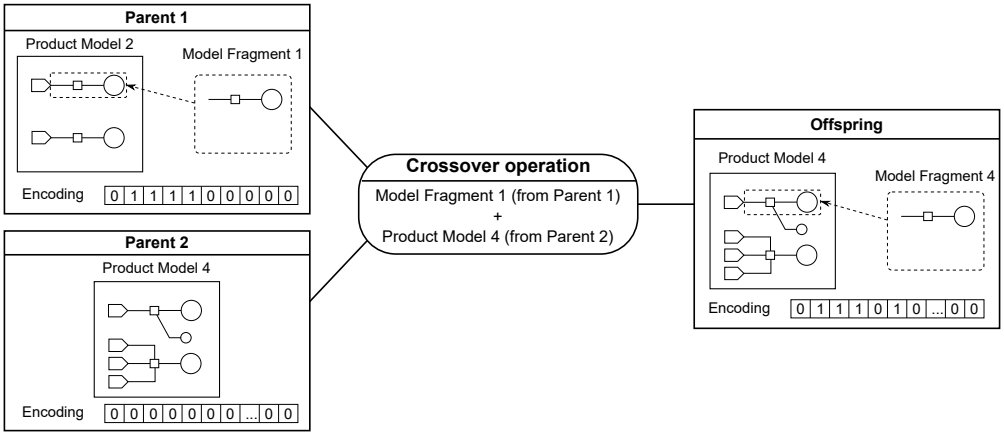

Figure 4: Crossover Operation

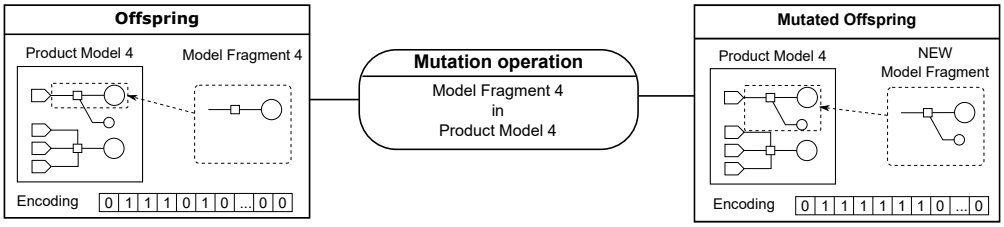

Figure 5: Mutation Operation

As a result, a new model fragment is created, but it still references the same product model. In other words, the individual represents other possible feature realizations (that are present in the product model) for the specific feature being located.

6. Fitness of the FeLLaCaM Approach

In evolutionary algorithms, the fitness step is used to imitate the different degrees of adaptation to the environment that different individuals have. Following this idea, our fitness step is used to determine the suitability of each model fragment to the problem. The input of this step is a population of model fragments, and the output produced is a set where each model fragment has been assigned with three fitness values: similarity to the feature description, feature commonality throughout the product model retrospective, and feature modifications throughout the product model retrospective.

6.1. Model Fragment similarity to the Feature Description

To assess the relevance of each model fragment in relation to the feature description provided by the user, we apply methods based on Information Retrieval (IR) techniques. Specifically, we apply Latent Semantic Analysis (LSA) [7] to analyze the relationships between the description of the feature provided by the user and the model fragments. There are many IR techniques, but most of the efforts show better results when applying LSA [19, 20, 21].

LSA constructs vector representations of a query and a corpus of text documents by encoding them as a term by document co-occurrence matrix, (i.e., a matrix where each row corresponds to terms and each column corresponds to documents, with the last

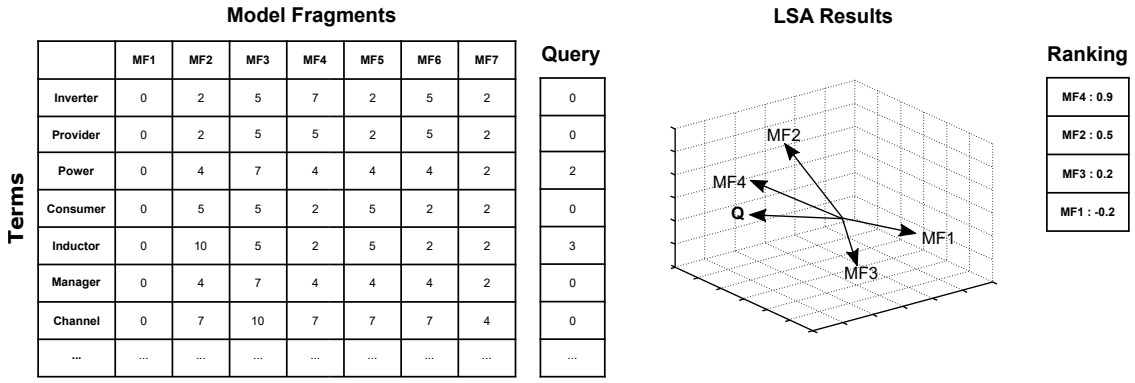

Figure 6: Fitness Operation via Latent Semantic Analysis (LSA)

column corresponding to the query). Each cell holds the number of occurrences of a term (row) inside a document or the query (column).

LSA provides good results when applied to source code [19, 20, 21], product models are representations at a higher abstraction level than the source code and the language used to build them is closer to the feature description language; therefore, we expect it to work better than when applied to source code.

In our work, the documents are model fragments, i.e., a document of text is generated from each of the model fragments. The text of the document corresponds to the names and values of the properties and methods of each model fragment (e.g. a model element of the class inductor will contain some properties related to its coil manufacturer and heat potential). The query is constructed from the terms that appear in the feature description. If the textual terms used for the model and the feature description differ too much the LSA will not work. Therefore, the text from the documents (model fragments) and the text from the query (feature description) are homogenized by applying well-known Natural Language Processing techniques to eventually reduce this gap. If the languages used differ too much, other techniques as manual annotation of the model elements could be applied at the expense of increasing the effort.

- First, the text is tokenized (divided into words). Usually, a white space tokenizer can be applied (which splits the strings whenever it finds a white space), but for some sources of description, more complex tokenizers need to be applied. For instance, when the description comes from documents that are close to the implementation of the product, some words could be using CamelCase naming.

- Second, we apply the Parts-of-Speech (POS) tagging technique. POS tagging analyzes the words grammatically and infers the role of each word in the text provided. As a result, each word is tagged, which allows the removal of some categories that do not provide relevant information. For instance, conjunctions (e.g., *or*), articles (e.g., *a*) or prepositions (e.g., *at*) are words that are commonly used and do not contribute relevant information that describes the feature, so they are removed.

- Third, stemming techniques are applied to unify the language that is used in the text. This technique consists of reducing each word to its roots, which allows different words that refer to similar concepts to be grouped together. For instance, plurals are turned into singulars (*inductors* to *inductor*) or verbs tenses are unified (*using* and *used* are turned into *use*).

The union of all the keywords extracted from the documents (model fragments) and from the query

(feature description) are the terms (rows) used by our LSA fitness.

Figure 6 (left) shows an example of the co-occurrence matrix for our running example. Each column is one of the model fragments. The last column is the query obtained from the feature description of the user. Each row is one of the terms extracted from the corpuses of text composed by all of the model fragments and the query itself (we show the terms before the stemming process to improve readability). Each cell shows the number of occurrences of each of the terms in the model fragments.

Once the matrix is built, we normalize and decompose it into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [22]. One vector that represents the latent semantics of the document is obtained for each model fragment and the query. Finally, the similarities between the query and each model fragment are calculated as the cosine between the two vectors. The fitness value that is given to each model fragment is the one that we obtain when we calculate the similarity, obtaining values between -1 and 1.

Figure 6 (right) shows a three-dimensional graph of the LSA results. The graph shows the representation of each one of the vectors, which are labeled with letters that represent the names of the model fragments. Finally, after the cosines are calculated, we obtain a value for each of the model fragments, indicating its similarity with the query.

6.2. Model Fragment Commonality and Modifications in the Product Family

To assess the relevance of each model fragment in terms of feature commonality and feature modifications throughout the product model retrospective, we apply Conceptualized Model Patterns (CMP) [8]. Given a model fragment, CMP allows alternative model fragments (throughout the product model retrospective) that match the model signature of the given model fragment to be extracted (see the model signature of Section 2.2). By calculating the differences among the given model fragment and the alternative model fragments, we can determine the commonality and modification fitness.

We adapted CMP to calculate feature commonality and modifications throughout the product model retrospective as follows:

- First, the model fragment signature is calculated for an input model fragment; that is, we calculate the boundary information between the model fragment and the rest of the model elements of the product model. Specifically, we calculate the set of incoming and outgoing relationships regarding the model fragment and the product model (see Step 1 of Figure 7).
- We then match the model fragment signature against each product model from the product model retrospective (the set of past revisions for that particular product model). If the match is positive, the model fragment of that particular product that matches with the model fragment signature is extracted. We denote the set of extracted model fragments as Alternative Model Fragments (AMF) (see Step 2 of Figure 7).
- The commonality of the input model fragment is determined by the number of occurrences of the input model fragment in AMF. This value represents the times that the model fragment remained unaltered throughout the product family (see Step 3 of Figure 7).
- The modifications of the input model fragment is determined by the summation of model differences (in terms of model elements) between the input model fragment and each model fragment of the AMF. This value represents the total number of changes performed to the input model fragment throughout the product model retrospective (see Step 4 of Figure 7).

Figure 8 shows the application of our adapted CMP to Model Fragment 5 (MF5) of the running example. The first column shows MF5, the second column shows the model fragment signature calculated for the model fragment, and the third column shows the matching of the model fragment signature against each of the product models (AMF1-AMF6).

Figure 9 depicts the model comparisons between MF5 and its AMF (AMF1-AMF6) calculated by

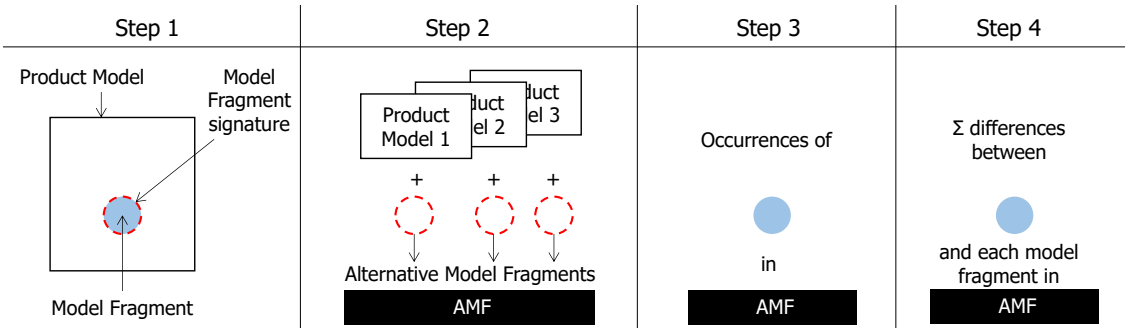

Figure 7: The steps of our adapted CMP to calculate feature commonality and modifications

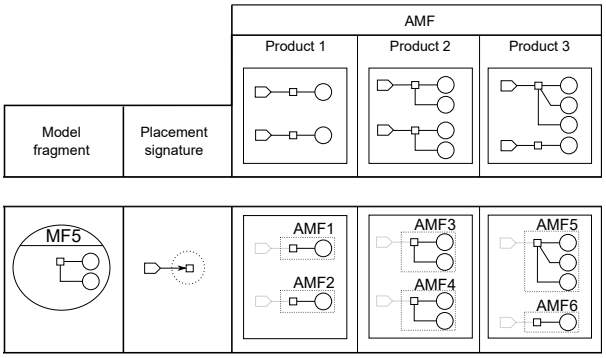

Figure 8: Application of our adapted CMP to MF5

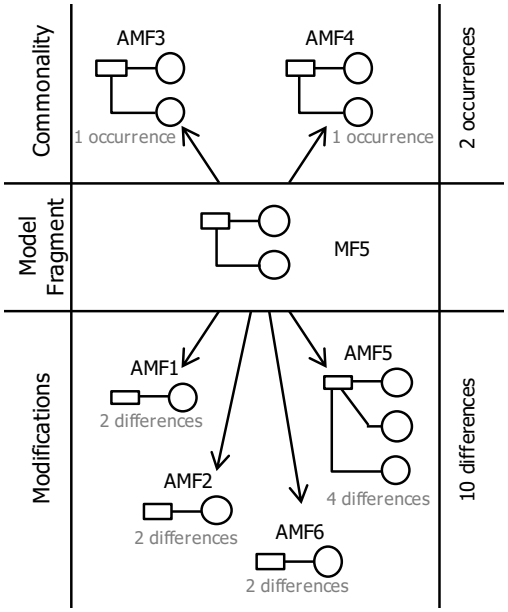

Figure 9: Feature commonality and modification values of MF5

means our CMP. There are two occurrences of MF5 in AMF (AMF3 and AMF4); therefore, the feature commonality value is 2. There are two differences (in terms of model elements) between MF5 and AMF1. There are also two differences between MF5 and AMF2 and two differences between MF5 and AMF6. Finally, there are four differences between MF5 and AMF5. Therefore, the feature modification value is 10. Both the feature commonality and feature modification values, in conjunction with the similarity to the feature description (see Section 6.1), are the fitness values that guide the evolutionary algorithm of our FeLLaCaM approach.

7. Implementation Details

Our algorithm is based on NSGA-II [23], one of the most frequently used multiobjective genetic algorithms. Given a population of model fragments (where each model fragment has a fitness value for its feature description (see section 6.1), feature commonality and feature modifications (see section 6.2)), NSGA-II orders these model fragments by means of nondominated sorting. A model fragment is nondominated whether does not exist another model fragment better than the current one in some fitness value without worsening other fitness value. As a result, NSGA-II finds pareto-optimal model

fragments.

We implemented our evolutionary algorithm as outlined in Algorithm 1. During the generation of the initial population (Lines 3-8), model fragments are randomly generated. The algorithm evaluates the fitness for each model fragment and adds it to the initial population. During the evolution process (Lines 11-18), new offspring is generated as a result of selecting model fragments (by means of the binary tournament selection), recombining them and applying a mutation operation. The algorithm evaluates the fitness for each offspring and adds it to a temporal population (Line 17). By means of a combination of non-dominated sorting and crowding distance sorting [23], the algorithm selects the model fragments from both the old population and the temporal population (Line 19) to create a new population (Line 20).

Algorithm 1 FeLLaCaM (NSGA-II-based)

Require: P_{size}, $p_{crossover}$, $p_{mutation}$, $eval_{max}$
Ensure: PF (a set of nondominated solutions)
1: $P = \emptyset$
2: $evaluations = 0$
3: **for** $i = 1$ to P_{size} **do**
4: $s = NewSolution()$
5: $EvaluateFitness(s)$
6: $evaluations = evaluations + 1$
7: $P = P + s$
8: **end for**
9: **while** $(evaluations < eval_{max})$ **do**
10: $P_O = \emptyset$
11: **for** $i = 1$ to $P_{size}/2$ **do**
12: $parents = Selection(P)$
13: $offsp = Crossover(parents, p_{crossover})$
14: $offsp = Mutation(offsp, p_{mutation})$
15: $EvaluateFitness(offsp)$
16: $evaluations = evaluations + 1$
17: $P_O = P_O + offsp$
18: **end for**
19: $P = P \cup P_O$
20: RankingAndCrowdingDistance(P)
21: **end while**
22: $PF = BestFront(P)$

Table 1: FeLLaCaM configuration parameters

Parameter description	Value
$Size$: Population Size	100
μ: Number of Parents	2
λ: Number of offspring from μ parents	2
r: Solutions replaced at population size	2
$p_{crossover}$: Crossover probability	0.9
$p_{mutation}$: Mutation probability	0.1

The crossover operation is applied with a crossover probability of 0.9. The mutation operation is applied with a probability of 0.1. The number of generations (repetitions of the genetic operations and fitness loop) allowed for the algorithm is 2500, as it is the value needed by our case studies to converge (Note that this value is case specific). The rest of the settings are detailed in Table 1. We have principally chosen values for those settings that are commonly used in the literature [24].

As suggested by [25] and confirmed in [26], tuned parameters can outperform default values generally, but they are far from optimal in individual problem instances. Therefore, the objective of this paper is not to tune the values to improve the performance of our algorithm but rather to compare it to a different version of our algorithm using default parameter values.

We have used the Eclipse Modeling Framework to manipulate the models and CVL to manage the fragments of models and implement CMP. The IR techniques used to process the language have been implemented using OpenNLP [27] for the POS-Tagger and Snowball [28] for the stemming. The LSI has been implemented using the Efficient Java Matrix Library (EJML [29]). The genetic operations are built upon the Watchmaker Framework for Evolutionary Computation [30].

8. Evaluation

This section presents the evaluation of our approach: the oracle preparation, the compared approaches, the experimental setup with a description of the case studies where we applied the evaluation, and the results obtained. To evaluate the approach,

we applied it to two long-living industrial case studies from two of our partners: BSH, the leading manufacturer of home appliances in Europe; and CAF, an international provider of railway solutions all over the world.

8.1. Oracle Preparation

The oracle will be considered the ground truth and will be used to compare the results provided by the approach. To prepare the oracle, our industrial partners provided us with documentation about feature decisions that they have been taken in the product models. These feature decisions comprise feature descriptions using natural language (e.g., these descriptions can come from textual documentation) and the approved feature realizations, which are a set of model fragments that realize a target feature.

Next, we apply the methods based on IR techniques as described in Subsection 6.1 to the feature descriptions, and the approved feature realizations provided by our industrial partners are considered the oracle.

8.2. Compared Approaches

We are going to compare the presented approach (FeLLaCaM) with two other approaches for Feature Location in Models: (1) FLL (Feature Location through Latent Semantic Analysis) and (2) FLC (Feature Location through Comparisons).

FLL is a version of FeLLaCaM which is guided only by the feature description using LSA (see Subsection 6.1). This comparison will enable us to determine the impact of the feature commonality and modifications fitness values present in FeLLaCaM. Hence, FLL is a Single-Objective Evolutionary Algorithm (SOEA), whereas FeLLaCaM is Multi-Objective Evolutionary Algorithm (MOEA). The work in [31] shows that common MOEA measures such as hypervolume [32] are not necessarily suitable for comparing solutions by MOEAs (our FeLLaCaM approach) with solutions by SOEAs (FLL in this work). Therefore, in order to compare FLL with FeLLaCaM, we first take the best solution of FLL for its single-objective (the similarity with the feature description). Second, we take the best solution of FeLLaCaM with regard to the objective of FLL (the similarity with the feature description) as described in [31]. Finally, we compare the best solution obtained in both FLL and FeLLaCaM with the oracle in order to obtain performance measurements.

Traditionally, Feature Location in models has been performed through model comparisons among the members of the family of models [10, 11, 9, 12, 13, 14]. These works classify the elements based on their similarity and identify the dissimilar elements as the features realizations. The predominant technology of choice to implement their approaches is EMF Model Compare. EMF Model Compare relies on Model Matching to perform the comparisons. To relate our work to the above works, we also compare our approach to FLC. FLC is our implementation of the algorithms to locate features presented in [9]. As the above works, FLC also uses EMF Model Compare to perform the model comparisons. Conversely to our approach, FLC do not use: evolutionary algorithm, LSA, nor the model retrospective.

8.3. Experimental Setup

The goals of this experiment are both the evaluation of our FeLLaCaM approach in terms of performance measurements and the comparison of FeLLaCaM with FLL and FLC. Figure 10 shows an overview of the process that was followed to perform the evaluation. The left part of the figure shows the inputs of the evaluation process provided by our industrial partners, which are the product family and a feature description for each of the features present in each case study. The descriptions have been extracted from internal documents by our industrial partners and have an average of 50 words each. They do not follow any specific template, an example of feature description can be seen in the top right part of Figure 2.

Then, each of the feature descriptions and the family of models are fed to the three different approaches, resulting in a feature realization in the form of a model fragment. Then, the resulting feature realizations are compared with the oracle. To compare them, we are going to use an error matrix [33], also known as confusion matrix.

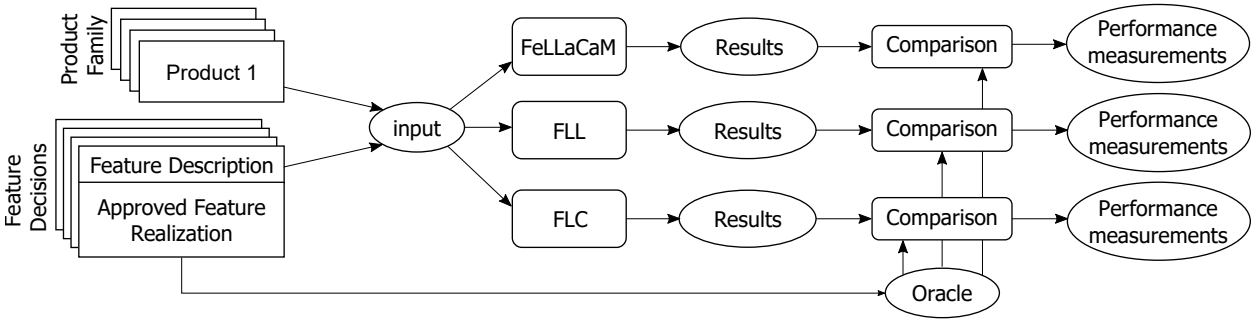

Figure 10: Evaluation process

A confusion matrix is a table that is often used to describe the performance of a classification model (in this case the approaches evaluated) on a set of test data (the resulting model fragments) for which the true values are known (from the oracles). In our case, each feature realization is a model fragment composed of a subset of the model elements that are present in the product model (where the feature is being located). Since, the granularity will be at the level of model elements, each model element presence or absence will be considered as a classification. Therefore, our confusion matrices will distinguish between two values (TRUE or presence and FALSE or absence).

Figure 11 shows an example of the comparison process performed to compare a result from one of the evaluated approaches with the ground truth from the oracle and the resulting confusion matrix. The left part shows the actual realization of the feature #1 (obtained from the oracle and considered the ground truth) while the right part shows the predicted realization of the feature #1 outputted by the approach. The confusion matrix arranges the results of the comparison into four categories:

- True positive (TP): A model element present in the predicted realization that is also present in the actual realization (e.g.: model element B is a TP).
- True Negative (TN): A model element not present in the predicted realization that is not present in the actual realization (e.g.: model element H is a TN)
- False Positive (FP): A model element present in the predicted realization that is not present in the actual realization (e.g.: model element A is a FP)
- False Negative (FN): A model element not present in the predicted realization that is present in the actual realization (e.g.: model element D is a FN)

The confusion matrix holds the results of the comparison between the predicted results and the actual results. However, in order to evaluate the performance of the approach it is necessary to extract some measurements from the confusion matrix. The next subsection presents the five measurements that we use to evaluate the performance of our approach.

8.3.1. Performance measurements

In this subsection we present the five measurements (derived from the confusion matrices) used to evaluate the performance of the presented approach (FeLLaCaM) and to compare it with two other approaches (FLL and FLC). The five measurements are Precision, Recall, F-Measure, Matthews Correlation Coefficient (MCC), and Area Under the Receiver Operating Characteristics curve (AUC).

Precision measures the number of elements from the prediction (result of the approach) that are correct according to the ground truth (the oracle). **Recall** measures the number of elements of the

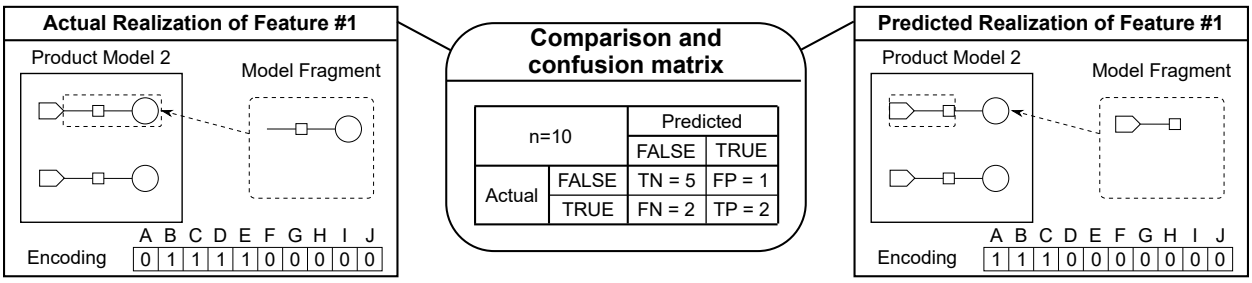

Figure 11: Example of Comparison process and confusion matrix

ground truth (the oracle) that are correctly retrieved by the prediction (result of the approach). The **F-measure** combines both recall and precision as the harmonic mean of precision and recall. The Recall, Precision and F-Measure are calculated as follows:

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

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

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

Recall values can range between 0% (which means that no single model element from the realization of the feature obtained from the oracle is present in any of the model fragments of the feature candidate) to 100% (which means that all the model elements from the oracle are present in the feature candidate).

Precision values can range between 0% (which means that no single model fragment from the feature candidate is present in the realization of the feature obtained from the oracle) to 100% (which means that all the model fragments from the feature candidate are present in the feature realization from the oracle). A value of 100% precision and 100% recall implies that both feature realizations are the same.

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

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

In addition, a Receiver Operating Characteristics (ROC) [34] graph is a technique for visualizing, organizing and selecting classifiers based on their performance that has into account the TN rate. In a ROC graph, the Rate of True Positives (TPR) and the Rate of False Positives (FPR) are compared into a two-dimensional space. By doing so, the graph depicts relative tradeoffs between benefits (TP) and costs (FP). A common method to compare different classifiers is to compare their ROC curves calculating the **AUC**. The TPR and FPR are calculated as follows:

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

$$FPR = \frac{FP}{FP + TN}$$

8.3.2. BSH

The first case study where we apply our evaluation process is BSH (already presented in Section 2.1 as the running example). The oracle is composed of 46 induction hob models and its revisions over time where, on average, each product model is composed of more than 500 elements. The oracle includes 96 different features that can be part of a particular product model. Those features correspond to products that are currently being sold or will be released to the market in the near future. For each of the 96 features, we create a test case that includes the

set of product models where that feature is used and a feature description (this information is obtained from the documentation).

For this case study, we executed 30 independent runs for each of the 96 test cases for each approach (as suggested by [35]), i.e., 96 (features) x 3 (approaches) x 30 repetitions = 8640 independent runs.

8.3.3. CAF

The second case study where we apply our evaluation process is CAF. Their trains can be seen 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 through 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, or the circuit breaker that isolates or connects the electrical circuits of the train. The control software of the train unit is in charge of making all the equipment cooperate to achieve the train functionality while guaranteeing compliance with the specific regulations of each country.

The DSL of our industrial partner has the required expressiveness to describe the interaction between the main pieces of equipment installed in a train unit. Moreover, this DSL also has the required expressiveness to specify non-functional aspects related to regulation, such as the quality of signals from the equipment or the different levels of installed redundancy.

As an example, the high voltage connection sequence can be described with the DSL. One of the scenarios from this sequence is the one presented when it is initiated, under demand from the train driver, who requests its start through interface devices fitted inside the cabin. The control software is in charge of raising the pantograph to harvest power from the overhead wire and of closing the circuit breaker so the energy can get to converters that adapt the voltage to charge batteries which, in turn, power the traction equipment. Another example of the functionality that the DSL can specify is the coupling between train units. A train unit can physically connect to a second train unit and control it in order to increase its passenger capacity or to rescue the second train unit in case it has suffered any damage while functioning.

Again, we extract an oracle that is composed of 23 trains (and its revisions over time) where each product model is composed of more than 1200 elements on average. They are built from 121 different features that can be part of a particular product model. For each of the 121 features, we create a test case that includes the set of product models where that feature is used and a feature description (this information is obtained from the documentation).

For this case study, we executed 30 independent runs for each of the 121 test cases for each approach, i.e., 121 (features) x 3 (approaches) x 30 repetitions = 10890 independent runs. The sum for the two case studies presented gives a total of 19530 independent runs.

8.4. Results

In this section, we present the results obtained for each case study in FeLLaCaM, FLL and FLC. Figure 12 shows the charts with the recall and precision results for the industrial domain of BSH (left column) and CAF (right column). A dot in a graph represents the average result of precision and recall for each feature (96 features in BSH and 121 features in CAF) for the 30 repetitions. The first row of charts shows the results of FeLLaCaM, the second row of charts shows the results of FLL, and the third row of charts shows the results of FLC.

Table 2 shows the mean values of precision, recall, F-measure, MCC and AUC for the approaches evaluated in the two case studies. FeLLaCaM and FLL obtain similar results for recall (an average value of 81%) while FLC obtains an average value of 57.96%. FeLLaCaM reaches the best results for precision, providing a precision value of 83.59% in the BSH case study and a precision value of 73.62% in the CAF case study. FLC is the second in precision, with values of 44.27% in BSH and 41.90% in CAF. FLL achieves a precision value of 22.95% in the BSH case study and a precision value of 17.41% in the

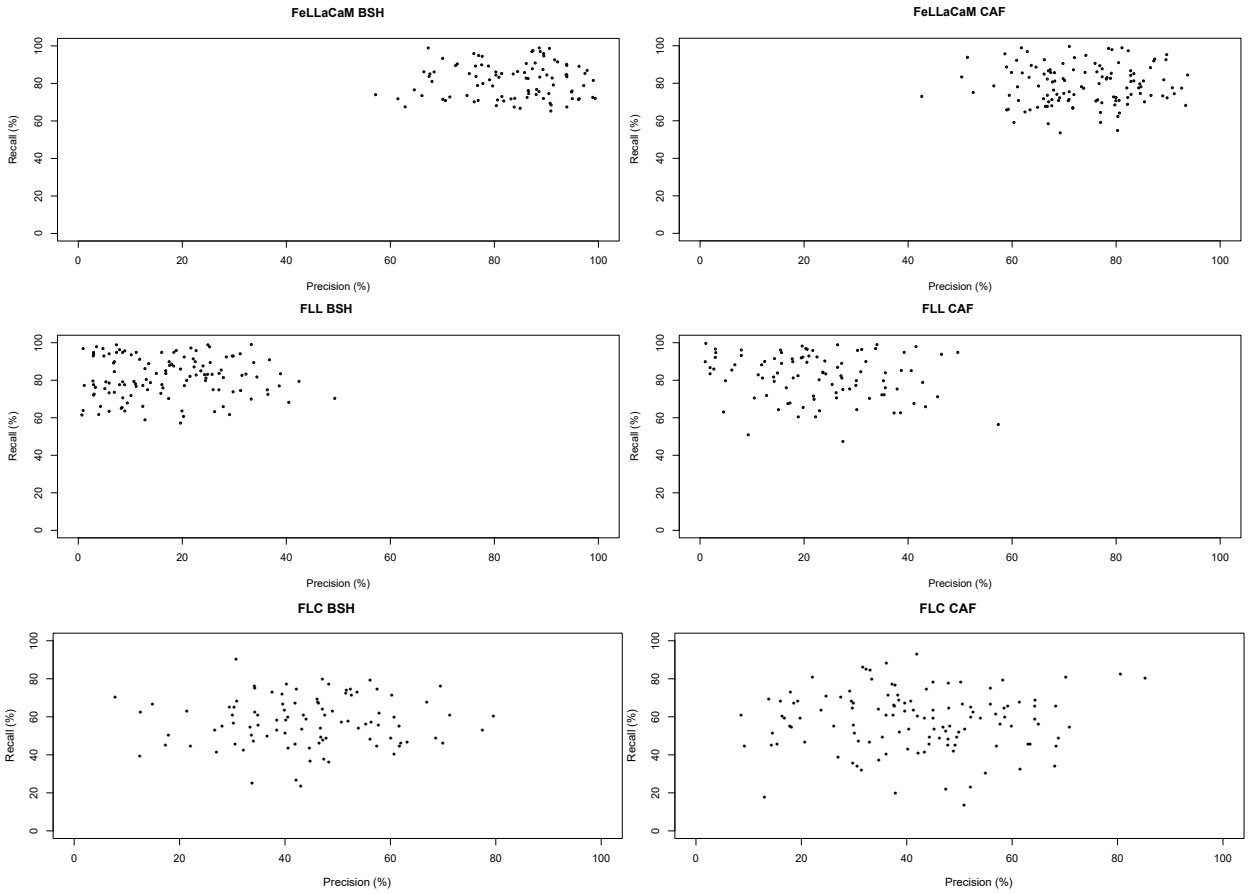

Figure 12: Mean Precision and Recall for FeLLaCaM, FLL and FLC

CAF case study. In terms of precision, FeLLaCaM (78.61% BSH-CAF) outperforms FLC (43.09% BSH-CAF) and FLL (20.18% BSH-CAF).

Regarding MCC measurements, FeLLaCaM reaches the best results, providing a value of 0.76 in the BSH case study and a value of 0.71 in the CAF case study. FLL and FLC obtains similar results, an average value of 0.35. In terms of AUC measurement, FeLLaCaM obtains the best results, a value of 0.88 in the BSH case study and a value of 0.87 in the CAF case study. It is followed by FLL with an average value of 0.79. Finally, FLC obtains an average value of 0.7. Figure 13 shows the graphical representation of the ROC curves for FeLLaCaM, FLL and FLC in BSH and CAF.

9. Discussion

The results of evaluating our approach show that the obtained solutions do not include all the model elements of the features. We detected that this happens because the fitness step that guides the evolutionary algorithm is not giving the highest fitness values to the approved feature realizations. The following paragraphs explain the issues that are causing this outcome:

1. **Incomplete feature description:** The feature description does not completely describe the

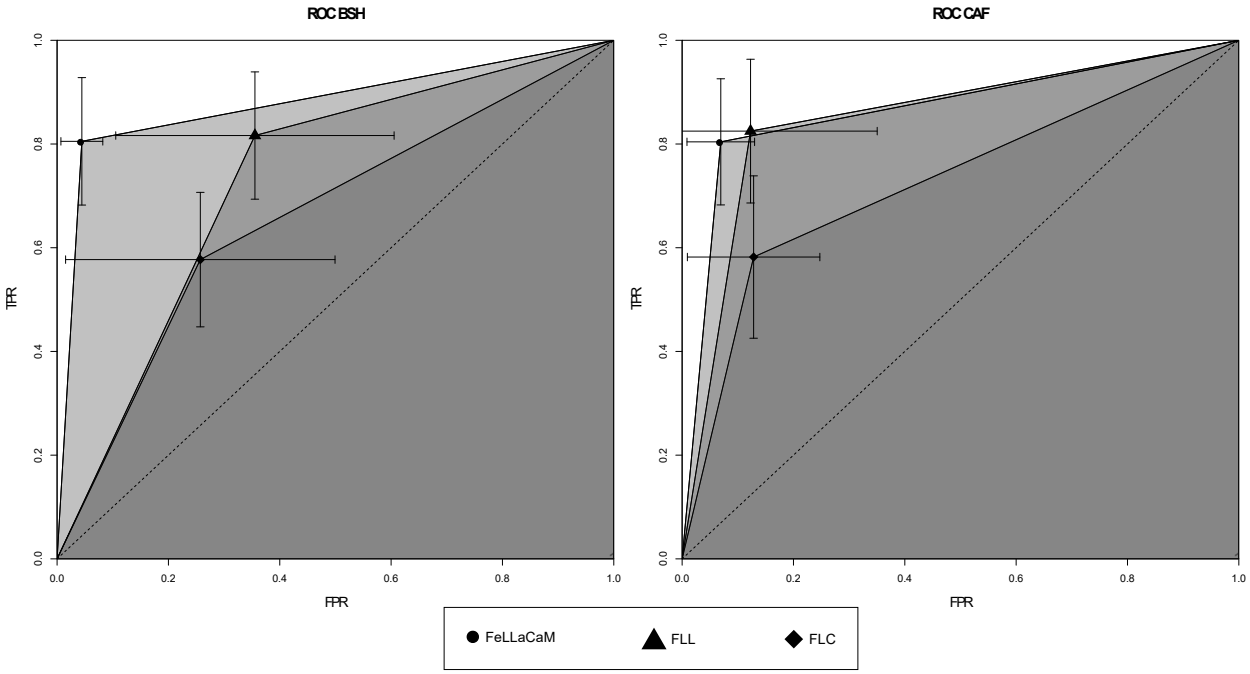

Figure 13: ROC curves for FeLLaCaM, FLL and FLC in BSH and CAF

approved feature realization. The second column of Figure 14 illustrates this issue: the feature description includes the $DomainTerm_A$ but it does not include the $DomainTerm_B$. However, $DomainTerm_B$ is included in the approved feature realization and it should be part of a complete feature description.

2. **Vocabulary mismatch:** For a particular concept, the term used in the feature description is different from the term used in the approved feature realization. The third column of Figure 14 illustrates this issue: the feature description includes the $DomainTerm_A$ that differs from the $DomainTerm_{A'}$ included in the approved feature realization. However, both $DomainTerm_A$ and $DomainTerm_{A'}$ stand for the same concept.
3. **Domain terms in the code generation rules:** Model elements of the approved feature realization may lack domain terms that are embedded in the code generation rules. The fourth column of Figure 14 illustrates this issue: the approved feature realization includes the $DomainTerm_B$. The code generation rules include the $DomainTerm_A$ and *Implementation terms*. However, a complete feature description includes both $DomainTerm_A$ and $DomainTerm_B$.
4. **Modification of boundary model elements:** Model elements that play the role of boundary information (incoming and outgoing relationships regarding the model fragment and the product model) of the approved feature realization are modified through the product model retrospective.
5. **Modification of no-boundary model elements:** Model elements that are within the approved feature realization are modified through the product model retrospective.

The issues 1-3 (incomplete feature description, vocabulary mismatch, and domain terms in the code generation rules) negatively influence the fitness value of feature similarity. These issues not only pe-

Table 2: Mean values and standard deviations for Precision, Recall, F-measure, MCC, and AUC in the two case studies

	Precision $\pm$ (σ)		
Case Study	FeLLaCaM	FLL	FLC
BSH	83.59 $\pm$ 9.86	22.95 $\pm$ 12.36	44.27±14.53
CAF	73.62 $\pm$ 10.27	17.41 $\pm$ 11.02	41.90±16.16
	Recall $\pm$ (σ)		
Case Study	FeLLaCaM	FLL	FLC
BSH	81.48 $\pm$ 9.09	81.95 $\pm$ 12.12	57.71±12.97
CAF	79.07 $\pm$ 10.42	81.50 $\pm$ 10.83	58.20±15.66
	F-measure $\pm$ (σ)		
Case Study	FeLLaCaM	FLL	FLC
BSH	81.99 $\pm$ 6.86	33.77 $\pm$ 15.10	48.13±11.54
CAF	75.57 $\pm$ 7.62	27.01 $\pm$ 14.64	45.98±13.44
	MCC $\pm$ (σ)		
Case Study	FeLLaCaM	FLL	FLC
BSH	0.76 $\pm$ 0.12	0.36 $\pm$ 0.01	0.31 $\pm$ 0.01
CAF	0.71 $\pm$ 0.14	0.34 $\pm$ 0.01	0.39 $\pm$ 0.18
	AUC $\pm$ (σ)		
Case Study	FeLLaCaM	FLL	FLC
BSH	0.88 $\pm$ 0.07	0.73 $\pm$ 0.17	0.66 $\pm$ 0.14
CAF	0.87 $\pm$ 0.07	0.85 $\pm$ 0.13	0.73 $\pm$ 0.10

nalize the similitude between the feature description and the approved feature realization, but also suggest similitude of the feature description with incorrect feature realizations.

The issues 4-5 (modification of boundary and no-boundary model elements) negatively influence the fitness values of feature commonality and feature modification. Modifications to the boundary information of the model fragment penalize the extraction of model fragments as AMF (see Section 6.2). Consequently, commonality fitness will achieve lower values. In addition, modifications to the model elements within the model fragment (that do not alter the boundary information) penalize the modifications fitness.

Figure 15 shows the occurrences of each issue in the features of the case studies. Note that a particular feature can be affected by several issues simultaneously. The first group of bars shows the occurrences of each issue in the 96 features of the BSH case study, whereas the second group of bars show

	1. Incomplete feature description	**2. Vocabulary mismatch**	**3. Domain terms in the code generation rules**
Feature Description	$DomainTerm_A$	$DomainTerm_A$, $DomainTerm_B$	$DomainTerm_A$, $DomainTerm_B$
Approved Feature Realization	$DomainTerm_A$, $DomainTerm_B$	$DomainTerm_{A'}$, $DomainTerm_B$	$DomainTerm_A$
Code Generation Rules	Implementation terms	Implementation terms	$DomainTerm_B$, Implementation terms
Implementation Code	Implementation terms, $DomainTerm_A$, $DomainTerm_B$	Implementation terms, $DomainTerm_{A'}$, $DomainTerm_B$	Implementation terms, $DomainTerm_A$, $DomainTerm_B$

Figure 14: Illustration of Issues 1-3

the occurrences of each issue in the 121 features of the CAF case study. The third group of bars shows the mean value of occurrences of each issue in the two case studies. The fourth group of bars shows the mean value of occurrences in the case studies of the issues 1-3 (53.67%) and the issues 4-5 (25.36%). In industrial domains like the ones of our industrial partners, issues 1-3 (that negatively influence the feature similarity) are more frequent than issues 4-5 (that negatively influence the feature commonality and modifications).

On the one hand, FFL is vulnerable to issues 1-3 because it is guided by feature similarity. On the other hand, FeLLaCaM is vulnerable to issues 1-3 because it is guided by feature similarity, and to issues 4-5 because FeLLaCaM is also guided by feature commonalities and modifications. One may think that been vulnerable to a lower kind of issues is positive to locate features. However, the results show that FeLLaCaM (vulnerable to 1-3 and 4-5) outperforms FLL (vulnerable to 1-3) in terms of precision. It turns out that although the inclusion of feature commonalities and modifications expose the location to issues 4-5, the resulting multi objective fitness compensates negative influence of issues 1-3. The occurrences of issues 1-3 are more frequent than the occurrences of issues 4-5, and the negative influence of issues 1-3 misguide the location to include in the solutions model elements that do not belong to the approved feature realization.

We measure the precision and recall of the results of our approach against the oracle. The oracle is the feature realization approved by our industrial

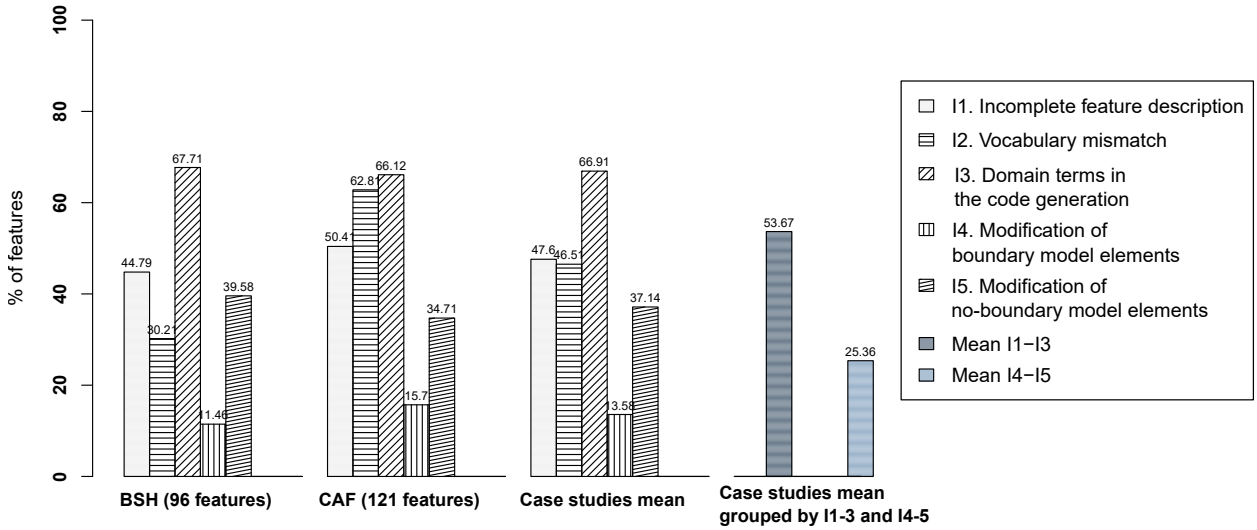

Figure 15: Occurrences of each issue in the features of the case studies

partner. A 100% of precision and 100% of recall indicate that the approach solution is as good as the feature realization provided by the industrial partner. That is, we are evaluating whether the performance of the approach is as good as the performance of the domain experts that approved the feature realizations. This is an improvement with regard to the model fragments obtained mechanically, which are solutions that are not considered as good as the provided by the engineers [8].

However, product models might have better feature realizations than the ones selected by the industrial partners as approved features. Our approach can help look for these better feature realizations, since it provides a ranking of solutions that the user can inspect. The users can also consider the solutions of the ranking as a starting point from where solutions can be manually refined. Furthermore, after inspecting the solutions of the ranking (and the description, commonality and modification values of each solution), the user may refine the feature description and iterate the location of solutions towards feature realizations that achieve better values.

The benefits of using models for software development can be very appealing [36]. However, maintaining a SPL in which the products are specified using models (and where model fragments materialize features) may arise challenges. In particular, metamodels evolve over time, and the conformance between model fragments and the metamodel can be broken. Migration strategies aim to co-evolve models and metamodels together, but their application is currently not fully automated and is thus cumbersome and error prone. In another work, we proposed and evaluated a Variable MetaModel (VMM) strategy to address the evolution of the reusable model fragments of a model-based Software Product Line [37]. Our results indicate that the VMM achieves better results than the migration strategy in domains like that of our BSH industrial partner in terms of indirection, automation, and trust leak. Nevertheless, there are still open issues (e.g., evolutions that turn variabilities into commonalities) that we plan to address in our future work.

Current research on architecture sustainability focuses on addressing guidelines [38, 5] for achieving sustainability, and on methods and metrics for evaluating [39] sustainability. These works are useful to face the challenge of designing SW systems that are supposed to have a long lifespan. The work presented in this paper does not research architecture sustainability in the former sense. Our work explores

a new direction: is it possible to take advantage of already long-living software systems (designed with sustainability in mind) to address the challenge of Feature Location? If these already long-living software systems are fully successful from a suitability point of view, their feature realizations supported changes during its life-cycle while remaining intact [6]. Our feature commonality and modifications fitness values promote model fragments that suffered less modification throughout time.

Model fragment modifications happen even in systems designed with sustainability in mind. However, in these software systems the modification issues are less frequent (and less negative) than the issues suffered by a Feature Location approach guided by feature similarity only. Therefore, we claim that, in the face of already long-living SW systems (designed for sustainability), our ideas improve the Feature Location process.

10. Threats to Validity

In order to identify the threats to validity that are applicable to this work, we have followed the guidelines suggested by De Oliveira et al. [40].

Conclusion validity threats: The first threat of this type that has been identified is *not accounting for random variation*. To address this threat we considered 30 independent runs for each feature with each algorithm. The second threat is the *lack of a meaningful comparison baseline*. We addressed this threat by using a Feature Location through Comparison approach as a standard comparison baseline. The third threat is the *lack of a good descriptive analysis*. In this work, we have used precision, recall, F-measure, MCC, and AUC metrics to analyze the confusion matrix obtained from the experiments; however, other metrics could be applied.

Internal validity threats: The first threat of this type that has been identified is the *poor parameter settings*. As suggested by Arcuri and Fraser [25], default values are good enough to measure the performance of search-based techniques in the context of testing. These values have been tested in similar algorithms for Feature Location [41]. In addition, the choice of the k value in the application of SVD can produce sub-optimal accuracy when using LSA for software artifacts [42]. However, we plan to evaluate all the parameters of our algorithm in future iterations of our work. The second threat is the *lack of real problem instances*. The evaluation of this paper was applied to two industrial case studies from two of our partners, BSH and CAF.

Construct validity threats: The threat of this type that has been identified is the *lack of assessment of the validity of cost measures*. To address this threat, we performed a fair comparison among the algorithms by generating the same number of model fragments and using the same number of fitness evaluations.

External validity threats. The first threat of this type that has been identified is the *lack of a clear object selection strategy*. This threat is addressed by using two industrial case studies from two of our partners, BSH and CAF. Our instances are collected from real-world problems. In addition, we have two different domains (induction hobs and trains) with different size and complexity. Besides, this addressed the second identified threat, the *lack of evaluations for instances of growing size and complexity*.

With regard to what extent it is possible to generalize to other cases, we selected two different long-living industrial domains in order to mitigate this threat. Our approach has been designed to be applied not only to the domains of our industrial partners but also to other different domains. The requisites to apply our approach are that the product family has product model retrospectives, that the set of models where features have to be located conform to MOF (the OMG metalanguage for defining modeling languages), and that feature descriptions must be provided using natural language. Furthermore, the fitness functions can also be applied to any MOF-based model. Our approach extracts the text elements and calculates both the commonalities and the modifications associated to model fragments in an automatic fashion, by using the reflective methods provided by the Eclipse Modeling Framework. That is, there is no need of knowledge about the domain of application in order to calculate the fitness values.

It is also worth noting that the language used for the textual elements of the models and the

provided feature descriptions must be the same. This language is specific to each domain, but as long as both elements are built using the same terminology, LSA will work. Eventually, some tweaks can be applied to narrow the gap between both elements (different tokenizers, stemming, or POS tagging techniques). For instance, the naming conventions used by companies for model elements, properties, and functions, can all follow different formats, but the approach can be tailored to handle them. In our case studies some model elements follow the CamelCase convention, while others follow the Underscore convention. To address that, we applied different tokenizers in order to obtain the terms properly. Nevertheless, even if our approach can theoretically be used to locate features on MOF-based models from different domains, it should be applied to these distinct scenarios before assuring its generalization.

With regard to the kind of models in use, there are different kinds of models in the models paradigm for software development, such as sketches for analysis, models that are reverse engineered from source code, or models for code generation. The architecture of a software system can be described with different kinds of models. However, different kinds of models may contain different amounts of information. Typically, the models used for code generation contain the highest amount of information. In this work, the models of the two case studies are used to generate code by our industrial partners. It is necessary to perform more experiments to know both (1) the performance of the approach as the amount of information in the models decreases, and (2) what is the minimum information amount that an architecture model must contain to successfully apply the approach. This is aligned with a recent systematic review [39] which concludes that using more formal architecture models could provide more automated analysis during architecture analysis.

11. Related Work

Other approaches that are related to this work are classified in three groups taking into account the underlying techniques used to perform the Feature Location.

11.1. Feature Location through Mechanical Comparisons

Some works [10, 11, 9, 12, 13, 43, 44, 14] focus on the location of features over models by comparing the models with each other to formalize the variability among them in the form of a Software Product Line:

- Wille et al. [10] present an approach where the similarity between models is measured following an exchangeable metric, taking into account different attributes of the models. Then, the approach is further refined [11] to reduce the number of comparisons needed to mine the family model.

- The authors in [9] propose a generic approach to automatically compare products and locate the feature realizations in terms of a CVL model. In [12], the approach is refined to automatically formalize the feature realizations of new product models that are added to the system. A similar approach is proposed in [43] where the Feature Location results are validated in an industrial environment.

- Martinez et al. [13] propose an extensible approach based on comparisons to extract the feature formalization over a family of models. In addition, they provide means to extend the approach to locate features over any kind of asset based on comparisons.

- The MoVaPL approach [14] considers the identification of variability and commonality in model variants as well as the extraction of a Model-based Software Product Line (MSPL) from the features identified on these variants. MoVaPL builds on a generic representation of models making it suitable for any MOF-based models.

However, all of these approaches are based on mechanical comparisons among the models, classifying the elements based on their similarity and identifying the dissimilar elements as the features realizations.

In contrast, our work does not rely on model comparisons to locate the features. Specifically, in our work, humans are involved in the search by means an evolutionary algorithm. Domain experts and application engineers become part of the process, contributing their knowledge of the domain in order to tailor the approach by (1) providing the feature description that guides the search and (2) selecting the best realization among the ranking provided as output. The model fragments obtained mechanically are less recognizable by software engineers than those obtained with the participation of software engineers [44].

11.2. Feature Location through Information Retrieval (IR)

There are many Feature Location approaches that have been proposed to find relevant code taking textual information as input [45]. For example, Marcus et al. [46] used IR techniques to map descriptions expressed in natural language (NL) to source code. Cavalcanti et al. [47] used IR techniques to assign change requests in software maintenance or evolution tasks based on context information. Kimmig et al. [48] proposed an approach for translating NL queries to concrete parameters of the Eclipse JDT code query engine.

Recently, several approaches have been proposed to improve the effectiveness of Feature Location. For example, Wang et al. [49] proposed a code search approach, which incorporates user feedback to refine the query. Hill et al. [50] proposed automatically extracting NL phrases to categorize them into a hierarchy in order to help developers to discriminate the relevance of results and to reformulate queries. Zou et al.[51] investigated the "answer style" of software questions with different interrogatives and proposed a re-ranking approach to refine search results.

Other approaches have been proposed to improve the effectiveness of Feature Location by getting information from public repositories [52] or expanding a user query with semantically similar words from websites [53]. For example, Dietrich et al. [54] improved the efficacy of future queries using feedback captured from a validated set of queries and traceability links. Lv et al. [55] enrich each API with its online documentation to match the query based on text similarity.

Even though these approaches improve the effectiveness of Feature Location, they require an additional effort to enrich the code, to keep the documentation synchronized with the changes in the code throughout maintenance and evolution activities, and to incorporate users' feedback. In contrast, our work does not require additional efforts to enrich the code or the query.

11.3. Feature Location through Search-based software Engineering

Lopez-Herrejon et al. [41] evaluate three standard search-based techniques (evolutionary algorithm, hill climbing, and random search) in order to calculate the relationships of a feature model. The authors do not address how each feature is materialized as our work does. Therefore, both works are complimentary: [41] calculates the feature relationships of the feature specification layer, while our work locates the model fragments of the product realization layer (see Figure 1).

Harman et al. [56] performed a survey on the topic of search-based software engineering applied to SPLs. They present an overview of recent articles classified according to themes such as configuration, testing, or architectural improvement. Lopez-Herrejon et al. [57] performed a preliminary systematic mapping study at the connection of search-based software engineering and SPL. They categorized the articles along a known framework for SPL development. These two surveys indicate that search-based software engineering techniques are being applied to SPLs. However, these surveys do not identify works that focus on finding model fragments that materialize the features of the SPL, as our work does.

Font et al. [44] propose a generic approach to locate features among a family of product models based on a human-in-the-loop process. The features are located by comparison of models and the interaction of engineers that provide their knowledge of the domain. The approach is further refined in [58] and generalized through the use of a genetic algorithm to create the model fragments. They introduce a

genetic operator for mutation that can work over a single model fragment and a crossover operator that combines two different product models. The results show that the use of a genetic algorithm allows the approach to provide accurate location of features in spite of inaccurate information on the part of the user.

However, since the work from [58] is designed to locate features by comparisons among the members of a family, the participation of the software engineers is limited and the resultant model fragments are less recognizable to them. In contrast, in this paper, we augment the participation of humans (feature description) with architecture sustainable ideas. Our results show that introducing architecture sustainable ideas in Feature Location contributes to improving the precision results.

12. Conclusions

Since the benefits of SPLs are very appealing but the migration of a product family to a SPL requires deep knowledge and is not straightforward, Feature Location could be useful in easing this migration. In this paper, we present our FeLLaCaM approach for Feature Location. Our approach takes into account the following to locate a feature: the description of the feature to be located, the occurrences across the product model retrospective in which the candidate feature remains unchanged, and the changes made to the candidate features across the product model retrospective.

We have validated our approach in two different long-living industrial domains that have a model-based product family (firmware of induction hobs in the BSH group and control software of trains in CAF). In addition, we have compared the results of our approach with two other approaches for Feature Location: FLL a version of our approach which is guided only by feature descriptions, and FLC which performs comparisons among the different product models. We have measured the performance of the approaches in terms of recall, precision, F-measure, MCC, and AUC using the approved feature realizations as oracle. Our results have shown that FeLLaCaM outperforms FLL and FLC in terms of precision since our approach has reached about 78.61%, whereas FLL has reached about 20.18% and FLC about 43.09%. Moreover, the MCC and AUC measurements have confirmed that our approach provides better results than FLL and FLC.

The presented approach is based on standards as the MOF metalanguage (used to create DSL for describing architectures for the different domains). Therefore, it is generic and can be applied to any domain that fulfills three requisites: (1) The existence of a product family with product model retrospectives where we can benefit from commonality and changes fitness. Usually, these kind of data is found in long-lived systems that have overcome enough evolution. (2) The set of models used to describe the architecture of the system must conform to MOF meta language (see section 2.2). (3) The language used for the textual elements of the models and the feature descriptions provided must be the same. This language is particular for each domain, but as long as both elements are built using the same terminology the LSA will work. Eventually, some tweaks can be applied to narrow the gap between both elements (different tokenizers, stemming or POS tagging techniques).

Our approach goes a step beyond the Feature Location problem because we explicitly introduce feature commonality and modifications throughout the product model retrospective to guide the location of features. In industrial environments where the product family has been developed over decades, our FeLLaCaM approach improves the precision results. Furthermore, current research on architecture sustainability focuses on addressing guidelines [38, 5] for achieving sustainability, and on methods and metrics for evaluating [39] sustainability. Our work explores a new direction: to take advantage of already long-living software systems (designed with sustainability in mind) to address the challenge of Feature Location.

Acknowledgements

This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and

ERDF funds under the project Model-Driven Variability Extraction for Software Product Line Adoption (TIN2015-64397-R). We also thank ITEA3 15010 REVaMP2 Project.

References

[1] P. C. Clements, L. Northrop, Software Product Lines: Practices and Patterns, SEI Series in Software Engineering, Addison-Wesley, 2001.

[2] P. Clements, J. McGregor, Better, faster, cheaper: Pick any three, Business Horizons 55 (2012) 201 – 208.

[3] B. Dit, M. Revelle, M. Gethers, D. Poshyvanyk, Feature location in source code: a taxonomy and survey, Journal of Software: Evolution and Process 25 (2013) 53–95.

[4] A. H. Dutoit, R. McCall, I. Mistrik, B. Paech, Rationale Management in Software Engineering, Springer Publishing Company, Incorporated, 2014.

[5] U. Zdun, R. Capilla, H. Tran, O. Zimmermann, Sustainable architectural design decisions, IEEE Software 30 (2013) 46–53.

[6] R. C. andy Elisa Yumi Nakagawa andy Uwe Zdun andy Carlos Carrillo, Toward architecture knowledge sustainability: Extending system longevity, IEEE Software 34 (2017) 108–111.

[7] T. K. Landauer, P. W. Foltz, D. Laham, An introduction to latent semantic analysis, Discourse processes 25 (1998) 259–284.

[8] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Building software product lines from conceptualized model patterns, in: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015, 2015, pp. 46–55.

[9] X. Zhang, Ø. Haugen, B. Moller-Pedersen, Model comparison to synthesize a model-driven software product line, in: Proceedings of the 2011 15th International Software Product Line Conference (SPLC), 2011, pp. 90–99. doi:10.1109/SPLC.2011.24.

[10] D. Wille, S. Holthusen, S. Schulze, I. Schaefer, Interface variability in family model mining, in: Proceedings of the 17th International Software Product Line Conference: Co-located Workshops, 2013, pp. 44–51. doi:10.1145/2499777.2500708.

[11] S. Holthusen, D. Wille, C. Legat, S. Beddig, I. Schaefer, B. Vogel-Heuser, Family model mining for function block diagrams in automation software, in: Proceedings of the 18th International Software Product Line Conference: Volume 2, 2014, pp. 36–43. doi:10.1145/2647908.2655965.

[12] X. Zhang, Ø. Haugen, B. Møller-Pedersen, Augmenting product lines, in: Software Engineering Conference (APSEC), 2012 19th Asia-Pacific, volume 1, 2012, pp. 766–771. doi:10.1109/APSEC.2012.76.

[13] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, Y. L. Traon, Bottom-up adoption of software product lines: a generic and extensible approach, in: Proceedings of the 19th International Conference on Software Product Line (SPLC), 2015, pp. 101–110. doi:10.1145/2791060.2791086.

[14] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, Y. l. Traon, Automating the extraction of model-based software product lines from model variants (t), in: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, 2015, pp. 396–406. doi:10.1109/ASE.2015.44.

[15] J. Ven, A. Jansen, J. Nijhuis, J. Bosch, Design decisions: The bridge between rationale and architecture, volume Rationale Management in Software Engineering, Springer Berlin Heidelberg, 2006, pp. 329–348.

[16] M. A. Babar, T. Dingsyr, P. Lago, H. van Vliet, Software Architecture Knowledge Management:

Theory and Practice, 1st ed., Springer Publishing Company, Incorporated, 2009.

[17] Ø. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, A. Svendsen, Adding standardized variability to domain specific languages, in: Software Product Line Conference, 2008. SPLC '08. 12th International, 2008, pp. 139–148. doi:10.1109/SPLC.2008.25.

[18] Meta Object Facility, Meta object facility (MOF) version 2.4.1, 2013. Object Management Group (OMG) Specification.

[19] M. Revelle, B. Dit, D. Poshyvanyk, Using data fusion and web mining to support feature location in software., in: ICPC, 2010, pp. 14–23.

[20] D. Liu, A. Marcus, D. Poshyvanyk, V. Rajlich, Feature location via information retrieval based filtering of a single scenario execution trace, in: Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering, ASE '07, ACM, New York, NY, USA, 2007, pp. 234–243. URL: http://doi.acm.org/10.1145/1321631.1321667. doi:10.1145/1321631.1321667.

[21] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, V. Rajlich, Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval, IEEE Transactions on Software Engineering 33 (2007) 420–432.

[22] T. K. Landauer, P. W. Foltz, D. Laham, An introduction to latent semantic analysis, Discourse processes 25 (1998) 259–284.

[23] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation 6 (2002) 182–197.

[24] A. S. Sayyad, J. Ingram, T. Menzies, H. Ammar, Scalable product line configuration: A straw to break the camel's back, in: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, 2013, pp. 465–474. doi:10.1109/ASE.2013.6693104.

[25] A. Arcuri, G. Fraser, Parameter tuning or default values? an empirical investigation in search-based software engineering, Empirical Software Engineering 18 (2013) 594–623.

[26] A. Kotelyanskii, G. M. Kapfhammer, Parameter tuning for search-based test-data generation revisited: Support for previous results, in: 2014 14th International Conference on Quality Software, 2014, pp. 79–84. doi:10.1109/QSIC.2014.43.

[27] Apache opennlp: Toolkit for the processing of natural language text, https://opennlp.apache.org/, 2016. [Online; accessed 7-April-2016].

[28] Snowball : Snowball is a small string processing language designed for creating stemming algorithms for use in information retrieval, http://snowball.tartarus.org/, 2016. [Online; accessed 7-April-2016].

[29] Efficient java matrix library, http://ejml.org/, 2016. [Online; accessed 7-April-2016].

[30] D. Dyer, The watchmaker framework for evolutionary computation (evolutionary/genetic algorithms for java), http://watchmaker.uncommons.org/, 2016. [Online; accessed 7-April-2016].

[31] H. Ishibuchi, Y. Nojima, T. Doi, Comparison between single-objective and multi-objective genetic algorithms: Performance comparison and performance measures, in: IEEE International Conference on Evolutionary Computation, CEC 2006, part of WCCI 2006, Vancouver, BC, Canada, 16-21 July 2006, 2006, pp. 1143–1150. URL: http://dx.doi.org/10.1109/CEC.2006.1688438. doi:10.1109/CEC.2006.1688438.

[32] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms - a comparative case study, in: Proceedings of the

5th International Conference on Parallel Problem Solving from Nature, PPSN V, Springer-Verlag, London, UK, UK, 1998, pp. 292–304. URL: http://dl.acm.org/citation.cfm?id=645824.668610.

[33] S. V. Stehman, Selecting and interpreting measures of thematic classification accuracy, Remote Sensing of Environment 62 (1997) 77 – 89.

[34] T. Fawcett, An introduction to roc analysis, Pattern Recognition Letters 27 (2006) 861 – 874. ROC Analysis in Pattern Recognition.

[35] A. Arcuri, L. Briand, A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering, Softw. Test. Verif. Reliab. 24 (2014) 219–250.

[36] B. Selic, The pragmatics of model-driven development, IEEE Softw. 20 (2003) 19–25.

[37] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Leveraging variability modeling to address metamodel revisions in model-based software product lines, Computer Languages, Systems & Structures 48 (2017) 20–38.

[38] Z. Durdik, B. Klatt, H. Koziolek, K. Krogmann, J. Stammel, R. Weiss, Sustainability guidelines for long-living software systems, in: 2012 28th IEEE International Conference on Software Maintenance (ICSM), 2012, pp. 517–526. doi:10.1109/ICSM.2012.6405316.

[39] H. Koziolek, Sustainability evaluation of software architectures: A systematic review, in: Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS, QoSA-ISARCS '11, ACM, New York, NY, USA, 2011, pp. 3–12. URL: http://doi.acm.org/10.1145/2000259.2000263. doi:10.1145/2000259.2000263.

[40] M. de Oliveira Barros, A. C. Dias-Neto, 0006/2011-threats to validity in search-based software engineering empirical studies, RelaTeDIA 5 (2011).

[41] R. E. Lopez-Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, A. Egyed, An assessment of search-based techniques for reverse engineering feature models, Journal of Systems and Software 103 (2015) 353 – 369.

[42] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, A. D. Lucia, Parameterizing and assembling ir-based solutions for se tasks using genetic algorithms, in: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), volume 1, 2016, pp. 314–325. doi:10.1109/SANER.2016.97.

[43] J. Font, M. Ballarín, Ø. Haugen, C. Cetina, Automating the variability formalization of a model family by means of common variability language, in: Proceedings of the 19th International Conference on Software Product Line (SPLC), 2015, pp. 411–418. doi:10.1145/2791060.2793678.

[44] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Building software product lines from conceptualized model patterns, in: Proceedings of the 19th International Conference on Software Product Line (SPLC), 2015, pp. 46–55. doi:10.1145/2791060.2791085.

[45] B. Dit, M. Revelle, M. Gethers, D. Poshyvanyk, Feature location in source code: a taxonomy and survey., Journal of Software: Evolution and Process 25 (2013) 53–95.

[46] A. Marcus, A. Sergeyev, V. Rajlich, J. Maletic, An information retrieval approach to concept location in source code, in: Proceedings of the 11th Working Conference on Reverse Engineering, 2004, pp. 214–223. doi:10.1109/WCRE.2004.10.

[47] Y. a. C. Cavalcanti, I. d. C. Machado, P. A. d. M. S. Neto, E. S. de Almeida, S. R. d. L. Meira, Combining rule-based and information retrieval techniques to assign software change requests, in: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14, ACM, New York, NY, USA, 2014, pp. 325–330. URL: http://doi.acm.org/10.1145/2642937.2642964. doi:10.1145/2642937.2642964.

[48] M. Kimmig, M. Monperrus, M. Mezini, Querying source code with natural language, in: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE '11, 2011, pp. 376–379. URL: http://dx.doi.org/10.1109/ASE.2011.6100076. doi:10.1109/ASE.2011.6100076.

[49] S. Wang, D. Lo, L. Jiang, Active code search: Incorporating user feedback to improve code search relevance, in: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, ASE '14, 2014, pp. 677–682. URL: http://doi.acm.org/10.1145/2642937.2642947. doi:10.1145/2642937.2642947.

[50] E. Hill, L. Pollock, K. Vijay-Shanker, Automatically capturing source code context of nl-queries for software maintenance and reuse, in: Proceedings of the 31st International Conference on Software Engineering, ICSE '09, 2009, pp. 232–242. URL: http://dx.doi.org/10.1109/ICSE.2009.5070524. doi:10.1109/ICSE.2009.5070524.

[51] Y. Zou, T. Ye, Y. Lu, J. Mylopoulos, L. Zhang, Learning to rank for question-oriented software text retrieval, in: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering (ASE 2015), 2015, pp. 1–11. URL: http://dblp.uni-trier.de/db/conf/kbse/ase2015.html#ZouYLM015.

[52] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, M. Mirakhorli, On-demand feature recommendations derived from mining public product descriptions, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, 2011, pp. 181–190. URL: http://doi.acm.org/10.1145/1985793.1985819. doi:10.1145/1985793.1985819.

[53] Y. Tian, D. Lo, J. Lawall, Automated construction of a software-specific word similarity database, in: IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014, pp. 44–53. doi:10.1109/CSMR-WCRE.2014.6747213.

[54] T. Dietrich, J. Cleland-Huang, Y. Shin, Learning effective query transformations for enhanced requirements trace retrieval, in: 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE), 2013, pp. 586–591. doi:10.1109/ASE.2013.6693117.

[55] F. Lv, H. Zhang, J. g. Lou, S. Wang, D. Zhang, J. Zhao, Codehow: Effective code search based on API understanding and extended boolean model, in: Automated Software Engineering (ASE2015), 2015. URL: http://research.microsoft.com/apps/pubs/default.aspx?id=259612.

[56] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, Y. Zhang, Search based software engineering for software product line engineering: A survey and directions for future work, in: Proceedings of the 18th International Software Product Line Conference - Volume 1, SPLC '14, ACM, New York, NY, USA, 2014, pp. 5–18. URL: http://doi.acm.org/10.1145/2648511.2648513. doi:10.1145/2648511.2648513.

[57] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, L. Linsbauer, A. Egyed, E. Alba, A hitchhiker's guide to search-based software engineering for software product lines, CoRR abs/1406.2823 (2014).

[58] J. Font, L. Arcega, Ø. Haugen, C. Cetina, Feature location in model-based software product lines through a genetic algorithm, in: 15th International Conference on Software Reuse, ICSR 2016, Limassol, Cyprus, 2016.

Carlos Cetina is an associate professor at San Jorge University and the head of the SVIT Research Group. His research focuses on software product lines, variability modeling and model-driven development. Cetina received a PhD in computer science from the Universitat Politècnica de València. Contact him at ccetina@usj.es

Jaime Font is a PhD student in computer science at the University of Oslo and a researcher in the SVIT Research Group at San Jorge University. His research interests include reverse engineering and variability modeling. Font received an MSc in Advanced Software Technologies from San Jorge University. Contact him at jfont@usj.es

Lorena Arcega is a PhD student in computer science at the University of Oslo and a researcher in the SVIT Research Group at San Jorge University. Her research interests are software evolution, variability modeling and models at run-time. Arcega received an MSc in Advanced Software Technologies from San Jorge University. Contact her at larcega@usj.es

Francisca Pérez is an assistant professor in the SVIT Research Group at San Jorge University. Her research interests include model-driven development, variability modeling, end-user development, and collaborative modeling. Pérez received a PhD in computer science from the Universitat Politècnica de València. Contact her at mfperez@usj.es