Feature Location in model-based Software Product Lines through a Genetic Algorithm [⋆]

Jaime Font[1,2], Lorena Arcega[1,2], Øystein Haugen[3], and Carlos Cetina[1]

[1] San Jorge University, SVIT Research Group, Zaragoza, Spain
{jfont,larcega,ccetina}@usj.es
[2] University of Oslo, Department of Informatics, Oslo, Norway
[3] Østfold University College, Department of Information Technology, Halden, Norway
oystein.haugen@hiof.no

Abstract. When following an extractive approach to build a model-based Software Product Line (SPL) from a set of existing products, features have to be located across the product models. The approaches that produce best results combine model comparisons with the knowledge from the domain experts to locate the features. However, when the domain expert fails to provide accurate information, the semi-automated approach faces challenges. To cope with this issue we propose a genetic algorithm to feature location in model-based SPLs. We have an oracle from an industrial environment that makes it possible to evaluate the results of the approaches. As a result, the proposed approach is able to provide solutions upon inaccurate information on part of the domain expert while the compared approach fails to provide a solution when the information provided by the domain expert is not accurate enough.

1 Introduction

A recent survey [2] reveals that most of the Software Product Lines (SPLs) are built following an extractive approach, where a set of existing products is reengineered into a SPL [12]. The resulting SPL is capable of generating the products used as input (among others) with the benefit of having the variability among the products formalized, enabling a systematic reuse.

Several reverse engineering approaches can be used to identify and locate the features [4–6, 14, 16, 18] from the existing product models and formalize them in the form of a model-based SPL (where the features are realized in the form of model fragments). In our previous work [5] we show that Conceptualized Model Patterns to Feature Location (CMP-FL) provides features more recognizable by the engineers that must use them thanks to the inclusion of information from the domain experts into the feature location process.

[⋆] 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 Lines Adoption (TIN2015-64397-R)

However, in CMP-FL the set of possible solutions is too big to be evaluated exhaustively, resulting in the need of very precise information from the domain engineers to accelerate the process. If the information provided is not accurate enough the feature location will fail, not being able to provide the expected solution. When the family of product models is built following clone-and-own techniques, the variability among the products is not always properly documented, resulting in a lack of precise information.

To cope with the above, we propose an approach based on a Genetic Algorithm to Feature Location (GA-FL) among a set of product models. Specifically, we propose new model-based genetic operations capable of working with model fragments: (1) the crossover operation, that combines information from two possible solutions into a single offspring; (2) the mutation operation, that randomly mutates one model fragment (while keeping the consistency with the product model where the fragment was extracted from); (3) a fitness function that evaluates the population of possible solutions and ranks them depending on how they solve the problem and (4) a parent selection operation to find candidates that feed the rest of genetic operations.

We have compared the CMP-FL with GA-FL through the use of an oracle extracted from our industrial partner (BSH), whose induction department produces the firmware for their induction hobs (sold under the brands of Bosch and Siemens) based on a model-based SPL. It turns out that our GA-FL is able to provide the solution expected in scenarios where the CMP-FL fails. When the information provided is accurate, the GA-FL algorithm is able to enrich the set of best solutions produced given that it explores a broader search space.

The rest of the paper is organized as follows: next section presents some background about the domain of our industrial partner and its SPL. In Section 3 we present our approach, the GA-FL. Section 4 compares the presented approach with the best alternative from literature. In Section 5 we discuss some related work. Finally, we conclude the paper.

2 Formalizing the Variability

This section presents the Domain Specific Language (DSL) used by our industrial partner to formalize their products, the IHDSL. It will be used through 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 approach (GA-FL) to formalize the location of the features as reusable model fragments.

2.1 The Induction Hobs Domain Specific Language (IHDSL)

The newest Induction Hobs (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 cost of increasing the software complexity.

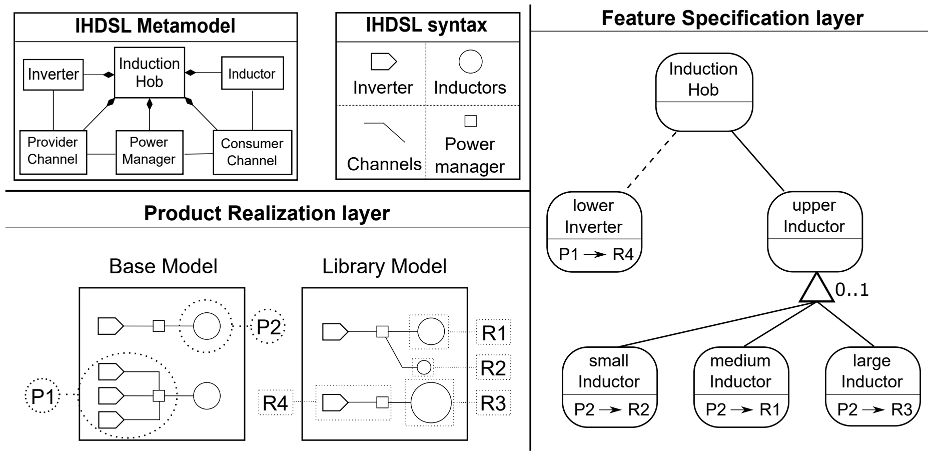

Fig. 1. CVL applied to IHDSL

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 gain legibility and due to intellectual property rights concerns, in this paper we use a simplified subset of the IHDSL (see Fig. 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 runtime. 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 model that captures which model fragments are used by each of the products that can be built from the SPL. To build it, the presented approach uses the Common Variability Language (CVL) [8], given its expressiveness to properly formalize the feature realizations in terms of model fragments. CVL defines variants of a base model conforming to MOF (Meta-Object Facility, the Object Management

Group metalanguage for defining modeling languages) by replacing variable parts of the base model by 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 replacement.

Fig. 1 shows an example of variability specification of IH through CVL. In the product realization layer, two placements are defined 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 fragment has a signature, which is a set of references (boundaries) going from and towards that replacement. A placement can only be replaced by replacements that match the signature. 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.

Through the rest of the paper, we will use the term feature location in models formalized through CVL as “the process of obtaining the particular model fragments (or alternatives e.g. R1, R2 and R3) that are used in a particular placement (or variation point e.g. P1) among a set of products”. Therefore, we will refer to the variation point as the feature being located and each of the alternative model fragments will be referred as different realizations for that feature (in fact, they are realizations of the alternatives of the feature).

3 Genetic Algorithm for Feature Location

This section present our approach, a Genetic Algorithm to Feature Location (GA-FL). Fig. 2 shows an overview of the GA-FL process. The input of the process is a set of interrelated product models with implicit variability among them.

In the Genetic Algorithm process, the set of solutions that will be iterated need to be properly encoded (see **A - Encoding of the Population**), enabling the GA to work with them. The DE (domain expert or domain engineer) provides information about the set of product models to initialize the population of model fragments (see **B - Initialize Population**), the DE will select some product

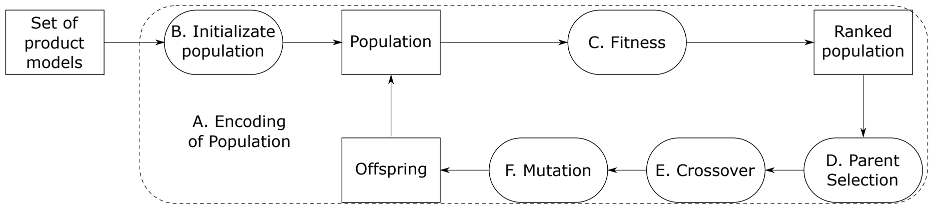

Fig. 2. Overview of the Genetic Algorithm to Feature Location

models to locate a particular feature and an initial model fragment for each of the selected product models. Next, each possible individual from the population is evaluated to determine how good is as a solution to the problem (see **C - Fitness**), as a result the population of solutions is ranked depending on their fitness value. Based on the ranked population, the parents for the new element are randomly selected (see **D - Parent Selection**), giving a higher probability to the solutions with higher fitness values. The first operation applied to the parents is the crossover, that joins two parents into a new solution (see **E - Crossover**). The resulting model fragment will be bound by a different product model and thus will evolve differently than the original one. The second operation applied to the solution resulting from the crossover is the mutation (see **F - Mutation**), the model fragment will evolve, growing or shrinking, resulting in a different model fragment that will be evaluated as possible solution in further generations. Finally, the set of solutions obtained will be presented to the DE, to select the solution that best represent their understanding of the feature being located.

3.1 Encoding of the Population

Traditionally, genetic algorithms encoded each possible solution of the problem (or chromosome) as a fixed-size string of binary values. Each position of the chromosome string (called locus) has two possible values (called alleles): 0 or 1.

However, to encode each model fragment as a string of binary values is not straightforward. As suggested by Davis [3], we decided to use an encoding natural for our problem and then devise a GA for that specific encoding. Therefore, we will encode our individuals as model fragments. To do so, we rely on MOF as the standard to define the models and CVL to specify fragments over those models and manipulate them.

Each individual of our Genetic Algorithm will be a model fragment defined over one of the product models. That is, each individual is a set of model elements and relationships among them that is present in one of the product models (see right part of Fig. 3 to see the representation of the individual). Therefore, to work with these individuals (model fragment defined over a product model), we will present genetic operations that can be applied directly to those model fragments. Through the rest of the paper we will refer to each individual as a model fragment that is always part of a product model.

3.2 Initialize Population

The first step of the process is to initialize the population of the GA. This is done by the DEs, preferably the same DE that created the products or work directly with them. The initialization is done based on DE's knowledge of the domain and the products themselves. This step is performed only one time for each feature that wants to be located.

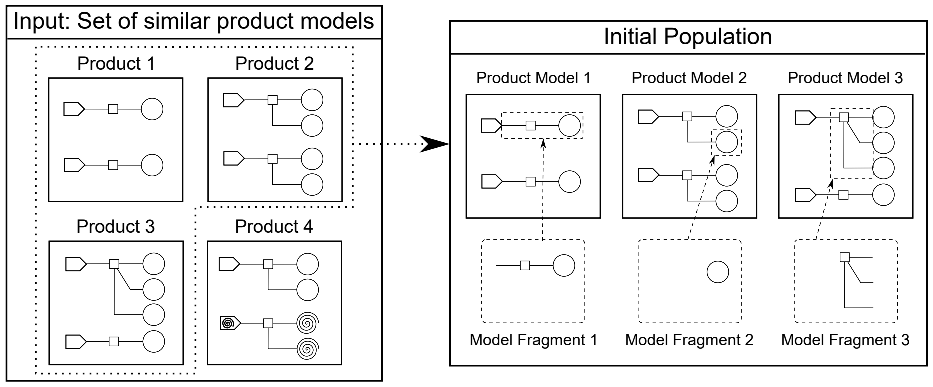

Fig. 3. Initialize Population

Fig. 3 shows an overview of this step. Top part shows the set of similar product models that where the feature will be located (Product Model 1 to 4). First, (1) the DE selects a subset of product models representative of the feature that will be used as input (in this example Product Model 1, 2 and 3), then, (2) for each product model from the subset the DE selects a model fragment that he believes will be part of the realization of that particular feature (Model Fragment 1, 2 and 3). As a result we get an initial population composed of pairs of model fragments and the product models where they were extracted from.

It is important to remark that we focus in the location of the features, leaving out of the scope of this work the features constraints discovery. That is, there could exists a cross-tree constraint among the feature 'upper heating spot' and the feature 'lower heating spot' (e.g. power consumption of combination of several inductors is higher than power consumption of single small inductors), but feature constraint discovery is not covered by this work.

3.3 Fitness Function

The fitness function is used as an heuristic to find the best solutions for the given problem. It is applied to each individual in the population and the function assigns a value that assesses how good is the solution. This information can be used in two ways: to determine that the algorithm should terminate as a desirable level of fitness has been reached and to determine the best candidates as parents for the next generation.

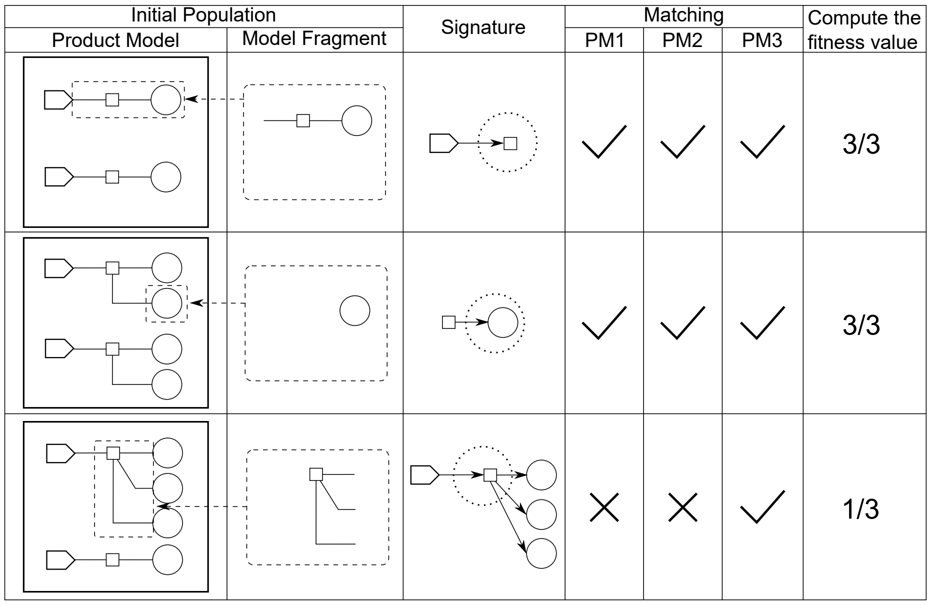

Fig. 4. Fitness function application

Our fitness function proceeds as follows: (1), the process abstracts from each model fragment to a placement signature in their referenced model fragment; (2), placement signatures are compared and grouped together if they are equal; (3), each placement signature is matched against all the product models from the initial subset of product models; (4), the fitness is computed for each placement signature and the fitness values are spread to the elements of the population.

Fig. 4 depicts an overview of the model pattern extraction process [5] adapted to be used as a fitness function. The input of the process is the present population (the set of model fragments and their reference to a product model), see first and second column.

Step 1: The first step (see third column of Fig. 4) is to obtain a placement signature for each of the individuals (model fragment and the product model). The placement signature formalizes the set of elements that must be present in a model in order to connect the given model fragment. This is done comparing the model fragment with the product model from which it was originally extracted (when the initial population was created). The model fragment is present in the product model and connected to other model elements of the product model. The process looks for those boundary elements that link an element from the model fragment with the rest of the product model and extracts them as a placement signature. That is, the set of elements needed to connect the given model fragment. Therefore, the model fragment used as input match this placement signature. As a result, step 1 produces a placement signature for each model fragment used as input.

Step 2: The second step (not shown in Fig. 4, there are no duplicates) is to compare the placement signatures and group the ones that are equal. To do so, the process compares pairwise the placement signatures. If two placement signatures have the same elements in the boundaries, they are considered to be equal. Then, both placement signatures are grouped together. As a result, this step produces a set of unique placement signatures and each model fragment is associated to a single placement signature.

Step 3: The third step (see fourth, fifth and sixth columns of Fig. 4) is to match each placement signature with all the product models present in the initial subset of product models. That is, the process looks for spots where a given placement signature matches in each of the given product models. When a placement signature matches a particular spot of a product model, means that the model fragments associated to that placement signature could be inserted in the given spot. As a result, step 3 provides a set of spots (across all the product models) where the given placement signature matches.

Step 4: The fourth step (see seventh column of Fig. 4) is to compute the fitness value for each of the placement fragments and spread it to the associated model fragments. The process computes the number of product models where the placement matches (no matter how many times). This value indicates the number of product models where the resulting placement could be used. As the purpose of the genetic algorithm is to locate variation points and alternative realizations across the product models, the higher the number of product models that match the better. Finally, the value of each placement signature is spread to the associated model fragments. As a result, step 4 assigns a fitness value for each model fragment present in the population.

After applying these steps, each model fragment gets a fitness value. The higher the number of products where the placement signature is present the better, as this means that it will be able to formalize the variability of a higher number of product models.

Once the population fitness has been assessed, it is time to create the next generation of individuals. This new generation will be based on present generation and the fitness value will be used to ensure that best candidates are chosen as parents for the evolution process. To do so, the process makes use of three different genetic operations that will act over the individuals of the population to generate new ones. First, a selection operation will be used to select the elements that will be used as parents of the new individual. Then, a crossover operation will be used to broad the solution space that a particular solution can reach. Finally a mutation operation will be used to introduce variations in the individual hoping that the new individual performs better than its antecessor.

3.4 Selection of Parents

The selection of parents is performed following the roulette wheel selection method [1], one of the most common methods used in GA. In this method, each individual is assigned with a share of a wheel roulette proportional to their fitness. By doing so, fitter individuals will have higher chance to be selected and go forward with the rest of genetic operations while weaker individuals will have lower probability of being selected. Other selection strategies present in literature can be used with our model fragments, as the operation simply selects individuals, the encoding does not affect the selection.

This operation selects the individuals that will be parents of the new individual that is going to be generated. Traditionally, genetic algorithms select two elements as parents with the only restriction of avoiding the same element being 'father' and 'mother' (as this would nullify the effect of the crossover operation). However, when applying our genetic algorithm to model fragments a new restriction applies: both fragment selected must reference different product models. By doing so we ensure that the crossover operation can be properly applied.

First, we perform the selection of the first parent with no restrictions. Then, when selecting the second parent, we will only allow selections of elements referencing a product model different from the first parent. However, in order to allow the algorithm to browse into a broader search space, the product models not included into the input subset by the DE will be also eligible (with a low fitness value). That is, elements already present in the population will have the fitness value from the previous step while product models not present in the population will have a fitness value of 1.

As a result, the selection operation provides a parent model fragment (obtained from the present population) and another product model (that could not be present in the actual population) that will be used for the crossover operation.

3.5 Crossover

In genetic algorithms, crossover enables the creation of a new individual generated combining the genetic material of both parents. 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 latter, our crossover operation is based on model comparisons. Fig 5 shows an example of application of the crossover operation over model fragments. First we select the model fragment from the first parent. Then we select the product model from the second parent. Then the model fragment (from first parent) is compared with the product model (from the second parent). If the comparison finds the model fragment in the product model, the process 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

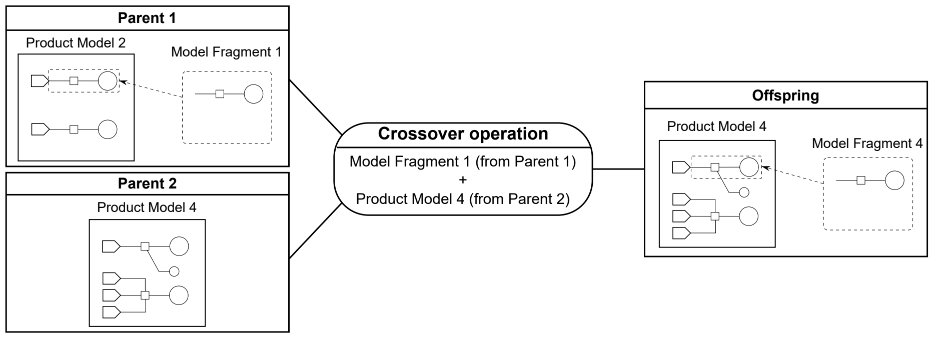

Fig. 5. Crossover Operation

comparison does not find a similar element, the crossover will return the first parent unchanged.

This operation enables to broad the search space to a different product model. That is, both model fragments (the one from the first parent and the one from the new individual) will be the same. However, as each of them is referencing a different product model, they will mutate differently and provide different individuals in further generations. As the solution we are looking for should apply to all the models provided as input, it can be reached from any of them, but some product models can yield to the solution faster than others.

3.6 Mutation

In genetic algorithms, mutation operation introduces a random variation to the new individuals generated by the crossover operation. The mutation operation often results in a weaker individual, but occasionally the result might be a stronger individual.

Fig. 6 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 their associated product model. That is, the model fragment will gain or drop some elements, but the resulting model fragment will be still part of the referenced product model. The mutation possibilities of a given model fragment are driven by their associated product model.

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

Removal of elements: This kind of mutation randomly removes some elements from the model fragment. The only constraint is that elements are selected from the edges of the model fragment (they are connected with a single element), so the resulting model fragment is still connected (we can navigate from any element to any other element through the connections between the elements) and is not split in two isolated groups of elements. As 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 be always present in the referenced product model.

Addition of elements: This kind of mutation randomly adds some elements to the model fragment. The only constraint is that the resulting model fragment is present in the referenced product model. To achieve it, 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.

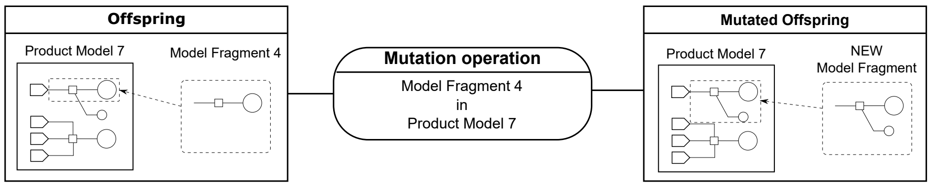

Fig. 6. Mutation operation

As a result, a new model fragment is created but it still references the same product model. That is, the individual represent other possible feature realization present in the product model for the particular feature being located. The next time the fitness is calculated, the placement signature described by this model fragment will be extracted and evaluated to assess how good it is as a solution.

4 Case Study

To evaluate the approach we are going to compare the presented GA-FL approach with CMP-FL, an approach to Feature Location in product models that makes use of the information provided by DEs. We are going to validate the results from both approaches against an oracle obtained from our industrial partner (BSH), the leading manufacturer of home appliances in Europe. Their induction division has been producing induction hobs (under the brands of Bosch and Siemens among others) for the last 15 years. The firmware of the different induction hobs is generated following a model-based SPL approach. First, a resolution for a product is created choosing from the set of features present in the variability model (each feature is formalized as model fragments). Then, a product model is generated by executing the product resolution (CVL execution capabilities produce a product model including the model fragments from the features selected). Finally, the firmware of the induction hob is obtained applying a model transformation to the resulting product model.

4.1 Case Study Setup

Fig. 7 presents an overview of the process followed to evaluate the presented approach. Top part shows the oracle, a set of product models and their formalization of features. The product models from the oracle are used to construct three different scenarios regarding how good is the input fed to the approaches (left part of Fig. 7). Then each scenario is test against both approaches, (CMP-FL) and the presented approach (GA-FL). As a result each approach provides a set of placement signatures that realize the feature being located. Each set of solutions is compared with the placement signature present in the oracle for that particular feature being located (right part of Fig. 7). We want to determine if the solution used by our industrial partner (from the oracle) is present among the solutions provided by each approach in each scenario.

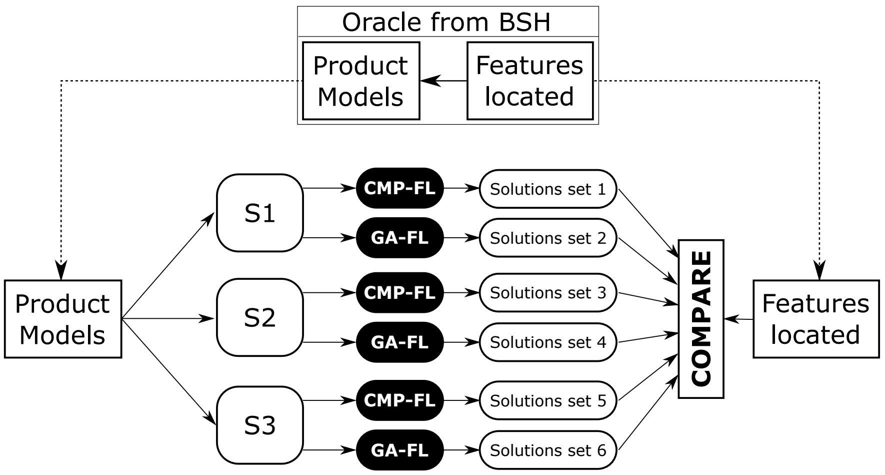

Fig. 7. Overview of the evaluation with the oracle

The oracle is composed of a set of product models and the set of features (used to define the products) properly located. That is, for each feature used by the products (around 100 features) has been previously located and validated by our industrial partner (the oracle is extracted from a set of product models that are currently under production). Therefore, we will consider the oracle as the ground truth for the evaluation process. The set of product models consist of 46 induction hob models, each of them model composed of around 100 elements (on average) that can be part or not of a model fragment. Therefore, the number of possible combinations can be calculated as the power set of the set S of elements $P(S)$, resulting in around 2^{100} ($|P(S)| = 2^n \quad where\,|S| = n$) different potential model fragments. We generate the product models attending to the oracle to distinguish three different scenarios regarding how accurate is the input fed to the approaches:

S1 high accuracy: The first scenario corresponds to what we consider a high accuracy input from the user. More than a 75% of the products used as input for the approaches corresponds to the subset of product models (46 available) that actually include a formalization of the feature that is being located (extracted from the oracle); and thus the placement signature will match with those product models.

S2 medium accuracy: The second scenario corresponds to a medium accuracy input from the user. Between 25% and 75% of the products used as input for the approaches include a formalization of the feature that is being located. Therefore, a similar percentage (25% to 75%) of the products do not contain a formalization of the feature being located.

S3 low accuracy: The third scenario corresponds to a low accuracy input from the user. Only less than a 25% of the products used as input include a formalization of the feature that is being located. This results in some deliberately bad cases (e.g. select only products that do not include the feature being located).

In the three scenarios, the size of the input is randomly selected and ranges from 1 to 5 product models. The seed fragments have been obtained randomly. For each of the features present in the oracle we generate 100 different test cases for each of the three scenarios (S1, S2 and S3). Then, each test case is tested against both approaches (CMP-FL and GA-FL). Finally, the solutions sets (placement signatures) provided by the approaches are compared against the oracle. As a result, we can determine if the feature realizations that is actually being used by our industrial partner (the expected solution) is present among the solution sets returned by the approaches. We do this comparing the placement signature from the oracle with the set of placement signatures provided as solution and determining whether it is present or not.

4.2 Results

The CMP-FL was able to provide a set of solutions that included the expected solution in 86% of the cases from S1 (high accuracy input). Nevertheless, the presented GA-FL was able to include the expected solution in 79% of the cases. The CMP-FL was able to include the expected solution into the solutions set in 48% of the cases from S2 (medium accuracy input). When the information provided by the user is not accurate enough, the approach fails to include the expected (oracle) option into the resulting set. By contrast, the GA-FL was able to include it in 73% of the cases. Finally, the CMP-FL was able to include the expected solution into the solutions set in 16% of the cases from S3 (low accuracy input). The approach only search in the product models provided by the user and is not able to look for the solution in other product models. By contrast, the GA-FL approach was able to include the expected solution in 63% of the cases from S3. Given the stochastic nature of the Genetic Algorithm, the approach is able to find the solution even if the input provided is not accurate.

The justification of the different results provided by both approaches resides in how the search space is traversed. That is, the different elements evaluated as possible solutions by each of the approaches. The CMP-FL approach only explores the portion of the solution space delimited by the product models used as input. In contrast, the GA-FL approach is capable of traversing the entire solution space, independently of the input.

The GA-FL approach is capable of reaching any possible solution from the search space, as it can move across the search space in any direction. The mutation enables the exploration of solutions within the same product, while the crossover operation enables to switch to another product (an further explore it with subsequent random mutations). By contrast, the CMP-FL approach is bounded by the input of the user and only explores solutions within the product provided as input; thus, some areas of the search space cannot be reached.

As a result, the CMP-FL is not able to provide better results than the input provided; that is, upon a 75% of accuracy will provide the expected result 75% of the cases. In particular in all the cases where the accuracy was 0% (from S3) the expected solution was not included. In contrast, the presented approach is able to explore solutions beyond the input provided by the user. This means that upon the scenarios where the input is not accurate enough, the crossover operation will (eventually) be able to switch to different product models that convey to the expected solution.

5 Related Work

Some works report their industrial experiences in a wide range of fields transforming legacy products into Product Line assets [10, 11, 13]. These approaches focus on capturing guidelines and techniques for manual transformations. In contrast, our approach introduces automation into the process while taking advantage from the knowledge of the domain experts.

Other works focus on the automation of the extraction process [6,9,14,16–18], obtaining the variability from legacy products by comparing the products with each other. In [17], the similarity between models is measured following an exchangeable metric, taking into account different attributes of the models. Then, the approach is further refined [9] to reduce the number of comparisons needed to mine the family model. Rubin et al. [16] propose a generic framework for mining legacy product lines and automating their refactoring. They compare the elements of the input with each other, matching those whose similarity is above a certain threshold and merging them together. The authors in [18], propose a generic approach to automatically compare products and extract the variability among them in terms of a CVL variability model. The authors in [14] propose an approach based on comparisons to extract the variability of any kind of asset. However, these approaches are based on mechanical comparisons, automatically turning identical elements into common parts of the SPL, similar elements as alternatives for a feature and unmatched elements into optional features. In contrast, our work enables the DE to decide which elements should be formalized as part of a feature based on the results of the comparisons.

Finally, there are some research efforts that apply genetic algorithms to the SPLs domain. For instance, the authors in [7] present GAFES, an artificial intelligence approach for optimized feature selection in SPLs. The authors in [15] present a genetic algorithm that finds optimal configurations of a Dynamic SPL at run-time. However, the solutions of those genetic algorithms are encoded as strings of binary values specifying the presence or absence of each feature. By contrast, our approach is applied directly to the product models and model fragments, resulting in a different encoding and set of genetic operations customized to work with model fragments.

6 Conclusion

In this paper we have presented a Genetic Algorithm to Feature Location (GA-FL) approach. To the best of our knowledge it is the first Genetic Algorithm applied to feature location over models. We have provided a custom encoding that enable the GA to work with model fragments and a set of genetic operations that can be applied to individuals following that encoding. We have presented a fitness function, a parent selection operation, a crossover operation (capable of bring together elements from two parents into a single offspring) and a mutation operation (that produces slight variations of the individual being mutated).

Finally we have compared the presented GA-FL with CMP-FL in terms on how both approaches traverse the search space. This comparison shows that CMP-FL does not traverse the whole space, failing to find a solution under some scenarios, while the GA-FL is capable of traversing the whole search space reaching the solutions. In addition, in scenarios where the CMP-FL approach is able to find the best solution, our GA-FL approach is also able to do so while traversing more elements from the search space, providing a more complete solution.

The ideas of the presented approach are generic and can be applied to any MOF-based models. Our next steps will involve the application of the presented GA-FL approach to CAF [4], an international company that builds and deploy railway solutions. They are currently shifting to a model-based SPL approach and there is a need of locating the features among their existing product models.

References

1. M. Affenzeller, S. Winkler, S. Wagner, and A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications.* Chapman & Hall/CRC, 1st edition, 2009.
2. T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski. A survey of variability modeling in industrial practice. In *7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, 2013.
3. L. Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[4] www.caf.es/en

4. B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2013.
5. J. Font, L. Arcega, Ø. Haugen, and C. Cetina. Building software product lines from conceptualized model patterns. In *Proceedings of the 19th International Conference on Software Product Line (SPLC)*, pages 46–55, 2015.
6. J. Font, M. Ballarín, Ø. Haugen, and 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)*, pages 411–418, 2015.
7. J. Guo, J. White, G. Wang, J. Li, and Y. Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *J. Syst. Softw.*, 84(12):2208–2221, Dec. 2011.
8. Ø. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen. Adding standardized variability to domain specific languages. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 139–148, Sept 2008.
9. S. Holthusen, D. Wille, C. Legat, S. Beddig, I. Schaefer, and 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*, pages 36–43, 2014.
10. K. Kim, H. Kim, and W. Kim. Building software product line from the legacy systems "experience in the digital audio and video domain". In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 171–180, Sept 2007.
11. R. Kolb, D. Muthig, T. Patzke, and K. Yamauchi. Refactoring a legacy component for reuse in a software product line: A case study: Practice articles. *J. Softw. Maint. Evol.*, 18(2):109–132, Mar. 2006.
12. C. W. Krueger. Easing the transition to software mass customization. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 282–293, London, UK, UK, 2002. Springer-Verlag.
13. H. Lee, H. Choi, K. Kang, D. Kim, and Z. Lee. Experience report on using a domain model-based extractive approach to software product line asset development. In *Formal Foundations of Reuse and Domain Engineering*, volume 5791 of *Lecture Notes in Computer Science*, pages 137–149. Springer Berlin Heidelberg, 2009.
14. J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and 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)*, pages 101–110, 2015.
15. G. G. Pascual, M. Pinto, and L. Fuentes. Self-adaptation of mobile systems driven by the common variability language. *Future Generation Computer Systems*, 47:127 – 144, 2015. Special Section: Advanced Architectures for the Future Generation of Software-Intensive Systems.
16. J. Rubin and M. Chechik. Combining related products into product lines. In *Fundamental Approaches to Software Engineering*, volume 7212, pages 285–300. Springer Berlin Heidelberg, 2012.
17. D. Wille, S. Holthusen, S. Schulze, and I. Schaefer. Interface variability in family model mining. In *Proceedings of the 17th International Software Product Line Conference: Co-located Workshops*, pages 44–51, 2013.
18. X. Zhang, Ø. Haugen, and 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)*, pages 90–99, 2011.