

Feature Location in Models through a Genetic Algorithm Driven by Information Retrieval Techniques

Jaime Font^{1,2}

¹Universidad San Jorge
SVIT Research Group
Autovía A-23 Km. 299
50830 Zaragoza, Spain
{jfont,larcega,cetina}@usj.es

Lorena Arcega^{1,2}

²University of Oslo
Department of Informatics
Postboks 1080 Blindern
0316 Oslo, Norway
oystein@ifi.uio.no

Øystein Haugen³

³Østfold University College
Faculty of Computer Science
Postboks 700
1757 Halden, Norway
oystein.haugen@hiof.no

Carlos Cetina¹

ABSTRACT

In this work we propose a feature location approach that targets models as the feature realization artifacts. The approach combines Genetic Algorithms and Information Retrieval techniques. Given a model and a feature description, model fragments extracted from the model are evolved using genetic operations. Then, Formal Concept Analysis is used to cluster the model fragments based on their common attributes into feature realization candidates. Finally, Latent Semantic Analysis is used to rank the candidates based on the similarity with the feature description. As a result, the genetic algorithm evolves the population of model fragments to find the set of most suitable feature realizations. We have evaluated the approach with an industrial case study, locating features with precision and recall values around 90% (baseline obtains less than 40%). Finally, we provide recommendations on how to provide the input to the approach to improve the location of features over the models.

1. INTRODUCTION

Feature location (FL) is known as the process of finding the set of software artifacts that realize a particular feature. The topic has gained momentum during recent years [22, 6] and several research works focus on creating and improving methods to locate the features. FL is one of the main activities performed during software evolution [6] and up to an 80% of a system's lifetime is spent on the maintenance and evolution of the system [19].

However, most of the research on FL has been directed towards the location of features into source code artifacts, neglecting other software artifacts such as the models. In addition, the approaches that perform FL over models [10, 8, 20, 26, 27, 21] are designed to generate the variability specification over a family of models and built a Software Product Line from them and it is not possible to apply them to isolated models. Therefore, there is a lack of proper FL

techniques that can be applied to locate the model elements that belong to a feature that has to be evolved or maintained.

In this work we propose FLM (Feature Location in Models), an FL approach that targets models as the realization artifacts and does not rely in model comparisons, but on Information Retrieval (IR) techniques. The approach is based on a Genetic Algorithm (GA) that generates alternative model fragments that can be the realizations of the feature being located. Then, we use Formal Concept Analysis (FCA) [11] to cluster the model fragments by their common attributes and to generate feature candidates. The feature candidates are assessed comparing them to a search query that describes the target feature by using Latent Semantic Analysis (LSA) [17] to measure the similarity between both. As a result, feature candidates can be ranked based on the distance with the feature description, allowing the best ones to be selected to engenderate the next generations. When the improvement of the generations is stalled, the resulting feature candidates are provided as output.

The presented approach has been supported by a tool within the Eclipse environment and applied to the product models obtained from our industrial partner BSH, one of the largest manufacturers of home appliances in Europe. Its induction division has been producing Induction Hobs (sold under the brands of Bosch and Siemens) for the last 15 years. The firmware for their products is generated from models using a model-based approach. The application of the approach shows that the values of recall and precision higher than 85%. Finally, we provide some recommendations on how to provide the input to the approach to improve the location of features over the models.

The rest of the paper is structured as follows: Section 2 provides some background. Next, Section 3 provides the details of the presented approach. Section 4 presents the evaluation performed. Then, Section 5 discusses the approach. Finally, Section 6 presents some related work and the paper is concluded.

2. BACKGROUND

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 (FLM) to formalize the model fragments used

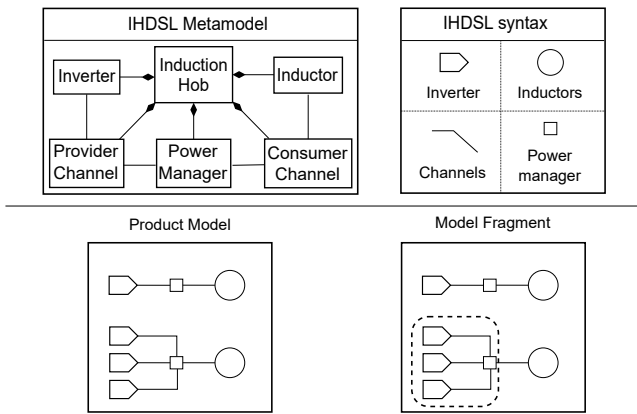


Figure 1: IHDSL product model and model fragment formalization

as feature candidates.

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 energy consumption of the IH. All of these changes are being possible at the cost of increasing the software complexity and thus require several modifications into the models used to formalize the products.

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 top part 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 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.

Bottom left part of Figure 1 depicts an example of a product model specified with the IHDSL. The product model contains four inverters used to power two different inductors. The upper inductor is powered by a single inverter while the lower inductor is powered by the combination of three different inverters. Power managers act as hubs to perform the connection between the inverters and the inductors.

To formalize the model fragments used by the approach we use the Common Variability Language (CVL) [13, 24], given its capabilities to formalize the feature realizations in terms of model fragments. CVL defines variants of a base model (conforming to MOF, the OMG metalanguage for defining modeling languages) by replacing variable parts of the base model (the features) by alternative model replacements (the feature realizations) found in a library.

Bottom right part of Figure 1 shows an example of a

model fragment of the product model (bottom-left part). The model fragment includes the three inverters in charge of powering the lower inductor along with the three provider channels and the power manager used to aggregate and manage the power provided by those inverters. This model fragment is the realization of the feature “triple inverter”.

3. FEATURE LOCATION IN MODELS (FLM)

This section presents FLM, the proposed approach for feature location in product models. The objective of the approach is to provide the subset of elements from a given product model that realize a particular feature being requested by the user. To do so, the approach relies on a Genetic Algorithm that iterates a population of model fragments and evolves them using genetic operations. As output the approach provides a list of feature candidates that might be realizing the feature. This list is ranked taking into account the information provided by the user as input.

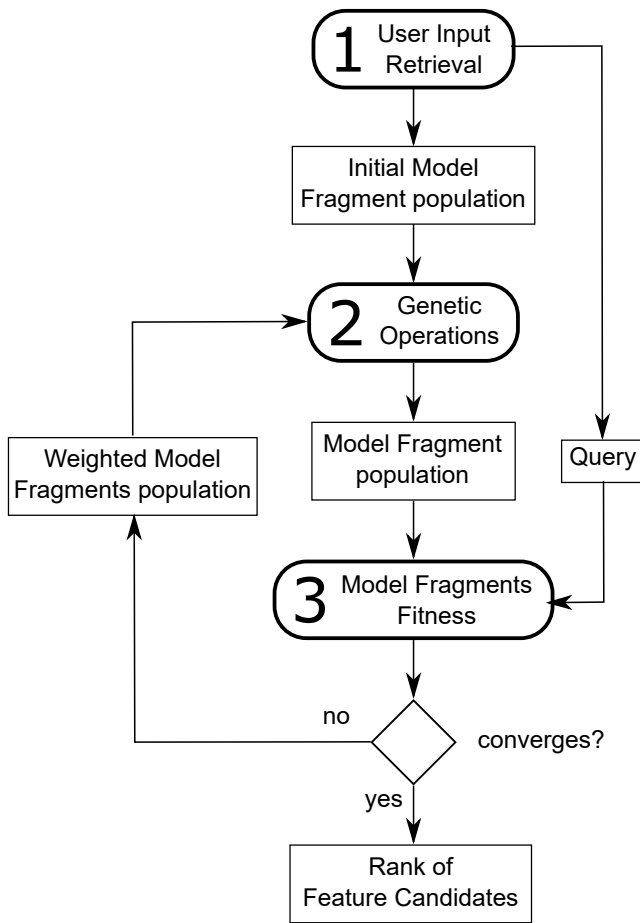


Figure 2: Feature Location in Models Overview

Figure 2 presents an overview of the approach. Rounded boxes represent the different steps of the approach while rectangular boxes represent the inputs and outputs of each of the steps. Lines indicate that an element is an input or output of one of the steps.

The input of the approach is the product model where the feature is going to be located. Then, the user provides a description of the target feature in terms of an initial seed

fragment and a textual description of the feature. The initial seed and the product model are used to generate some candidate fragments. Then, those candidates are assessed taking into account the textual description of the feature being located. These two steps (generation and assessment) are repeated until some stop condition is met. When the stop condition is fulfilled the process returns the list of fragment candidates ranked according to the assessments.

3.1 User Input Retrieval

The first step is to gather input for the feature location. The input will consist of the product model and information about the target feature provided by the user. In particular, the user will provide a seed fragment and a textual description of the feature.

A **seed** fragment of the target feature is an element or set of elements that the engineer believes that could be part of the feature being located. To do so, the engineer applies his knowledge of the domain and the product models to point to some elements that will be used as the starting point of the process.

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 similar to those used when specifying the product models. The knowledge of the engineers about the domain and the product models will be useful to select the textual description from the sources available.

Figure 3 presents an example of input for the approach. Left part presents the seed fragment proposed by the user (a model fragment of the product model where the feature is going to be located). The user believes that the selected inductor is going to be part of the feature realization. Then, the right part of the figure shows a textual description for the feature being located, the hotplate. It is a simplified version of a text description that has been extracted from the internal documentation used by our industrial partner to describe their products.

The textual description provided by the user is turned into a query by using some well established IR techniques:

- First, the textual description is tokenized (divided into words). Usually a white space tokenizer can be applied (that splits the strings whenever it finds a white space) but for some sources 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 can be following CamelCase naming.
- Secondly, we apply the Parts-of-Speech (POS) tagging technique. POS tagging analyses the words grammatically and infers the role of each word into the text provided. As a result, each word is tagged enabling 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 commonly used and do not contribute with relevant information that describe the feature, so they are removed.
- Thirdly, stemming techniques are applied to unify the language used in the text. This technique consists of

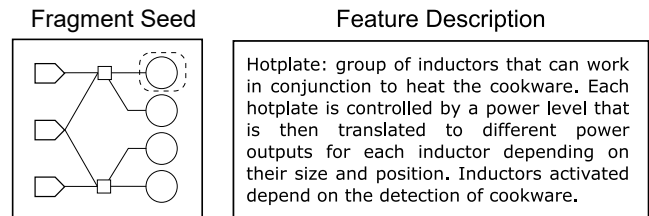


Figure 3: Input provided to the approach

reducing each word to its roots enabling that different words referring to similar concepts can be grouped together. For instance, plurals are turn into singulars (inductors to inductor) or verbs tenses are unified (using and used are turn into use).

The User Input Retrieval step generates as a result an initial population of fragments (that contain the model fragment provided as seed) and the query that will be used for the comparisons (obtained from the textual description). Then, the model fragment from the initial population will be evolved into several model fragments through the use of the genetic operations.

3.2 Genetic Operations

The second step is to generate a set of model fragments that could be realizing the feature. The generation of model fragments is done by applying genetic operators adapted to work over model fragments. That is, new fragments based on the existing ones (the seed fragment during the first execution) are generated through the use of three genetic operators: the selection of parents, the mutation and the crossover.

In order to apply the genetic operators, it is first necessary to apply the **selection operator** that selects the best candidates from the population to be the input for the rest of operators. There are different methods that can be used to perform the selection of the parents, but one of the most spread choices is to follow the wheel selection mechanism [4]. That is, each model fragment from the population has a probability of being selected proportional to their fitness score. Therefore, candidates with high fitness values will have higher probabilities of being chosen as parents for the next generation. Top part of Figure 4 shows an example of application of the selection operator.

The **crossover operator** is used to imitate the sexual reproduction followed by some living beings in nature to breed new individuals. That is, two individuals mix their genomic information to give birth a new individual that holds some genetic information from one parent and some from the other one. This could make him adapt better (or worse) to his living environment depending on the genetic information inherited from his parents.

Following this idea, our crossover operator applied to model fragments takes as input two model fragments and a randomly generated mask to combine them into two new individuals. The mask determines how the combination is done, indicating for each element of the model fragments if the offspring should inherit from one parent or the other (including the element or not if the element is present on the parent or not). A model fragment is a subset of the elements present in a product model. As both model fragments have

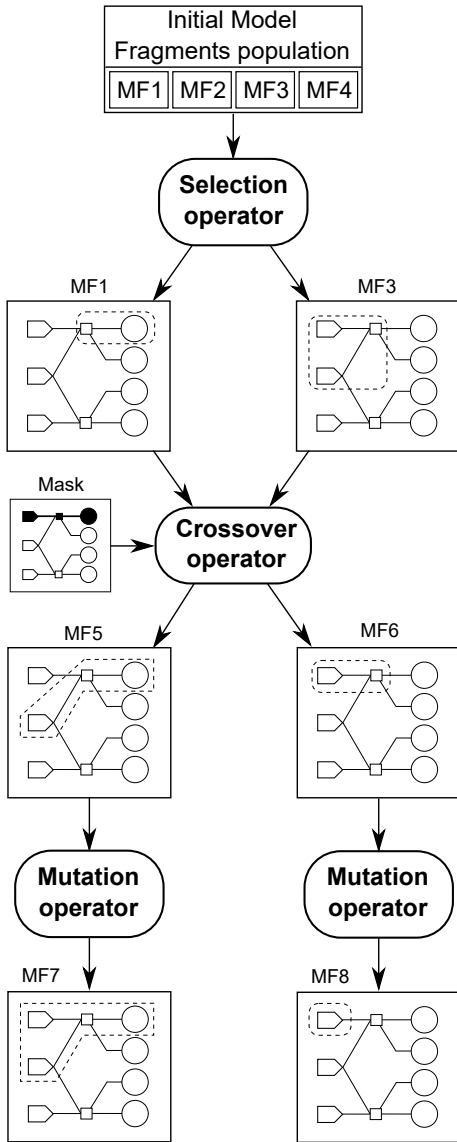


Figure 4: Genetic Operators: Mutation and Crossover over Model Fragments

been extracted from the same product model the combination (applying the mask) of them will always return a model fragment that is part of the product model. As a result, two individuals will be generated, one by applying directly the mask and another one by applying the inverse of the mask as it is usually done in genetic algorithms [5].

Figure 4 shows an example of application of the crossover operator. The input of the operator is the first parent (MF1), a mask indicating two sets of elements (one regular and one marked in black) and the second parent (MF3). To create the first of the new individuals we interpret the mask selecting the blacked out elements from the first parent (MF1) and the regular elements from the second parent (MF3). That is, the elements on the top part of the product model that are in black in this mask are selected depending on whether they are part of MF1 or not, while the rest of the elements that are not blacked out in the mask are selected depend-

ing on whether they are part of MF3 or not. As a result, the new MF5 contains some elements from the first parent (power group connected to the inductor) and some others from the second parent (the inverter that connects with the power group).

In addition, the mask is also interpreted in the opposite way, selecting the blacked out elements from the second parent and the regular elements from the first parent. This produces MF6 (see middle-right part of Figure 4), where an inverter connected to a power manager has been inherited from the second parent (MF3) and nothing has been inherited from parent 1 (MF1) as all the elements not blacked out in the mask are not part of MF1.

For the crossover operation to work, it is not necessary to have elements shared by both parents. It is possible to perform crossovers that return fragments where not all the elements are connected. Indeed, the feature being located could be realized by several model elements that are not directly connected in the model. Therefore, it is necessary to create this kind of fragments as they could be the ones realizing the target feature.

The **mutation operator** is used to imitate the mutations that randomly occur in nature when new individuals are born. That is, a new individual holds a small difference in regards to its parents that could make him adapt better (or worse) to their living environment.

Following this idea, the mutation operator applied to model fragments takes as input a model fragment and mutates it into a new one produced as output. As the approach is looking for fragments of the product model that realize a particular feature, the new modified fragment must remain being a part of the product model. Therefore, the modifications that can be done to the model fragment are driven by the product model. In particular, the mutation operator can perform two kind of modifications, addition of elements to the fragment, or removal of elements from the model fragment.

Bottom part of Figure 4 shows two examples of application of the mutation operator. Left part shows the first example, MF5 is used as input of the operator that produces MF7 as output. In this example, the mutation operation has added some elements (a new inverter connected to the power manager). The resulting model fragment remains being part of the product model that is driving the mutation, so it is a candidate as realization of the feature. Right part shows the second example, where MF6 is used as input and MF8 is produced as output. In this example the mutation operator has removed an element (the power manager).

3.3 Model Fragment Fitness

The third step of the process consists of the assessment of each candidate fragment produced and the ranking of them according to a fitness function. The fitness function is used to imitate the different degrees of adaptation to the environment that different individuals have. Therefore, individuals that result of mutations and crossovers that contribute to their adaptation to the environment will have higher chances of survival than others.

Following this idea, the fitness function is used to determine the suitability of each candidate as solution to the problem, enabling to rank them from the best candidate to the worst. The fitness function is based on the comparison between the feature description query and the identi-

fier names and other natural language items present in the model fragments. The input of this step is a population of candidate fragments, and the feature description query; the output produced is a ranking where each candidate has been assigned with a fitness value.

However, when locating features realized through model fragments, it is important to notice that a feature can be realized by the combination of more than one model fragment. Therefore, as part of our fitness function we will follow two steps: (1) the population of fragments will be grouped according to their similarities in terms of the domain, then (2) each one of these groups will receive a fitness value obtained using the feature description obtained from the user as part of step one.

3.3.1 Grouping of Model Fragments into Feature Candidates through FCA




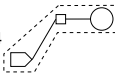

To perform the grouping of model fragments into feature candidates we rely on Formal Concept Analysis (FCA) [11], a branch of mathematical lattice theory that provides means to identify meaningful groups of objects that share common attributes. Groupings are identified by analysing a binary relationship between the set of all objects and all attributes. FCA takes as input a formal context (an incidence table indicating which attributes are possessed by each object) and returns a set of concepts where every concept is a maximal collection of objects that share some common attributes. Each concept will be considered as a feature candidate.

Therefore, in order to apply FCA we need to define a set of objects (model fragments), a set of common attributes (the metamodel elements used to build those model fragments) and a binary relationship between them (the presence or absence of a particular metamodel element in the model fragment). Then, a formal context that represents the relationship between the objects and the attributes can be built.

Top of Figure 5 shows an example of a formal context relating model fragments and the metamodel elements used to build them. Columns show each of the attributes present in the context, in this case the different metamodel elements used to build the model fragments. Rows show each of the objects of the context, in this case the different candidate model fragments present in the population. Each cell indicates if a particular metamodel element has been used to build each of the model fragments. For instance, MF1 and MF2 (first and second rows) are built using three different metamodel elements (power manager, consumer channel and inductor), while MF4 (fourth row) is built using all the elements from the metamodel (Inductor, Inverter, Provider Channel, Consumer Channel and Power Manager).

Using the formal context as input, FCA generates a lattice: a set of interrelated concepts where every one is a maximal collection of model fragments that share common metamodel elements. Bottom of Figure 5 shows the lattice obtained applying FCA to the formal context presented before. Each of the circles represents one concept (there are seven in total). The concepts are labeled with the metamodel elements (grey background labels) and the model fragments (white background labels) grouped by that concept. The concepts are organized hierarchically, indicating containment relationships between the sets of model fragments and metamodel elements of the concepts.

That is, the set of model fragments of a concept is con-

	Inverter	Provider Channel	Power Manager	Consumer Channel	Inductor
MF1 			✓	✓	✓
MF2 			✓	✓	✓
MF3 	✓	✓	✓		
MF4 	✓	✓	✓	✓	✓
MF5 	✓	✓	✓		
...

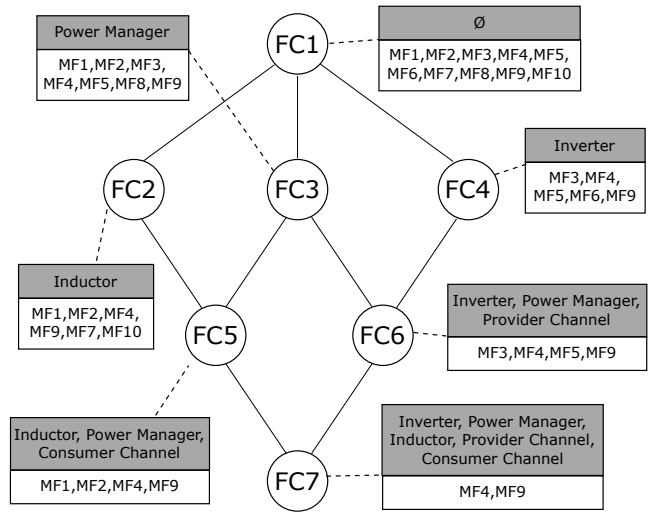


Figure 5: Formal context between model fragments and metamodel elements and Lattice obtained through FCA

tained by all the connected concepts above it and contains all the model fragments from connected concepts below it. For instance, the model fragments in FC6 (MF3, MF4, MF5, MF9) will be also part of all concepts above FC6 (FC4 and FC1). Likewise, the metamodel elements in FC3 (Power Manager) will be also part of all concepts below it (FC5, FC6 and FC7).

As a result of the application of the FCA, a set of Feature Candidates (FC1, FC2, FC3, FC4, FC5, FC6 and FC7) that clusters some of the model fragments based on their use of the elements of the metamodel is provided.

3.3.2 Feature Candidates assessment through LSA

To assess the relevance of each feature candidate with relation to the query extracted from the textual description provided by the user, we are going to apply methods based on Information Retrieval (IR) techniques. In particular we apply Latent Semantic Analysis (LSA) to analyse the relationships between the description of the feature provided by the user and the candidate features previously obtained.

LSA constructs vector representations of a query and a

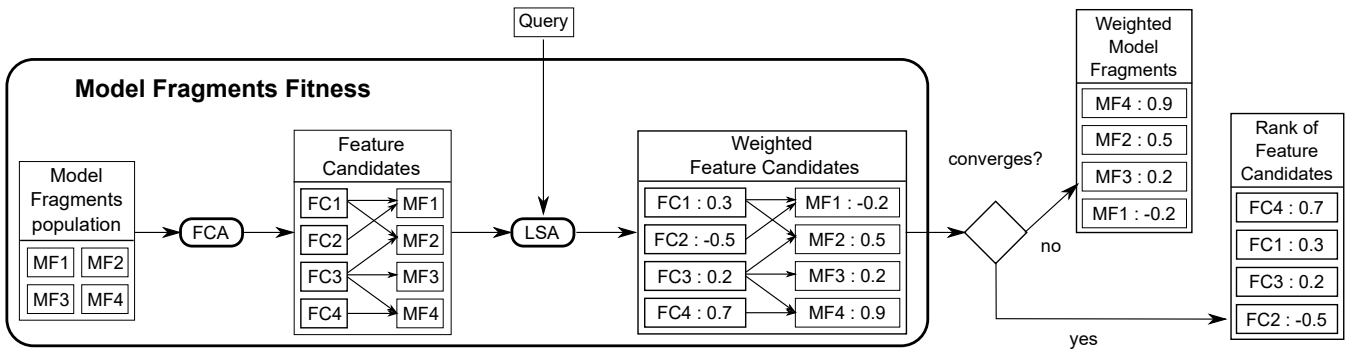


Figure 6: Model Fragment Fitness

corpus of text documents by encoding them as a term-by-document co-occurrence matrix. That is, a matrix where each row corresponds to terms and each column corresponds to documents, followed by the query in the last column. Then, each cell holds the number of occurrences of a term (row) inside a document or the query (column).

Once the matrix is built, it is normalized and decomposed into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [17]. One vector that represents the latent semantic is obtained for each document and the query. Finally, the similarities between the query and each document are calculated as the cosine between both vectors, obtaining values between -1 and 1.

We apply LSA to the feature candidates generated by FCA and the query. A document of text is generated from each of the feature candidates using the model fragments contained by the feature candidate. That is, the names and values of properties and methods are processed to extract the terms by applying Natural Language Processing techniques (as performed with the textual description provided in the first step, see Section 3.1). As a result we obtain a list of relevant terms present in the documents and the query. Finally, after the matrix is turned into vectors and the cosines are calculated, we obtain a value for each of the feature candidates indicating its similarity with the query.

	FC1	FC2	FC3	FC4	FC5	FC6	FC7	Q
Inverter	0	2	5	7	2	5	2	0
Provider	0	2	5	5	2	5	2	0
Power	0	4	7	4	4	4	2	2
Consumer	0	5	5	2	5	2	2	0
Inductor	0	10	5	2	5	2	2	3
Manager	0	4	7	4	4	4	2	0
Channel	0	7	10	7	7	7	4	0
...

Figure 7: Term-by-document co-occurrence matrix for Feature Candidates

Figure 7 shows an example of co-occurrence matrix for our running example. Each column is one of the Feature Candidates obtained through the application of FCA. Then,

the last column is the query provided by the user as part of the input of the process. Each row is one of the terms extracted from the corpuses of text conformed by all the feature candidates and the query itself (we show the terms before the stemming process to improve the readability). Each cell shows the number of occurrences of each of the terms into the feature candidates.

3.3.3 Loop

The next step is to spread the similarity values obtained by each feature candidate to the model fragments contained by that feature candidate. However, each model fragment can be part of more than one feature candidate. Therefore, in order to obtain the similarity of a model fragment with the query we need to combine the similarity values obtained by each of the feature candidates where the model fragment is present. As a result each model fragment is assigned with a value (fitness value).

Figure 6 shows an example of the assessment process. First, the set of model fragments from the population is used to build a set of feature candidates through FCA. Then, the set of feature candidates is compared with the query through the use of LSI, resulting in a set of weighted feature candidates. At this point, if the stop condition is met, the process will stop returning the rank of feature candidates. If the stop condition is not met yet, the genetic algorithm will keep its execution one generation more.

The next time that the genetic operators are applied, it will be necessary to select the best candidates as parents for the new generation. This will be done based on the score obtained by each model fragment. As a result, model fragments with higher similarities will have more chances to be selected as parents of the new generation. Notice that being part of more feature candidates does not guarantee a higher score for the model fragment, as the similarity between a feature candidate and the query can be negative.

The process of generation of fragments, extraction of feature candidates and assessment of those candidates is repeated until the stop condition is met. Usually, the stop condition can be a time slot, a fixed number of generations or a trigger value of the fitness that makes the process finish when reached. In addition, it is also possible to monitor the fitness values and determine when they are converging and no further improvements are being made by new generations. The stop condition highly depends on the domain and the problem being solved; therefore, it is adjusted depending on the results being outputted by the process.

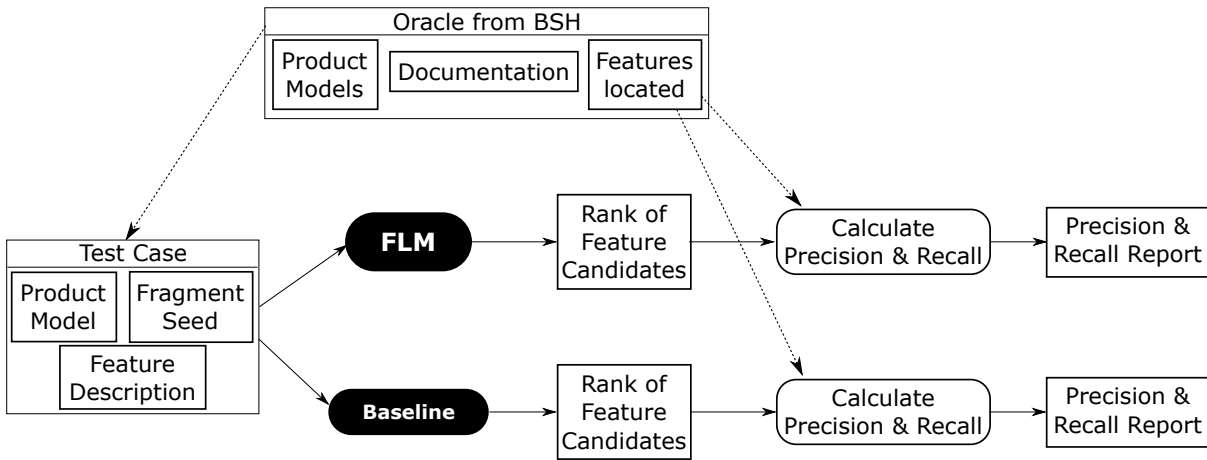


Figure 8: Evaluation Overview

4. EVALUATION

To evaluate the approach we applied it to a case study extracted 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 for the last 15 years. The firmware of the different induction hobs is generated following a model-based Software Product Line approach.

We are going to use the product models from BSH as an oracle to evaluate the presented approach. That is, we make use of a set of products models whose feature realizations are known beforehand and will be considered as the ground truth, enabling us to compare the results provided by the approach with the oracle.

4.1 Setup

Figure 8 shows an overview of the process followed to evaluate the presented approach. Top part shows the oracle for the case study, a set of product models, features located over those product models, and documentation obtained from the model-based SPL of our industrial partner. Those features correspond to products that are currently being sold or will be released to the market in the near future. This oracle will be considered the ground truth and will be used to evaluate the presented approach.

The oracle is composed of 46 induction hob models where each product model is composed of more than 500 elements on average. For each of the 96 features used to build the product models we got a test case including a product model, a fragment seed (extracted from the located feature in the oracle) and a feature description (obtained from the documentation of the features).

Then, each test case was fed as input for two different executions: the presented approach where the Genetic Algorithm fitness is performed through FCA and LSI (FLM); the same Genetic Algorithm but using a random fitness function to assign random values to each model fragment instead of using IR techniques (Baseline). As a result we got a pair (one for FLM and one for Baseline) of Feature Candidates rankings for each of the test cases.

Finally, we computed the precision, recall and F-measure values (three of the most common measures for information retrieval methods [23]) for each of the Feature Candidates

rankings, obtaining a precision and recall report. Precision measures the number of elements from the solution that are correct according to the ground truth (the oracle), while recall measures the number of elements of the solution that are retrieved by the proposed solution. F-measure combines precision and recall into a single value and corresponds to the harmonic mean of precision and recall.

To calculate the precision and recall we need to compute the true positives (TP); the number of elements in the solution that are actually correct according to the ground truth (the oracle). That is, the number of elements that are present in both, the solution and the ground truth. The precision is calculated dividing the TP by the total number of elements in the solution. The recall is calculated dividing the TP by the total number of elements in the ground truth.

In our case, each feature candidate from the rankings is a model fragment composed of a subset of the model elements present in the product model (where the feature is being located). The granularity will be at the level of model elements, so each model element present in both (the solution feature candidate and the located feature from the oracle) will be a TP.

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.

The presented approach has been implemented within the Eclipse environment. We have used Eclipse Modeling Framework (EMF) to manipulate the models from our industrial partner, and the Common Variability Language to manage the fragments of models. The genetic algorithm is built upon Watchmaker Framework for Evolutionary Computation [7] that enable us to implement our own genetic operators. Regarding the IR techniques, we have used colibri-java [12] to

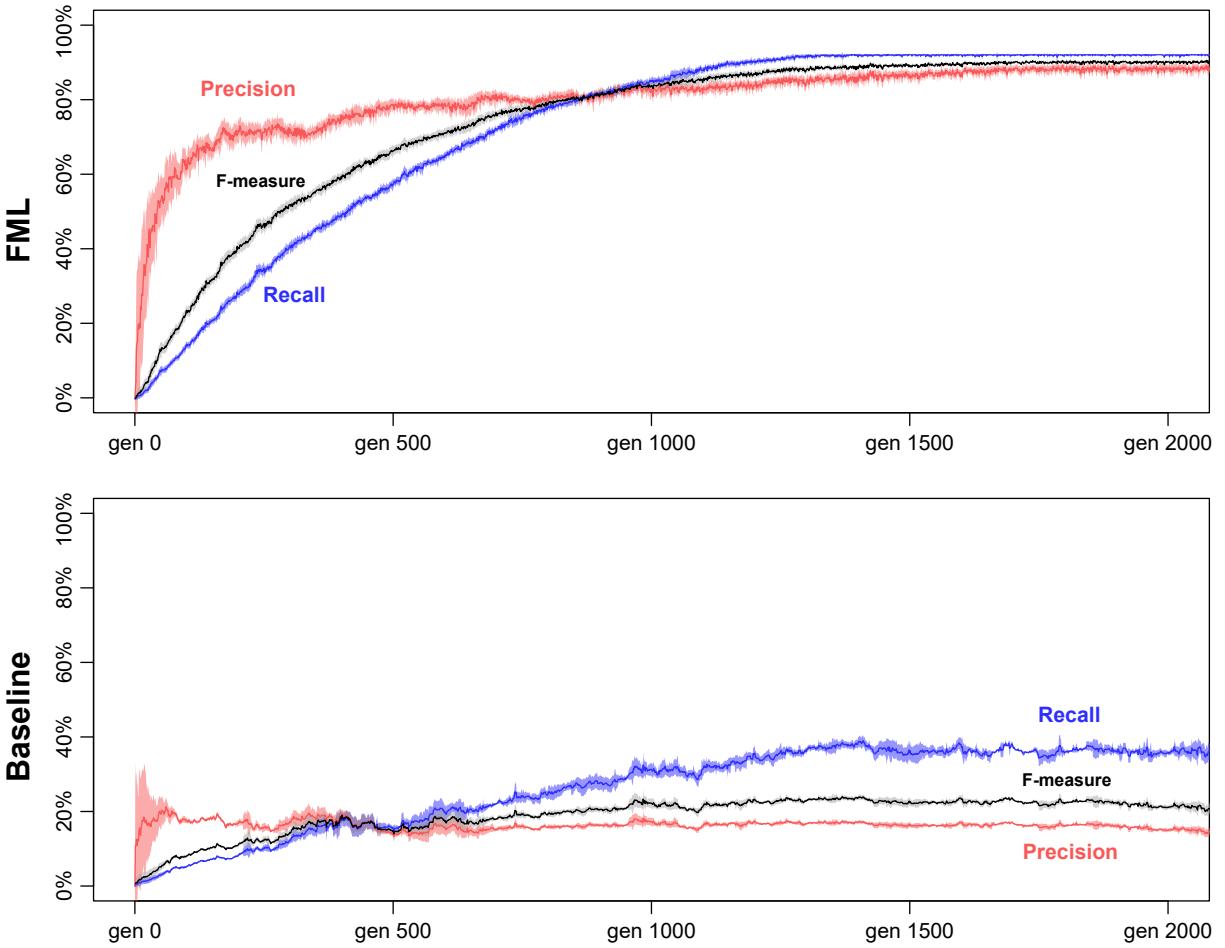


Figure 9: Mean Precision, Recall and F-measure for FLM and the Baseline

implement the FCA. The IR techniques used to process the language have been implemented using OpenNLP [2] for the POS-Tagger and Snowball [3] for the stemming. Finally, the LSI has been implemented using the Efficient Java Matrix Library (EJML [1]).

The evaluation has been executed using a Dell XPS with a processor Intel(R) Core(TM) i7-2670QM @2.2GHz with 8 GB or RAM and running Windows 10 Pro N 64 bits as the hosting Operative System. The approach has been executed under Java(TM) SE Runtime Environment (build 1.8.0_73-b02).

4.2 Results

Figure 9 shows the mean precision and recall values measured for the 96 features located by both executions (the presented approach and the Baseline). Top chart shows the results for the execution of the presented approach while bottom part shows the results for the Baseline. The values for the recall measure are in blue, the values for the precision measure are in red and the values for the F-measure are in black (in both charts). Each measure includes the standard deviation (shaded in the same color). The x axis indicates the number of generations of the genetic algorithm while the y axis measures the % value of the recall, precision and F-measures.

Each of the lines corresponds to the mean values for the

location of the 96 test cases obtained from the oracle. First, we have calculated the values for each of the test cases (including all the feature candidates from their rank). Then, mean values and standard deviations for the 96 test cases have been calculated.

The recall values for the presented approach (top chart blue line) start in a range between 0% and 20% for the first hundreds of generations but then start raising up to the 90% (around generation 1.400). Beyond generation 1.400, the recall values keep close to the 100%. The precision values for the presented approach (top chart red line) start in a range between 0% and 60% for the first hundreds of generations. Then, the precision values raise up to the range between 80% and 90% (around generation 1.500), beyond that generation there are no further changes in the tendency.

The recall values for the Baseline (bottom chart blue line) start in a range between 0% and 20% for the first hundreds of generations. Then, the recall values reach the range between 30% and 40% (around generation 1.400) and oscillate in that range for the rest of the generations. The precision values for the Baseline (bottom chart red line) raise sharply to the 20% and then drop slightly to a value around 15%, remaining steady for the rest of generations.

Overall, results show that the use of IR techniques as the fitness function of the GA (our approach) guides it to locate the feature better than if a random guide is provided (Base-

line). The comparison with the oracle enables to obtain the recall and precision values for both approaches and the IR provides higher mean values of precision and recall for any number of generations.

5. DISCUSSION

The evolution of the recall and precision values over the generations suggests that the proposed fitness function is performing well and guiding the algorithm to find feature realization candidates close to the target feature.

5.1 Input data

The presented approach relies on two pieces of information given by the engineer performing the feature location, the **seed** fragment and the **query**. These two elements will have an impact on the ranking of feature candidates produced and must be chosen carefully by the engineer performing the feature location.

To test out the impact of the **seed** fragment in the results, we have executed the approach with different kind of seed fragments containing one element (single element belonging to the feature being located or single element not belonging to the feature being located). But those executions did not produce noticeable differences in the resulting ranking of feature candidates or in the number of generations needed to converge.

However, when selecting seed fragments of sizes closer to the size of the feature being located, the effect is noticeable. The number of generations needed by the GA to converge was reduced when a seed fragment close to the feature being located was chosen. In particular, when the fragment seed contained about 50% of the elements belonging to the feature being located, the number of generations needed for the GA to converge was reduced up to 15%.

Overall, when the engineer provides a seed that is not related with the target feature, the recall and precision of the feature candidates returned is not affected. Therefore, our recommendation is that when the engineer is pondering whether to include a model fragment seed or not, he should do it. The approach is capable of accepting more than one model fragment as seed at the same time and we have performed some test (up to 10 related and unrelated seeds) that reveals that there is no negative impact when unrelated seeds are included.

To test out the impact of the **query** in the results, we have also executed the approach varying the text description used as input (using longer and smaller queries by subsetting the original description, including more or less domain terms and including more or less meta-element terms).

The search **query** used to locate the feature is in charge of driving the search and greatly impacts on the precision and recall results. In fact, depending on the level of detail of the query, the recall and precision values obtained will change. When the query provided is too broad, the precision decreases as there are several model elements matching the query not belonging to the target feature. Anyhow, the elements belonging to the feature will be also matched positively so the recall value will be high. However, when the query provided is too specific, some of the elements relevant for the feature being located can be missed out. Thus, the recall value is decreased although the precision values remain high.

To achieve good precision and recall values, it is important

to avoid the usage of words included into the meta-elements of the model elements. That is, if we refer to the meta-class name of one of the model elements, all instances of this class will match to that word (e.g. any inductor class model element will match the query “inductor”). By contrast, by using words specific for the model element (as the value of the name property or values of some of the parameters contained in those model elements), those model elements (and not others with the same class) will be included into the feature candidates, affecting positively to the precision values (e.g. only some inductors will match the query “doubleTwistedCoil” as it is the value of a property of the inductor class). In fact, when removing the usage of meta-element names in the queries, the approach obtained similar values of recall but the precision raised up to a 20% for best cases.

It is important to notice that we have made use of an oracle (obtained from our industrial partner model-based SPL and considering the ground truth) to evaluate the approach using test cases where the expected solution was known beforehand. By doing so, we were able to compute the recall, precision and F-measure for the feature candidates rankings provided by the approach. However, when applying the presented approach to locate features (and thus not having an oracle), the approach should be used iteratively, refining the query and the seed fragment as described above.

5.2 Scalability

The presented approach has been executed to locate 96 features over 46 product models of sizes ranging from around 400 elements to around 600 elements (around 500 elements on average). Each feature being located has a relative size in the range between 3% and 10% of the product model where the feature is being located. The mean number of generations needed to locate those features (when the genetic algorithm converges) is about 1500 generations.

The time needed to locate the features ranged between 12 seconds and 26 seconds. That is, the 1.500 generations were generated in an average time of 19 seconds. Most of the time (around 85%) was spent on the execution of the fitness function while the rest was used to process the query (3%) and execute the genetic operations (12%). The approach is able to reach a million of generations within less than 5 minutes when locating features over models with dimensions similar to the models of our industrial partner.

The implementation of the approach is far from being optimized. Furthermore, the computer used to run the case study is a four years old laptop. Therefore, the performance of the approach could be increased by some means if necessary.

5.3 Generalization

The presented approach has been designed to be applied, not only to our industrial partner domain, but to any domain. The only requisite to apply the approach is that the set of models where features have to be located conform to MOF (the OMG metalanguage for defining modeling languages). The query must be provided as a textual description.

The generation and management of fragments is performed using the Common Variability Language (CVL), which can be applied to any MOF-based languages. With the use of CVL, the approach is able to work with the model fragments

provided as seed and evolve them applying the genetic algorithms. As output, the approach produces a set of feature candidates rankings in the form of CVL model fragments.

Furthermore, the fitness function can also be applied to any MOF-based model. The text elements associated to the models are extracted automatically by the approach 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 extract the relevant terms.

However, the approach can be tailored to fit the needs of different domains if necessary. For instance, the naming conventions used by companies for model elements, properties and functions can follow different formats, but the approach can be tailored to handle them. In our case study 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. Similarly, the Part-of-Speech tagger that is used to eliminate non-relevant words based on their grammatical category is language dependant, but can be configured to other languages when necessary.

In summary, the approach can be applied to locate features on any MOF-based model from any domain. If necessary, some tweaks and modifications can be applied to tailor the approach to particular needs of the domains, but the core of the approach will remain unchanged.

6. RELATED WORK

Some works report their industrial experiences in a wide range of fields transforming legacy products into Product Line assets [15, 16, 18]. 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.

Some works [25, 14, 26, 27, 20, 10, 8] 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. [25] 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 [14] to reduce the number of comparisons needed to mine the family model.

The authors in [26] propose a generic approach to automatically compare products and locate the feature realizations in terms of a CVL model. In [27] the approach is refined to automatically formalize the feature realizations of new product models added to the system. A similar approach is proposed in [10] where the feature location results is validated against an industrial environment.

Martinez et al. [20] 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.

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 is applied to a single product model, so it does not rely on model comparisons to locate the features but in comparisons

with a textual description of the target feature.

Font et al. [8] 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 [9] and generalized through the use of a genetic algorithm to locate features among a family of models in the form of a variation point (model placements and a set of corresponding replacements).

However, the work from [9] cannot be applied to a single model as the fitness function is based on the number of occurrences of one model fragment among the family of models. In the present work, the feature location is performed over a single model; no family of models is required to apply the approach. We introduce a new crossover operator (that applies to model fragments extracted from the same product model) and a fitness function that combines FCA and LSA to determine the similarity between the query provided and the “evolving” model fragments. The approaches in [9] and the present work differ on (1) the scenarios where they can be applied (variability formalization of a family of products for SPL creation VS isolation of a feature realization from a single product for maintenance purposes), (2) the models that can be fed to the approach (family of models VS single model with meaningful identifier names and other natural language items present in the models), (3) the fitness function (occurrences of model fragment VS textual similarity between a search query and the model elements performed through FCA+LSA) and (4) the evaluation performed (impact of model fragments seed VS impact of the query used as input).

7. CONCLUSION AND FUTURE WORKS

As part of this work we have presented a Genetic Algorithm to Feature Location that target models as the feature realization artifacts. We propose a new crossover operator that combines two model fragments extracted from the same product model and generates a new individual that contains elements from both parents. We propose a fitness function that clusters model fragments into feature candidates and then assigns them a fitness value based on their similarity with a query. As a result, the features located by using the approach shows a recall and precision measures of around 90% while the baseline remains below 40%. Finally, we discuss the approach and provide recommendations on how to provide the input to the approach to improve the location of features over the models.

Our next steps involve the application of the presented approach to other domains. In particular we plan to apply the approach to locate the features on train models from CAF¹, an international company that builds and deploys railways solutions around the world.

Acknowledgment

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).

¹<http://www.caf.net/en>

8. REFERENCES

- [1] Efficient java matrix library. <http://ejml.org/>. [Online; accessed 7-April-2016].
- [2] Apache opennlp: Toolkit for the processing of natural language text. <https://opennlp.apache.org/>, 2016. [Online; accessed 7-April-2016].
- [3] 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].
- [4] 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.
- [5] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
- [6] 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.
- [7] D. Dyer. The watchmaker framework for evolutionary computation (evolutionary/genetic algorithms for java). <http://watchmaker.uncommons.org/>, 2016. [Online; accessed 7-April-2016].
- [8] 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.
- [9] J. Font, L. Arcega, Ø. Haugen, and 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, Jun 2016.
- [10] 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.
- [11] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1997.
- [12] D. Götzmann. Formal concept analysis implemented in java (colibri-java). <https://code.google.com/archive/p/colibri-java/>, 2016. [Online; accessed 7-April-2016].
- [13] Ø. Haugen, B. Møller-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.
- [14] 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.
- [15] 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.
- [16] 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.
- [17] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.
- [18] 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.
- [19] M. M. Lehman, J. Ramil, and G. Kahen. A paradigm for the behavioural modelling of software processes using system dynamics. Technical report, Imperial College of Science, Technology and Medicine, Department of Computing, Sep 2001.
- [20] 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.
- [21] 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.
- [22] J. Rubin and M. Chechik. A survey of feature location techniques. In I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, and J. Bettin, editors, *Domain Engineering*, pages 29–58. Springer Berlin Heidelberg, 2013.
- [23] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.
- [24] A. Svendsen, X. Zhang, R. Lind-Tviberg, F. Fleurey, Ø. Haugen, B. Møller-Pedersen, and G. K. Olsen. Developing a software product line for train control: a case study of cvl. In *14th international conference on Software product lines (SPLC)*, 2010.
- [25] 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.
- [26] X. Zhang, Ø. Haugen, and B. Møller-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.
- [27] X. Zhang, Ø. Haugen, and B. Møller-Pedersen. Augmenting product lines. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, volume 1, pages 766–771, Dec 2012.