Building Software Product Lines from Conceptualized Model Patterns

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

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

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

[3]Østfold University College
Department of Information Technology
Postboks 700
1757 Halden, Norway
oystein.haugen@hiof.no

ABSTRACT

Software Product Lines (SPLs) can be established from a set of similar models. Establishing the Product Line by mechanically finding model differences may not be the best approach. The identified model fragments may not be seen as recognizable units by the application engineers. We propose to identify model patterns by human-in-the-loop and conceptualize them as reusable model fragments. The approach provides the means to identify and extract those model patterns and further apply them to existing product models. Model fragments obtained by applying our approach seem to perform better than mechanically found ones. It turns out that the repetition of a fragment does not guarantee its relevance as reusable asset for the SPL engineers and vice versa, a fragment that has not been repeated yet, may be relevant as a reusable asset. We have validated these ideas with our industrial partner BSH, an induction hobs manufacturer that generates the firmware of their products from a model-driven SPL.

CCS Concepts

•Software and its engineering → Software product lines;

Keywords

Reverse Engineering, Model-based Software Product Lines, Variability Identification, Human-in-the-loop

1. INTRODUCTION

Software Product Lines (SPLs) aim at reducing development cost and time to market while improving quality of software systems by exploiting commonalities and variabilities across a set of software applications [10]. The SPL engineering paradigm separates two processes; domain engineering (where the variability of the SPL is defined and realized) and application engineering (where specific software products are derived by reusing the variability of the SPL) [2].

The extractive approach to SPLs capitalizes on existing systems to initiate a product line [7], formalizing variability among a set of similar products into a variability model. However, manually spotting the commonalities and variability among the set of product models may become cumbersome and error prone [1], especially as the number of models and its complexity increases.

Some reverse engineering model differencing approaches [15, 11] aim to automatically extract and formalize the variability among a set of similar product models. By performing several comparisons among the product models, commonalities and variabilities are identified and formalized as variability models. As a result, a variability model is automatically built from a set of similar products.

We have applied a model differencing approach to build a SPL around a set of product models from our industrial partner. However, the results are not the reusable model assets expected by our industrial partner, due to the lack of domain engineering knowledge embedded in the process; model assets that are automatically extracted may not match the expectations of domain experts and application engineers.

We propose a human-in-the-loop differencing approach to identify and extract reusable model patterns from a set of similar product models. Domain experts and application engineers become part of the decision-making process, contributing their knowledge of the domain to tailor the approach. Then, the model patterns are extracted and formalized into variability models. Finally, model patterns can be applied to the set of models, resulting in a formalization of the variability in terms that are relevant for the engineers.

We have validated the presented ideas with our industrial partner (BSH group), the induction division has been producing induction hobs (under the brands Bosch and Siemens among others) over the last 15 years. The Induction Hobs domain is facing big changes, which has triggered the creation of a Software Product Line around the Induction Hob models that already exist. The first attempt follows a model differencing approach; then we apply our human-in-the-loop approach. As a result, we have identified the main situations where domain experts prefer a model asset that is different to the one proposed by automatic approaches.

The rest of the paper is structured as follows: next section introduces some background about the Common Variability Language and our industrial partner's domain. Section 3

SPLC '15 June 20–24, 2015, Nashville, TN, USA
© 2015 ACM. ISBN 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

motivates the approach with an example extracted from our experience. Section 4 presents our approach to identify and extract resusable model patterns from a set of similar product models. In section 5 we present our experience applying the approach to our industrial partner's domain. Section 6 discusses related work. Finally we conclude the paper.

2. THE INDUCTION HOBS DOMAIN

Traditionally, stoves have a rectangular shape and feature four rounded areas that become hot when turned on. Therefore, the first Induction Hobs (IHs) created provided similar capabilities. However, the induction hobs domain is constantly evolving and, due to the possibilities provided by the induction phenomena and the electronic components present in the induction hobs, a new generation of IHs has emerged.

For instance, the newest IHs feature full cooking surfaces, where dynamic heating areas are automatically calculated and activated or deactivated depending on the shape, size, and position of the cookware placed on top. 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 being possible at the cost of increasing the software complexity.

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

Inverters are in charge of converting the input electric supply to match the specific requirements of the induction hob. Specifically, the amplitude and frequency of the electric supply needs to be precisely modulated in order to improve the efficiency of the IH and to avoid resonance. Then, the energy is transferred to the hotplates 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 are composed of a conductor that is usually wound into a coil. However, inductors vary in their shape and size, resulting in different power supply needs in order to achieve performance peaks. Inductors can be organized into groups in order to heat larger cookware while sharing the user interface controllers. Each group of inductors can have different particularities; for instance, some of them can be divided into independent zones or others can grow in size adapting to the size of the cookware being placed on top of them. Some of the groups of inductors are made at design time, while others can occur at runtime (depending on the cookware placed on top).

2.1 The Common Variability Language applied to Induction Hobs

The Common Variability Language (CVL) [3, 13] was recommended for adoption as a standard by the Architectural Board of the Object Management Group and is our industrial partner's choice for specifying and resolving variability.

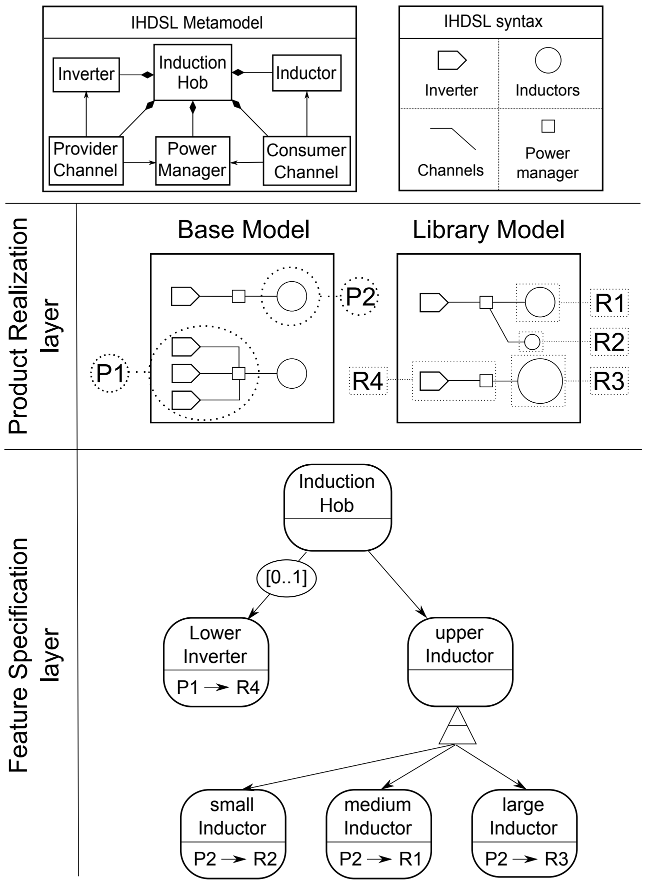


Figure 1: CVL applied to IHDSL

CVL defines variants of a base model (conforming to MOF) by replacing variable parts of the base model by alternative model replacements found in a library model.

The variability specification through CVL is divided across two different layers: the feature specification layer (where variability can be specified following a feature model syntax) and the product realization layer (where variability specified in terms of features is linked to the actual models in terms of placements, replacements and substitutions).

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 (hereinafter 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 (hereinafter replacement). Similarly to placements, a replacement can be any element or set of elements that can be used as variation for a replacement.

CVL defines variants of the base model by means of fragment substitutions. Each substitution references to a placement and a replacement and includes the information necessary to substitute the placement by the replacement. In other words, each placement and replacement is defined along with its boundaries, which indicate what is inside or outside each fragment (placement or replacement) in terms of references among other elements of the model. Then, the substitution is defined with the information of how to link the boundaries of the placement with the boundaries of the replacement. When a substitution is materialized, the base model (with placements substituted by replacements) continues to conform to the same metamodel.

Figure 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 previously defined. 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 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.

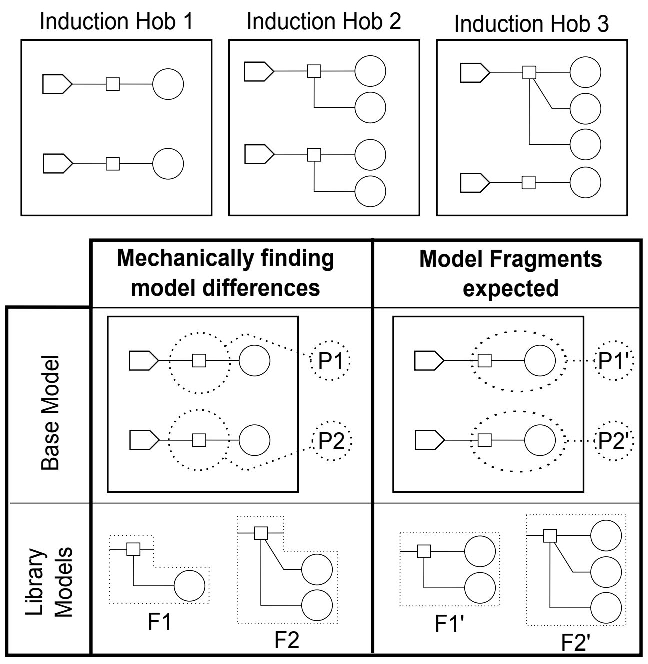


Figure 2: Motivation of the approach

3. MOTIVATION OF THE APPROACH

Reverse engineering approaches [11, 15] rely on mechanically finding model differences among the models. First, several comparisons among the product models are performed. Then, a set of model fragments is extracted based on the differences and common parts spotted among the models. Identical elements are extracted as common parts of the product line, similar elements are extracted as variable alternative parts, and unmatched elements are extracted as variable optional parts. As a result, the variability existing among the set of similar product models is formalized.

However, we have detected an issue when applying reverse engineering approaches of this kind to extract and formalize the variability existing among the IH product models of our industrial partner. Specifically, fragments obtained by these approaches do not match the expectations of our industrial partner. Figure 2 illustrates the issue experienced.

The top part of Figure 2 shows a representation of three of the IH models used by our industrial partner. To better illustrate the example, we only focus on the different inductors used by the IHs. Induction Hob 1 is the simplest IH; an inverter is connected to a power manager that connects with one standalone inductor (this construction is repeated two times in the IH). Induction Hob 2 is the next step in the evolution. An inverter is connected to a power manager that is connected to two inductors (one acts as the main inductor, and the other acts as a slave of the main; it is only activated if the main one is not able to heat the cookware placed on top by itself). Finally, Induction Hob 3 is composed by an inverter connected to a power manager that is connected to three inductors (they have different sizes and roles; one acts as main inductor, while the other two are auxiliary and are only activated when the size of the cookware is bigger than the main inductor). It is important to note that the three IHs share a common point (i.e., an inverter connected to a power manager that is connected to the main inductor). However, each IH provides different functionalities and is driven by different software elements.

The bottom left part of Figure 2 presents a representation of the results obtained after applying a reverse engineering model differencing approach. The inverter, power manager, and main inductor are identified as common parts to the three IHs and are therefore placed into the base model. Then a placement to hold the rest of the inductors (when they exist) is created. The first fragment holds the slave inductor that is present in the hotplate of the IH2. The second fragment holds the two auxiliary inductors that are present in the hotplate of the IH3.

The IH1 can be obtained without any substitution. The IH2 can be obtained by substituting the placement by the first fragment (IH2 = P1 → F1, P2 → F1). The IH3 can be obtained by substituting the placement by the second fragment (IH3 = P1 → F2). This division of the IH product models is valid, and the three input IHs can be derived from them. However, the results differ from the expected results; the groups of inductors have been divided, resulting in fragments that do not hold model units that are recognizable by our industrial partner's engineers.

The bottom right part of Figure 2 shows the expected result when dividing the IHs models into fragments. The base model is similar, but the placements are different, holding the power manager and the inductors to avoid the division of the groups of inductors. As previously, the three IHs can be derived from the model fragments (IH2 = P1' → F1', P2' → F1', and IH3 = P1' → F2'). Although the main inductor is the same for the three IHs, our industrial partner expects to have fragment models that hold whole conceptual patterns. Then, a new placement could be created inside the group of inductors to hold the main inductor. This is just a running example, but the problem grows bigger when taking into account real models (for instance, elements in charge of generating power are mixed with inductors in the same fragment). This results in a lack of recognition of the model

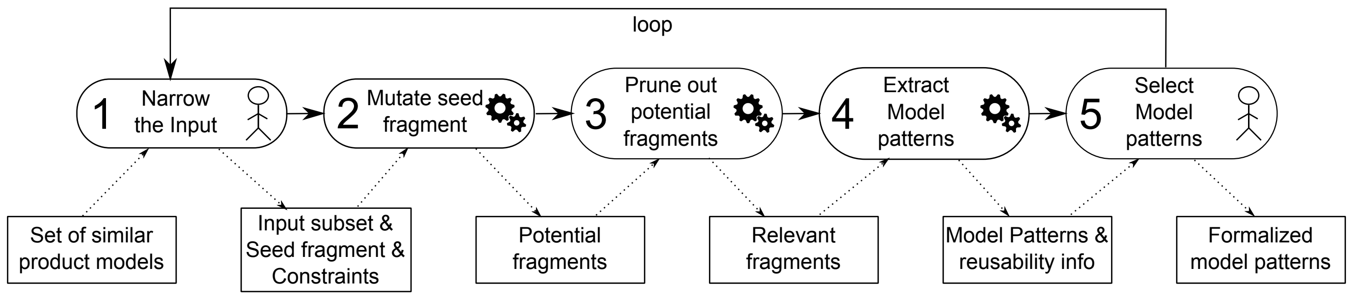

loop
1 Narrow the Input
2 Mutate seed fragment
3 Prune out potential fragments
4 Extract Model patterns
5 Select Model patterns
Set of similar product models
Input subset & Seed fragment & Constraints
Potential fragments
Relevant fragments
Model Patterns & reusability info
Formalized model patterns

Figure 3: The model pattern identification and extraction process

fragments produced that do not match the reusable units handled by our industrial partner.

To address this issue we propose a human-in-the-loop approach where the SPL engineers can take part in the variability identification and extraction process, contributing their knowledge and tailoring the process. As a result, the approach produces model patterns that hold the variability among the set of product models in terms of CVL placements and replacements. Those model patterns can then be applied to the product models to formalize the variability.

4. THE MODEL PATTERN IDENTIFICATION AND EXTRACTION PROCESS

Figure 3 presents an overview of the human-in-the-loop process to identify and extract model patterns, which consists of 5 steps. The initial input of the process is the set of similar models around which the SPL will be built. Some steps of the process are automatic (represented by two gears) while others are performed by the humans-in-the-loop (represented by a stickman). The domain experts and application engineers will contribute their knowledge to the process. Since the human roles involved in the establishment and further operation of a Product Line may vary depending on the particular approach adopted, we will refer to the people contributing knowledge to the process as the "humans".

The approach itself can be seen as a web search engine. After providing a search query (and some advanced search options) the search engine returns a set of webs that match the query, which are automatically ordered from the most accurate match to the last. The web search engine may return several millions of results, but only the most relevant will be browsed by the user (usually the top of the list). Then, the user can select any of the results provided or perform a new search trying to refine the results.

The first step enables the humans-in-the-loop to narrow the scope of the comparisons that will take place in further steps. By doing so, the task of identifying the model patterns is modularized, resulting in a manageable task. The humans also select the initial fragment that will be used as seed to identify the model pattern (i.e., a model fragment that the humans believe conforms a recognizable unit).

The second step performs mutations of the fragment designated as seed, taking into account the actual product model where the seed comes from. In other words, this step performs mutations (following the scope parameters provided in the first step), resulting in a set of potential fragments that are variations of the seed fragment. The selected seed will be used as a starting point, but a set of fragments built around the seed will be produced. The provided seed is not always accurate and by providing alternatives we facilitate the selection of the proper model pattern.

In the third step, the set of potential fragments is pruned out to discard the model fragments that do not fulfill the constraints stated by the humans in the first step. The objective of this step is to discard non-relevant fragments (according to the humans' constraints) as early in the process as possible, decreasing the cost of further steps. As a result, we produce a set of relevant fragments that has been built taking into account the humans' knowledge.

In the fourth step, a set of model patterns involving the relevant fragments is calculated. For each potential fragment, the corresponding placement signature in the original product model is calculated. Then, the process matches each signature with the product models selected in the first step; if the match is positive, a model fragment is extracted. Each model pattern consists of a placement signature and the set of model fragments that can be used with that particular placement signature. In addition, information about the occurrences for each model pattern is calculated (for the placement and for each of the matching fragments).

In the fifth step, the set of model patterns and the reusability information gathered is presented to the humans. The model patterns can be ordered following different criteria, such as the size of the placement or the number of alternatives for that placement. In addition, one of the model patterns is selected as the default choice, taking into account the number of times it has been reused in the models (following criteria similar to the fully automatic approaches [11, 15]). Finally, the humans select the model pattern that is relevant for them and that better formalizes their understanding of the domain, guided by the reusability information provided.

As a result of these five steps, a single model pattern is extracted and formalized. The process can be iterated to identify and extract new model patterns or to extend already extracted model patterns with the information present in new product models. The following subsections present a running example of the application of the approach to our industrial partner's domain.

4.1 Step 1 - Narrow the input

The presented process includes several comparisons among the product models which can be costly in terms of time and memory. The number of resulting model patterns presented to the humans in the fifth step depends on the products used as input and can be too high. Therefore, the approach provides the means to narrow the input, so that the number of resulting model patterns can be limited.

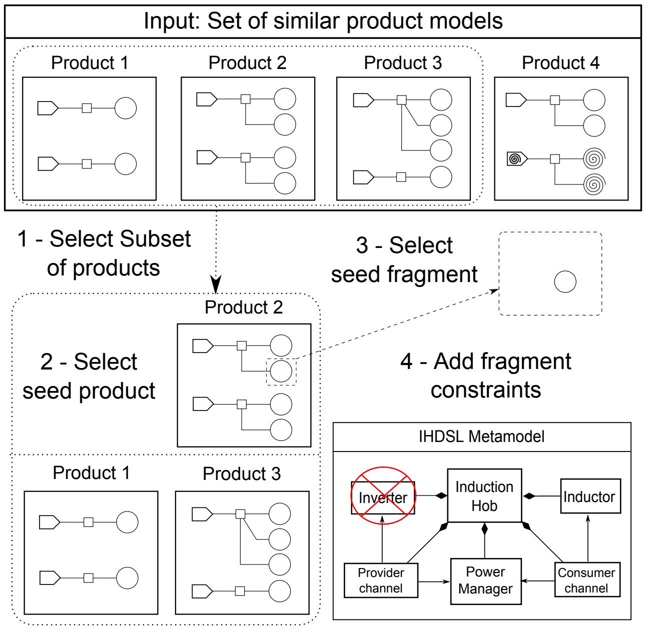


Figure 4: Step 1 - Narrow the input

As part of the first step, domain experts are in charge of four tasks: (1) selecting a subset of products from the input, (2) selecting one product as the seed product, (3) selecting one fragment from that product as seed fragment, and (4) narrowing the scope of the input to reduce the costs in further steps of the process.

The selection of the seed model and fragment is the most basic way of narrowing the input and the one that has the biggest impact on the results of the process. The resulting model patterns will include the selected fragment or slight mutations of it (i.e., a set of model patterns related to the selected fragment). Taking into account the knowledge of the domain (and the results of previous iterations) the humans select a fragment model from one of the product models used as input. The selection of the model fragment is done based on the intuition of what parts of the product model could be reused across several product models.

In addition, the knowledge of the humans about the domain and the product models enables them to have an accurate idea of the model fragments that they expect to obtain from this process. Taking into account this knowledge, the scope can be further refined in two different ways:

Input models: When a model is created following clone-and-own approaches [9], one of the existing models serves as the base for the new one. However, the model used as base in each case may vary. This practice could lead to a situation where there are different groups of models (which are not explicitly defined) that have a closer relation. If the humans are aware of the existence of these kinds of groups among the set of models, they can take advantage of it, scoping the iteration to just a subset of the input models.

Metamodel level: In order to narrow the set of potential fragments, the humans can indicate which meta-elements must be included in or excluded of from the resulting fragments. The process can be tailored so that each functional unit of the product is formalized as part of a different model pattern by defining constraints at the metamodel level. For instance, when applying the approach with large teams it is common to have experts of different parts of the domain, enabling each expert to work only with the subset of the model that the expert knows best, limiting the model patterns to just that subset.

Figure 4 shows a running example of the first step of the process, the scope of the input. A set of 4 products is provided as input (Product 1 - Product 4). First, the humans select a subset of the input (Product 1, Product 2 and Product 3); Product 4 is discarded since it is an odd product (it mixes induction with glass-ceramic radiant heaters). Then, the product model seed and the model fragment seed is selected from the subset of the input models. In this case, an inductor from Product 2 is selected; the humans are aware that inductors are present in all IHs and can be turned into a model pattern. Finally, a constraint is defined at the metamodel level and the resulting model patterns must comply with this constraint. In this case, the model patterns will not contain inverters to avoid the mix between power control and inductors in the same model pattern.

4.2 Step 2 - Mutate seed fragment

This step is performed automatically and takes as input the product model seed and the fragment seed selected in the previous step and the configuration of the mutations (if any). In this step, some potential fragments (which are closely related to the seed) are obtained. By doing this, the humans can evaluate whether the selected seed is a relevant fragment or there is another mutation that is more suitable.

The mutations are performed taking into account the fragment seed and the product model seed. Taking the seed fragment as starting point, some model elements are added to or removed from the seed fragment. However, the elements added during mutations are obtained from the seed product model, guaranteeing that the generated fragment is part of the seed model. In other words, apart from the selected seed fragment, the process proposes other variations of that fragment that are also part of the product model.

In order to obtain the mutations, the process performs additions, substractions, and combinations of both. In addition, the number and type of mutation operations can be configured or restricted to tailor the process towards the desired potential fragments. The number of chained mutations performed can also be adjusted to restrict the number of results.

Figure 5 shows the application of the step. A directed graph is built with all the potential fragments obtained by the mutations. Each node (circles) represents a potential fragment while each arc (arrows) represents one mutation (both directions, additive or substractive mutations). The nodes are classified by levels so that the process can be restricted to calculating fragments up to a fixed depth. The top part of Figure 5 shows the model fragment selected as seed (labeled as F1). It corresponds to Level 0 since no mutations are needed. Level 1 shows the fragments that can be obtained by one mutation; which in this case is F2, the result of adding the power manager to the original fragment. In this example, we are only considering the addition of elements that are connected in the seed model. We have restricted the generation of potential fragments up to Level 4, (i.e., fragments that can be obtained chaining up to four mutations). Note that some potential fragments (such as F5 or F8) are marked with a red crossed circle (this is part of

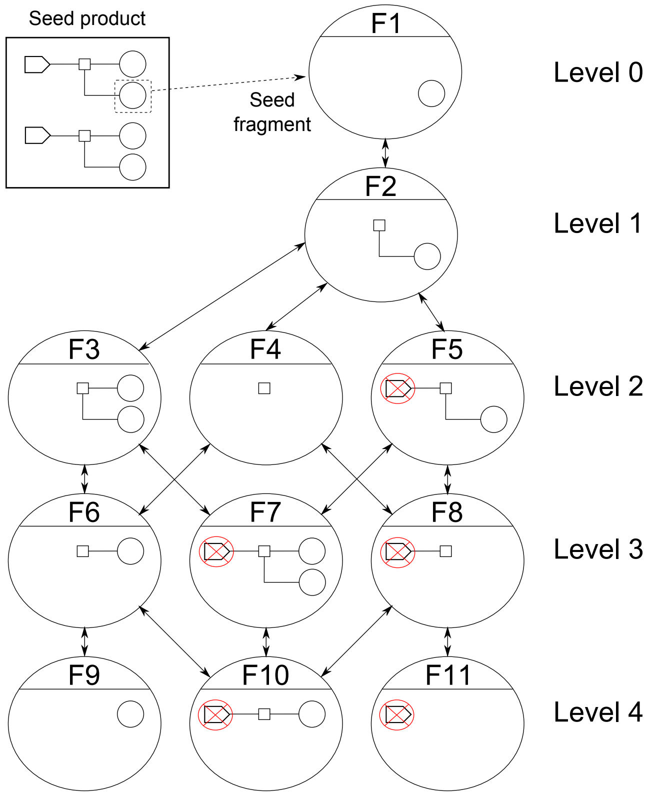


Figure 5: Step 2 - Mutate seed fragment

the prune out explained in the next step).

This step only produces fragments that are part of the original model. It does not create new elements; it just proposes variations of the seed fragment that are actual fragments of the original model. Note that all potential fragments produced (F1 to F11) are part of the seed product model. The elements added during mutations are obtained from the seed product model, so the resulting fragment is a subset of the seed product model.

As a result of the application of several chained mutations (addition or substractions), duplicated fragments may be produced. However, the process keeps track of the fragments that have already been produced to avoid duplication of work and results. Therefore, this step produces a set of potential fragments, including the seed fragment that was selected in the first step.

4.3 Step 3 - Prune out potential fragments

This step is performed automatically and prunes out the potential fragments calculated in the previous step, discarding the fragments that do not fulfill the constraints provided in the first step. In other words, for each of the potential fragments calculated, the process checks wether the constraints are satisfied or not; if not, the potential fragment is removed from the set. As a result, the set of potential fragments that will be used in further steps is reduced, avoiding the inclusion of undesired fragments in the result of the process.

In Figure 5, some of the potential fragments calculated contain inverters, but the humans decided to discard potential fragments containing inverter elements. Consequently, the potential fragments containing model instances of that metamodel element must be pruned out. In this example, since F5, F7, F8, F10, and F11 contain inverters they are discarded (see the bottom right corner of Figure 5).

By applying constraints to the resulting fragments, the humans can contribute their knowledge to the process. For instance, in this example, the humans are aware that there should be a reusable model fragment containing the inductor, but they are not totally sure about the exact elements that are part of the fragment. However, they know that the inverter belongs to another part of the system and want to prevent the inverter from being included in the model patterns that are calculated during this iteration. By means of this step, the potential fragments have been reduced from 11 to 6, simplifying further steps of the process. As a result, a single set containing all the potential fragments that meet the scope requirements provided by the humans is produced.

4.4 Step 4 - Extract model patterns

This step is performed automatically, and the input is the set of potential fragments that are relevant for the humans (from the previous step) and the subset of products provided by the humans as part of step 1. As output, this step produces a set of model patterns that is annotated with information about its reusability among the subset of products. A model pattern can be seen as a variation point (and the alternatives for that variation point) that can be reused several times.

First, the placement signature is calculated for each of the potential fragments; that is, we calculate the boundary information between the potential fragment and the rest of the seed model. Particularly, we calculate the set of incoming and outgoing relationships regarding the potential fragment and the seed model. This placement signature can be used to identify spots in the models where that potential fragment can be used.

Then, we match each of the placement signatures against each product model from the input subset. By doing so, we determine wether a particular placement can be found in each product model. If the match is positive, the fragment of that particular product that matches with the placement signature is extracted. Thus, each of the placements' signatures can be seen as a variation point (that is not bounded to a particular product) and each of the fragments extracted can be seen as an alternative for that variation point.

Finally, for each model pattern (placement signature and alternative fragments), the process computes the number of times it can be applied in the subset of products (i.e., the number of times that the match between the placement signature and the product is positive and the number of times that each of the fragments is used in the products).

Figure 6 shows the application of this step to the running example. Each row shows information for one of the relevant potential fragments obtained. The first column shows the potential fragment, and the second column shows the placement signature calculated for each potential fragment. The rest of the columns show the matching against each of the products from the subset provided in step 1.

For instance, the first row of Figure 6 shows the seed fragment selected by the humans (F1, the first column), the placement signature corresponding to that fragment (the second column) and the matching of the placement signature with the product models. In this case, another potential

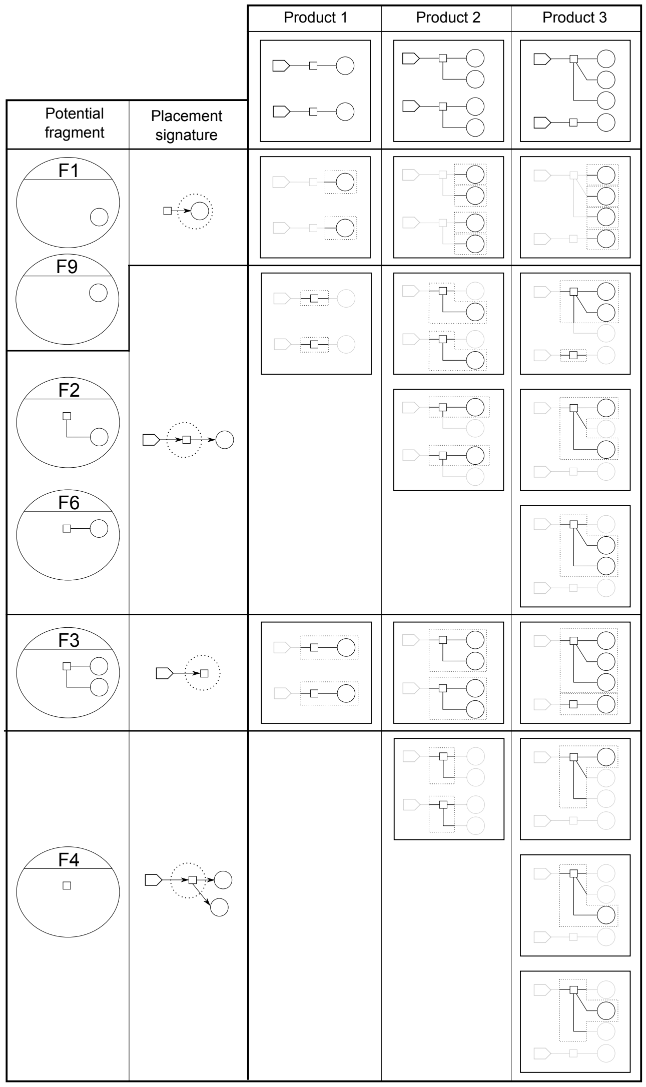


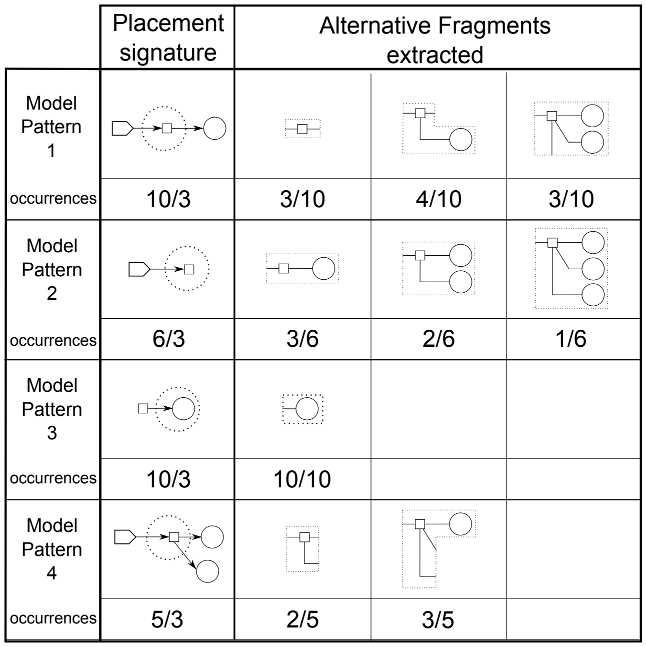
Figure 6: Step 4 - Extract model patterns and occurrences

placement (F9, the first column) produces the same placement signature as F1. Not all placement signatures match all products; for instance, the placement signature corresponding to F4 (the last row), does not match Product 1.

As a result of this step, a model pattern is generated for each of the potential fragments used as input (i.e., the calculated placement signature, the alternative fragments matching that placement extracted from the product models, and the information about occurrences for each of the fragments and the placement itself).

4.5 Step 5 - Select model patterns

In this step, the model patterns obtained are presented to the humans so they can choose the ones that are most relevant for them based on their knowledge and the information provided by the approach. Each model pattern is presented with the information about the occurrences in the product models gathered in the previous step (i.e., the number of positive matches of the placement signature among the products and the number of occurrences for each of the alternative fragments extracted).



	Placement signature	Alternative Fragments extracted		
Model Pattern 1				
occurrences	10/3	3/10	4/10	3/10
Model Pattern 2				
occurrences	6/3	3/6	2/6	1/6
Model Pattern 3				
occurrences	10/3	10/10		
Model Pattern 4				
occurrences	5/3	2/5	3/5	

Figure 7: Step 5 - Select Model Patterns

Figure 7 shows the application of this step to the running example. Each row shows information about a model pattern. The first column presents the placement signature of the model pattern and the rest of the columns show each of the extracted alternative fragments that match that particular placement signature. The number below each placement signature represents the number of possitive matches out of the total number of products used as input. The number below each alternative fragment represents the number of occurrences of that particular fragment out of the total number of matches of the placement signature.

For instance, the first row shows the model pattern selected by default. The pattern has been identified ten times in the three products analyzed and is the one that has the most fragment alternatives with three possibilities. Therefore, it has been selected as the default pattern by the process. However, even though this model pattern is the one that is most frequently repeated, the humans do not recognize all the alternative fragments presented because they are incomplete (they do not contain all the inductors), so it is discarded. The second row shows a model fragment that holds inductors; it has been identified six times and also has three different alternatives. In fact, each alternative contains the whole group of inductors connected to a power manager, as the humans were expecting. The third row shows the model pattern containing the selected seed fragment; However, even though it is matched ten times in the models provided, it does not present alternatives and would be considered as a common part of the model by traditional reverse engineering model differencing approaches.

Taking into account the reusability information, the humans select the model pattern that best fits their understanding of the domain. For instance, they can stick to the initial seed selection (model pattern 3), but some of the model patterns provided may be more accurate and relevant for them (like model pattern 2). The model pattern presentation (along with the reusability information) enables the humans to reason about the fragments, thus facilitating the task of determining which one should be selected.

4.6 Loop

The five-step process enables the extraction of one model pattern and can be repeated until all the recognizable units that conform to the models have been extracted. In addition, already extracted model patterns may be extended to include new replacements that were not present in the original iteration.

For instance, a new iteration could be run (taking into account the model patterns already selected) to extract another model pattern holding the inverters. Similarly, an iteration to refine an existing model pattern could be run, including new models that were not part of the original iteration (discarded during the step 1) and resulting in a refinement of the former pattern.

The number of iterations needed depends on the domain where it is being applied and the amount of variability that must be formalized. In addition, some of the iterations will not produce a relevant model pattern and will need to be refined until the desired model pattern is obtained.

5. CASE STUDY: INDUCTION HOBS DOMAIN

This section presents our experience building a Product Line from an existing set of products from our industrial partner (BSH group). This company is the largest manufacturer of home appliances in Europe and one of the leading companies in the sector worldwide. Their induction division has been producing induction hobs (the brand portfolio is composed by Bosch and Siemens among others) over the last 15 years.

In order to implement the approach, several technologies are involved. Specifically, CVL can be applied to MOF based models, so the approach is developed within the Eclipse environment using the Ecore implementation and the Eclipse Modeling Framework (EMF) [1]. The mutations of the seed fragment (Step 2, Section 4.2) are implemented based on ecore-mutator [2], which is an EMF-based framework to mutate EMF models that conform to an Ecore metamodel. The comparisons among models (Step 4, Section 4.4) are implemented based on EMF-Compare [3],which is an Eclipse framework to compare instances of EMF models.

As a first attempt, we applied a reverse engineering model differencing approach [11, 15] and followed up with an evaluation of the obtained fragments that revealed the problem presented in Section 3. Then, we applied the approach presented in this paper, a human-in-the-loop approach to identify and extract conceptualized model patterns as reusable variation points. Finally, the patterns were used to formalize the variability among the set of existing products and incorporated to the Product Line supporting tool to enable the reuse of the patterns when creating new products.

The model patterns extracted following the approach include the model information necessary to create CVL variation points. Specifically, a pattern contains the placement signature of a fragment and a set of matching fragment alternatives to replace it. Therefore, the patterns can be used to formalize the variability among a set of products.

We have used the model patterns identified and extracted to improve our industrial's partner tool, enabling the formalization of variability based on those patterns. For each model pattern, we have created (1) a custom editor that enables the creation of new replacements for a particular model pattern, (2) a fragment library that holds all the replacements identified throughout the process for that particular model pattern. The resulting tool can be seen here [4].

[1]http://www.eclipse.org/modeling/emf/

[2]https://code.google.com/a/eclipselabs.org/p/ecore-mutator/

[3]https://www.eclipse.org/emf/compare/index.html

5.1 Application of our approach

The initial input of the approach is a set of 46 induction hob models, corresponding to products that are currently being sold or that will be launched to the market in the immediate future. The set of models were developed following a clone and own [9] approach, where each IH has been modeled modifying a copy of the most similar IH present in the collection. Therefore, the variability present among the models has not been explicitly defined, resulting in a set of models with implicit variability among its members.

However, there is implicit knowledge among our industrial partner's engineers of the traceability of the clones performed. In other words, the engineers are aware of the existence of groups of models that may have more similarity among them since they share a common ancestor product used as base. Therefore, we used this information to perform the division of the input set into smaller subsets.

With regard to the products complexity, each of the IH models is composed of more than 500 elements, including around 100 class elements on average. For each IH, there are around 10^{29} different potential fragments composed of more than one class element. The magnitude gives the idea that manually processing this amount of data may be cumbersome, error prone, and could not be done in a reasonable time.

For the application of the approach, we had the collaboration of the engineers from the firmware department; they are experts in developing firmware for induction hobs and are in charge of maintaining and evolving the firmware used in the IH. Those engineers are the owners of the induction hob models and provided their knowledge to tailor the approach.

We performed several iterations looking for model patterns, but not all of them ended up in the extraction of a model pattern. Some of them were used as the base for further refinement. For instance, we performed several iterations looking for a model pattern that held groups of inductors (presented as running example in Section 4). We started with a subset containing the most basic IHs and with a very conservative set of constraints. With the information of that iteration, we refined the constraints until we obtained a pattern that satisfied our industrial partner's engineers.

Then, we performed new iterations, adding more induction hobs to the subset, in order to find new alternatives for the model pattern previously extracted. We added induction hobs that held each of the different group of inductors present in the products. Finally, we performed several extra iterations in order to confirm that the extracted model pattern was correct and contained all the alternatives desired.

[4]www.carloscetina.com/variabilitytool.htm

When looking for the inverter groups, we followed a similar strategy, performing several iterations and refining the search. However, the subsets employed to search for this model pattern were totally different (as suggested by our industrial partner engineers). Even more, the domain experts that took part in the process were also different (since they had more knowledge of this specific part of the system).

As part of the application of the approach, we conducted a usability evaluation, including satisfaction tests, interviews, and focus groups, where the engineers could talk freely about the process and the fragments that were being identified. Although the usability evaluation is out of the scope of this work, we outline some of the findings below.

Specifically, we wanted to know more about the rationale behind the selection of one model pattern over the others. The approach proposes a model pattern based on the number of times it is reused across the products (as other automatic approaches do), but the proposed pattern was not always the chosen one. We identified three main reasons for not choosing the proposed model pattern:

Odd elements in the fragments: It is not clear beforehand how resulting fragments are going to be, but there are some elements that are not expected to be part of the model pattern. However, odd elements can be incorporated by automatic approaches due to the number of occurrences (as in the motivation of the approach example). In the presented approach, the domain experts can state that odd elements pertain to different parts of the system and choose to have them in separate fragments. When browsing the results of the approach, the domain experts can compare among patterns with different levels of granularity, easing the task of determining which elements should be part of the fragment and which ones should not.

Deprecated elements: Another reason for selecting a model pattern that is different from the one proposed is the inclusion of deprecated elements as part of the model pattern. Thus, the default pattern proposed has a high number of alternatives for the placement, but the humans know that those elements are going to disappear in further versions. Therefore, they prefer to not give them too much relevance as they will not be part of future products anymore.

Future developments: Sometimes, there is a model pattern that is treated as irrelevant by the approach due to its low number of occurrences or alternatives. However, this pattern was relevant for the humans because they knew that in future developments that element would be split into several elements. Therefore, they wanted to include it among the model patterns selected. For instance, in Figure 7, the Model Pattern 3 presents no alternatives, but the engineers knew that different kind of inductors would be developed in the future, so they wanted to include that model pattern.

6. RELATED WORK

Some works focus on transforming legacy products into Product Line assets. For instance, in [5], the authors present their experience in the Digital Audio & Video Domain. In [6], the authors explain their experience reengineering the Image Memory Handler from Ricoh's products into a SPL. In [8], the authors report on their experience applying an extractive approach to a set-top box manufacturing company. These approaches extract variability from legacy products in industrial environments, but they focus on capturing guidelines and techniques for manual transformations. In contrast, our goal is to introduce automation into the process while taking advantage of the knowledge of the domain experts through a human-in-the-loop extractive approach.

Other works focus on the automation of the extraction process, obtaining the variability from legacy products by comparing the products with each other. In [14] the authors present an approach to mine family models from block-based models. The similarity between models is measured following an exchangeable metric, taking into account different attributes of the models and can be fine-tuned depending on the application. Then, the approach is further refined [4] to reduce the number of comparisons needed to mine the family model.In [11], the authors propose a generic framework for mining legacy product lines and automating their refactoring to contemporary feature-oriented SPLE approaches. They compare the elements of the input with each other, matching those whose similarity is above a certain threshold and merging them together. In [15], the authors propose a generic approach to automatically compare products and extract the variability among them in terms of a CVL variability model. These two approaches automatically turn identical elements into common parts of the product line, similar elements into alternative parts, and unmatched elements into optional parts. However, fully automating the decision of what should be reused and how it should be done reduces the flexibility of the approaches. In contrast, our work enables the domain experts to decide which elements should be formalized as part of the SPL based on the results of the comparisons, instead of performing them as an automatic output of the comparisons.

Finally, some research efforts focus on the source code of the products in order to extract the variability model. For instance, the authors in [12] present a tool-supported approach for reverse engineering feature models from different sources, such as makefiles, preprocessor declarations, and documentation. They focus on the crucial point of identifying parents and combine logic formulas and descriptions as complementary sources of information. In addition, the authors in [16] propose an approach to identify features from the source code of products. They reduce the noise induced by spurious differences of various implementations of the same feature. Then, the process produces feature candidates that are manually pruned (to remove non-relevant candidates). The approach is further extended in [17] to introduce ExtractorPL, an automated technique that infers a full implementation of a SPL from the given code. However, these approaches focus on the source code level of the products, while our approach is applied at the model level which enables domain experts to contribute their knowledge to the identification and extraction of reusable model fragments.

7. CONCLUSION AND FUTURE WORK

As part of this work, we have identified the need for an approach to identify and extract conceptualized model patterns from a set of similar product models that match the expectations of the domain experts. We have proposed a human-in-the-loop approach that enables domain experts to contribute their knowledge to the identification and extraction process. In addition, we have implemented the presented approach within the Eclipse environment. We have validated the approach with our industrial partner, extracting the model patterns that are present in a set of real induction hob models. We have outlined the rationale followed by our industrial partner to prefer one model pattern over the rest (even though it is reused fewer times among the models). Finally, we have applied the resulting model patterns to a modeling tool, enabling the evolution (creating new fragment alternatives) and reutilization of the model pattern.

The identification of model patterns through a human-in-the-loop approach has provided model fragments that are recognizable by the SPL engineers. The humans involved in the extraction process could contribute their knowledge to the process, tailoring it to produce the desired model fragments and thereby formalizing the variability among a set of products.

There are some open questions remaining about the presented approach, such as how many model patterns should be extracted, what model pattern granularity works best, or if nested model patterns could benefit the process. To address those open questions, we are currently applying our approach in CAF [5], an international company that builds and deploys railway solutions around the world. Similarly to our industrial partner, they need a solution to formalize the variability that exists among their products.

The use of a human-in-the-loop approach seems to be successful in identifying and extracting model patterns, as it enables the domain experts to immerse themselves in the process and contribute their knowledge. As a result, the fragments represent recognizable units that are extracted and fulfill the expectations of the engineers involved. As one of our industrial partner engineers reported: "The first fragment library (reverse engineering model differencing) was generic, but the new fragment library (our model patterns approach) was made just for us".

8. REFERENCES

[1] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire. Extraction and evolution of architectural variability models in plugin-based systems. *Software & Systems Modeling*, pages 1–28, 2013.

[2] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns.* Addison-Wesley Professional, 3rd edition, Aug. 2001.

[3] 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.

[4] 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: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, SPLC '14, pages 36–43, New York, NY, USA, 2014. ACM.

[5] 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.

[6] 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.

[7] C. Krueger. Easing the transition to software mass customization. In F. van der Linden, editor, *Software Product-Family Engineering*, volume 2290 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin Heidelberg, 2002.

[8] 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.

[9] N. Pham, H. Nguyen, T. Nguyen, J. Al-Kofahi, and T. Nguyen. Complete and accurate clone detection in graph-based models. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 276–286, May 2009.

[10] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques.* Springer, 2005.

[11] J. Rubin and M. Chechik. Combining related products into product lines. In J. de Lara and A. Zisman, editors, *Fundamental Approaches to Software Engineering*, volume 7212 of *Lecture Notes in Computer Science*, pages 285–300. Springer Berlin Heidelberg, 2012.

[12] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. Czarnecki. Reverse engineering feature models. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 461–470, New York, NY, USA, 2011. ACM.

[13] 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'10, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] 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*, SPLC '13 Workshops, pages 44–51, New York, NY, USA, 2013. ACM.

[15] X. Zhang, Ø. Haugen, and B. Møller-Pedersen. Model comparison to synthesize a model-driven software product line. In *Software Product Line Conference, 2011 15th International*, pages 90–99, Aug 2011.

[16] T. Ziadi, L. Frias, M. da Silva, and M. Ziane. Feature identification from the source code of product variants. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 417–422, March 2012.

[17] T. Ziadi, C. Henard, M. Papadakis, M. Ziane, and Y. Le Traon. Towards a language-independent approach for reverse-engineering of software product lines. In *Symposium on Applied Computing*, SAC '14, 2014.

[5]http://www.caf.es/en