

Automating the Variability Formalization of a Model Family By Means of Common Variability Language

Jaime Font^{1,2}

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

Manuel Ballarín¹

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

Øystein Haugen^{2,3}

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

Carlos Cetina¹

ABSTRACT

The aim of domain engineering process is to define and realise the commonality and variability of a Software Product Line. In the context of a family of models, spotting the commonalities and differences may become cumbersome and error prone as the number of models and its complexity increases. This work presents an approach to automate the formalization of variability in a given family of models. As output, the variability is made explicit in terms of Common Variability Language. The model commonalities and differences are specified as placements over a base model and replacements in a model library. The resulting Software Product Line (SPL) enables the derivation of new product models by reusing the extracted model fragments. Furthermore, the SPL can be evolved by the creation of new models, which are in turn automatically decomposed as model fragments of the SPL. The approach has been validated with our industrial partner (BSH), an induction hobs company. Finally, we present five different evolution scenarios encountered during the validation.

CCS Concepts

•Software and its engineering → Software product lines;

Keywords

Reverse Engineering, Model-based Software Product Lines, Variability Identification, Common Variability Language

1. INTRODUCTION

A Software Product Line (SPL) enables a planned reuse of software components into products within the same scope. The software product line 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) [9].

The proactive strategy for the adoption of an SPL is traditionally regarded as the typical approach. Following this strategy, the assets of the SPL are developed prior to the derivation of any product [6]. However, a recent survey reveals that only a minority of industrial SPLs are planned proactively, being the extractive approach more used (where existing products are re-engineered into an SPL) [3].

In particular, in model-based SPLs, the members of the SPL are specified in the form of models. However, in the context of a family of models, manually spotting the commonalities and variability among the models may become cumbersome and error prone, particularly as the number of models and its complexity increases.

There are several research efforts towards automating the formalization of the variability existing among products [1, 2, 11, 14]. However, those works are mainly based on Feature Models extraction and do not properly support variability formalization by means of the Common Variability Language (CVL). In addition, existing works [10, 12] are not designed with the evolution of the SPL on mind. The evolution of SPLs should be considered as the normal case, not as an anomaly [4].

This work presents Model Family to SPL, an approach to automate the variability formalization of a given family of models into an SPL. As output, the variability is made explicit in terms of CVL [5]. The model commonalities are formalized as a base model and variabilities are specified as placements over the base model and replacements in a model library. In addition, the resulting SPL can be further evolved to include new products. In particular, our approach enables the automatic decomposition of new product models into model fragments that are incorporated to the model library.

We have validated the approach with our industrial partner (BSH), the largest manufacturer of home appliances in Europe. Their induction division has been producing induction hobs (under the brands of Bosch and Siemens among others) over the last 15 years. We have applied the presented approach to a set of their induction hobs models to build an SPL to generate the firmware for their products. In addition, we present the five evolution scenarios faced by our industrial partner when evolving the SPL to incorporate new product models.

The rest of the paper is structured as follows: next section introduces some background about the CVL and our indus-

trial partner’s domain. Section 3 presents our approach for extracting variability from a set of product models. In section 4 we present our experience applying the approach to our industrial partner’s domain, focusing on the evolution scenarios encountered. Section 5 discusses related work. Finally we conclude the paper.

2. BACKGROUND

This section presents the main concepts of the Domain Specific Language (DSL) used to specify Induction Hobs (hereinafter referred as IHs) and the CVL. Both, the Induction Hob Domain Specific Language (IHDSL) and CVL are the techniques which we use to describe the model-based SPL of our industrial partner.

2.1 Induction Hob Domain Specific Language (IHDSL)

The IHDSL metamodel used by our industrial partner is composed of 46 metaclasses, 74 references among them and more than 180 metaclass properties. However, in order to gain legibility and due to intellectual property rights concerns, in this paper we use a meaningful simplification of it (see top-left corner of Figure 1).

Induction Hobs use electromagnetic induction phenomenon to cause the generation of heat on the cookware that is then transferred to the food. Induction hobs are composed of several elements, being the most important the inverter (where the energy is modulated) and the inductor (where the electromagnetic field is generated).

Top-right corner of Figure 1 shows the graphical representation of the IHDSL. The big rectangle represents the IH itself. It is composed of two power modules (vertical rectangles at both sides of the IH) and each of them holds two inverters (squares). Inverters are connected to the inductors (circles). The number inside each inductor represents the diameter of the inductor. The line that connects inverters and inductors represent the channel, which transfers energy from the inverter to the inductor. The user interface of an IH has buttons to configure the power level of each inductor. In top-right corner of Figure 1, the horizontal rectangle at the bottom of the IH represents the user interface. It has ports to connect each inductor with his button.

In order to gain legibility through the rest of the paper we will focus on the variability regarding the inductor. However, there are several parts of the induction hobs that are subject to variation such as the inverters and how are connected with the inductors. Our work with our industrial partner has covered the variability of the whole induction hob although only a subset is presented.

2.2 Common Variability Language (CVL)

CVL is a DSL for modeling variability in any model of any DSL based on Meta-Object Facility (MOF), an OMG’s specification to define a universal metamodel for describing modeling languages. CVL defines variants of the base model by replacing parts of the base model by Model replacements found in a library. Figure 1 presents an overview of CVL.

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 subject to variations are the **placement fragments** (hereinafter placements). A placement can be any element or set of elements that is subject to variation.

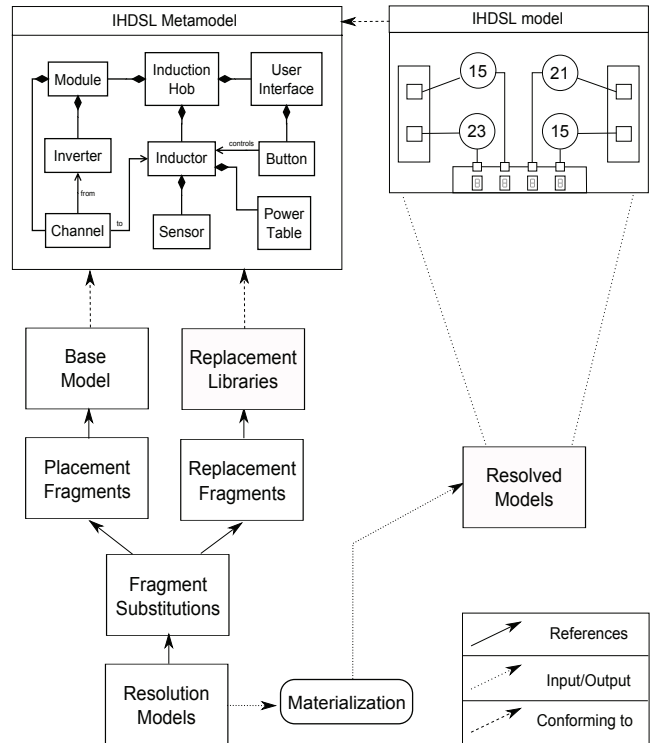


Figure 1: CVL Overview

To define alternatives for a placement we use a **replacement library**, a model 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** (hereinafter substitution). Each substitution references to a placement and a replacement and includes the information necessary to substitute the placement by the replacement. That is, 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 executed, the base model (with a placement substituted by a replacement) continues to conform to the same metamodel.

Each **resolution model** represents one variant of the base model. The resolution model references a set of substitutions that needs to be executed in order to create the variant. When a resolution model is materialized, produces a resolved model, which is a variant of the base model where the substitutions defined by the resolution model have been executed. For further details about the inner workings of CVL see [5].

3. MODEL FAMILY TO SPL

This section presents Model Family to SPL, our software process capable of turning the implicit variability existing

among a given set of similar models into explicit variability. In particular, Model Family to SPL takes a product family modeled in any DSL (conforming to MOF) as input and generates a CVL based SPL where commonalities and variabilities among the model family are explicitly defined. That is, each of the models of the given model family are expressed in terms of CVL, resulting in an SPL capable of generating all the products from the given model family.

Figure 2 shows an example of execution of *Model Family to SPL*. Top part shows the input of the process, the model family. Bottom part shows the output of the process, an SPL formalized by CVL models. Middle part shows how the execution of the processes is performed. Product Family to SPL is composed of two sub-processes, **Select Base Model** and **Product Model to SPL**. *Select Base Model* analyses the given family of models and determines which one of them is more suitable to be the base model. Once the base model is selected, *Product Model to SPL* compares a product model from the model family with the base model and updates CVL models to include the product into the variability definition. *Product Model to SPL* formalizes the variability of each given product model and incorporates it into the SPL

3.1 Select Base Model

The selection of the base model phase designates the base model that is used through the rest of the process. In this example we use the number of differences between the base model and the rest of the model family to determine the base model. Using the number of differences among the models produces simpler CVL models in terms of the number of substitutions needed to formalize each model. However, other values can be used to select the base model, the rest of the process can be executed no matter which base model is selected.

The first process executed as part of the Model Family to SPL is *Select Base Model*. Figure 2 shows an example of the execution of the process (top part). In addition, Figure 3 shows the state machine associated to the *Select Base Model* process (top part). Given a Product Model Family, the process *Select Base Model* takes the model family as input and proceeds as follows:

- 1.1 Compare.** All the models from the model family (IH1, IH2 and IH3) are input into the compare operation. The models are paired two by two in all the possible combinations (the order doesn't matter) and then each pair is compared. The comparison is performed at element level, matching one element from first model with another from the second model. The process continues comparing pairs until there are no more pairs. The result is a set of differences between each pair of models processed. Figure 2 shows the result of the operation. For instance, the comparison between IH1 and IH2 produces a set of 4 diffs, because the four inductors of IH1 are different from the inductors of IH2.

- 1.2 Aggregate.** The number of differences among each model and the rest of the models from the model family is added together. This is done for each of the models of the model family (IH1, IH2 and IH3). This aggregate value indicates the total number of differences among a given model and the rest of the product models. That is, the value indicates the number

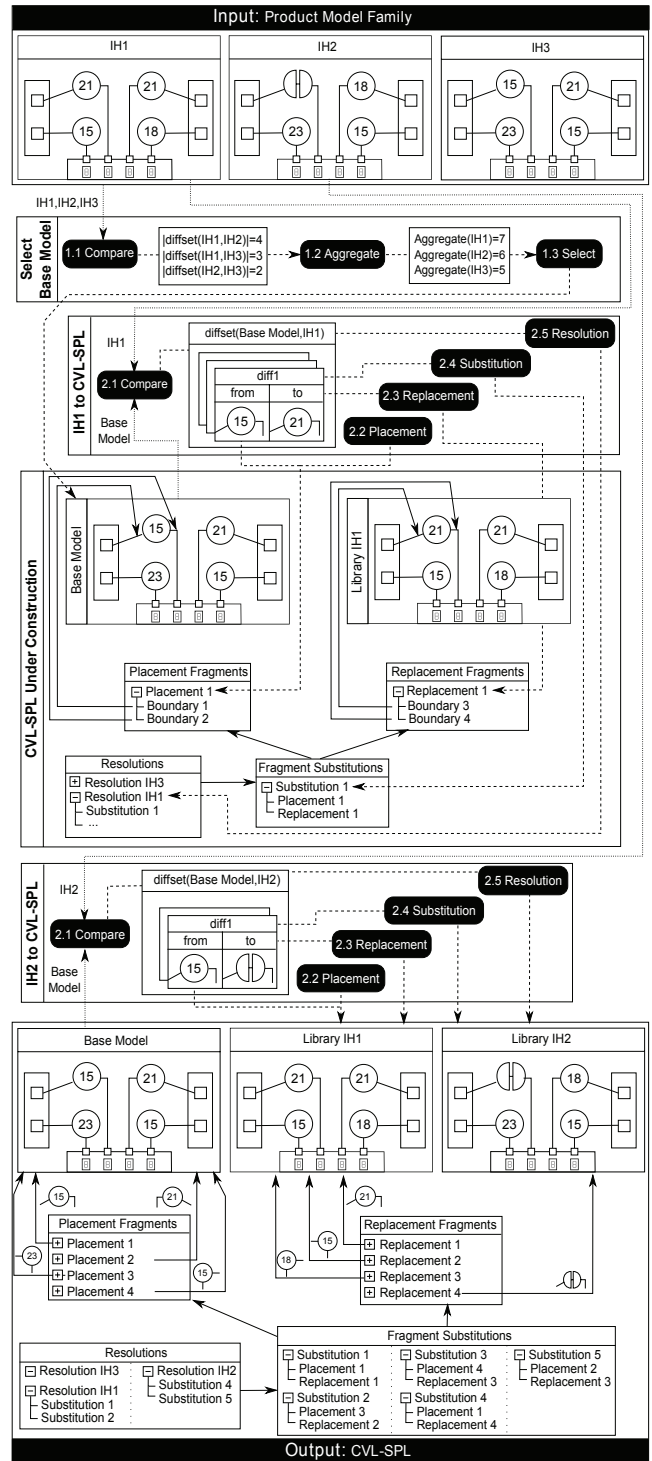


Figure 2: Model Family to SPL execution

of differences that would need to be addressed if that particular model were the base model. For instance, if IH1 were the base model a total number of 7 differences would need to be addressed (4 differences with IH2 and 3 differences with IH3).

- 1.3 Select.** When the aggregated values have been cal-

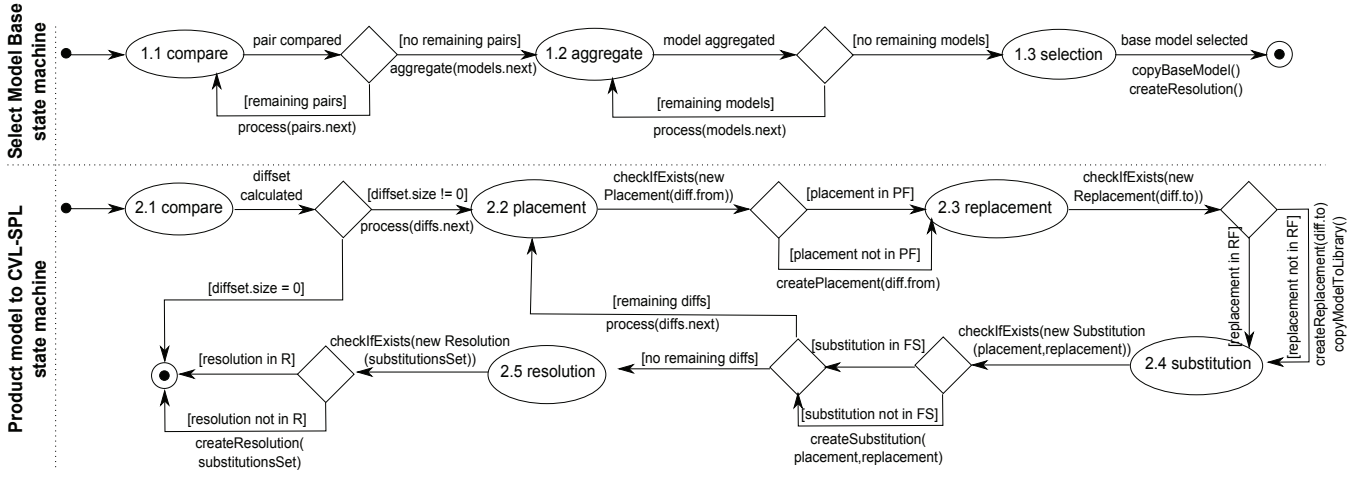


Figure 3: Select Base Model process and Product Model to SPL process state machines

culated for all the models, the model with the lowest value is designated as the base model. Therefore, it is included into the SPL, as it will be the base for all the products generated with the SPL. The model designated as base model (IH3 in this case) must be derivable from the SPL, therefore, a resolution model capable of generating the base model is created (Resolution IH3). As IH3 is the base model itself, there is no need of substitutions and Resolution IH3 is empty.

3.2 Product Model to SPL

After executing the first process (the base model has been designated), the second phase of the process starts, the population of the SPL. The population consists of executing the process *Product Model to SPL* for each of the product models of the input (except for the base model, that has been already included into the SPL).

The *Product Model to SPL* process, performs compare operations between each of the models from the given model family and the base model, using the same compare operation as in the previous process. However, this time we shall create and update the CVL models that define each of the differences among the model family as sets of placements, replacements, substitutions and resolutions.

Figure 2 shows an example of the execution of the *Product Model to SPL* process for IH1 (middle part, below *Select Base Model* process) and a snapshot of the SPL that is being constructed. In addition, Figure 3 shows the state machine for *Product Model to SPL* process (bottom part). Given a product model and a Base Model (designated by previous process), *Product Model to SPL* proceeds as follows:

2.1 Compare. The first model from the input model family (IH1) is compared with the base model. The result is a list of differences between the two models, $diffset(Base\ Model, IH1)$. Each difference has two elements, the *from* element references elements from the base model and the *to* element references elements from the other compared model (IH1 in this case). They reference the elements spotted as different by the compare operation. For instance, $diff1.from$ element references the inductor of size 15 of the base model while $diff1.to$ element references the inductor of

size 21 of the IH1 model. It is important to notice that the difference not only holds the element that is different (inductor), but also the references involving that element (in this case references from the button and from the inverter).

2.2 Placement. The process checks if a placement holding exactly the same elements of $diff1.from$ exists in the Placement Fragments model. As it does not exist, the process defines a placement over the base model (Placement 1). The references involving the differing element (the inductor) are defined as the boundaries of the placement (Boundary 1 and Boundary 2). If the placement is already defined in the Placement Fragments model it is not created again (see bottom of Figure 3).

2.3 Replacement. Once the placement is retrieved (created a new one or retrieving the existing one from the Placement Fragments model), the process continues with the replacement. Similarly as in previous step, the process checks if a replacement holding the information from $diff1.to$ exists in the Replacement Fragments model. It does not exist, therefore it needs to be created, but this time will be defined over a model of the Replacements Library. To accomplish that, the model being processed (IH1) is copied into the fragments library and then a replacement is defined over it (Replacement 1). As with placement fragments, the references involving the differing element (the inductor) are defined as the boundaries of the replacement (Boundary 3 and Boundary 4). If the replacement is already defined in the Replacement Fragments model, there is no need to create a new one as indicated by the state machine (see bottom right part of Figure 3).

2.4 Substitution. Once the placement (Placement 1 from step 2.2) and the replacement (Replacement 1 from step 2.3) had been retrieved (creating them if necessary), the process is ready to create the substitution of the placement by the replacement. Similarly to previous steps, the process first checks if the substitution already exists in the Fragment Substitutions model. As the substitution does not exist, the process needs

Family of product models

Product models editor

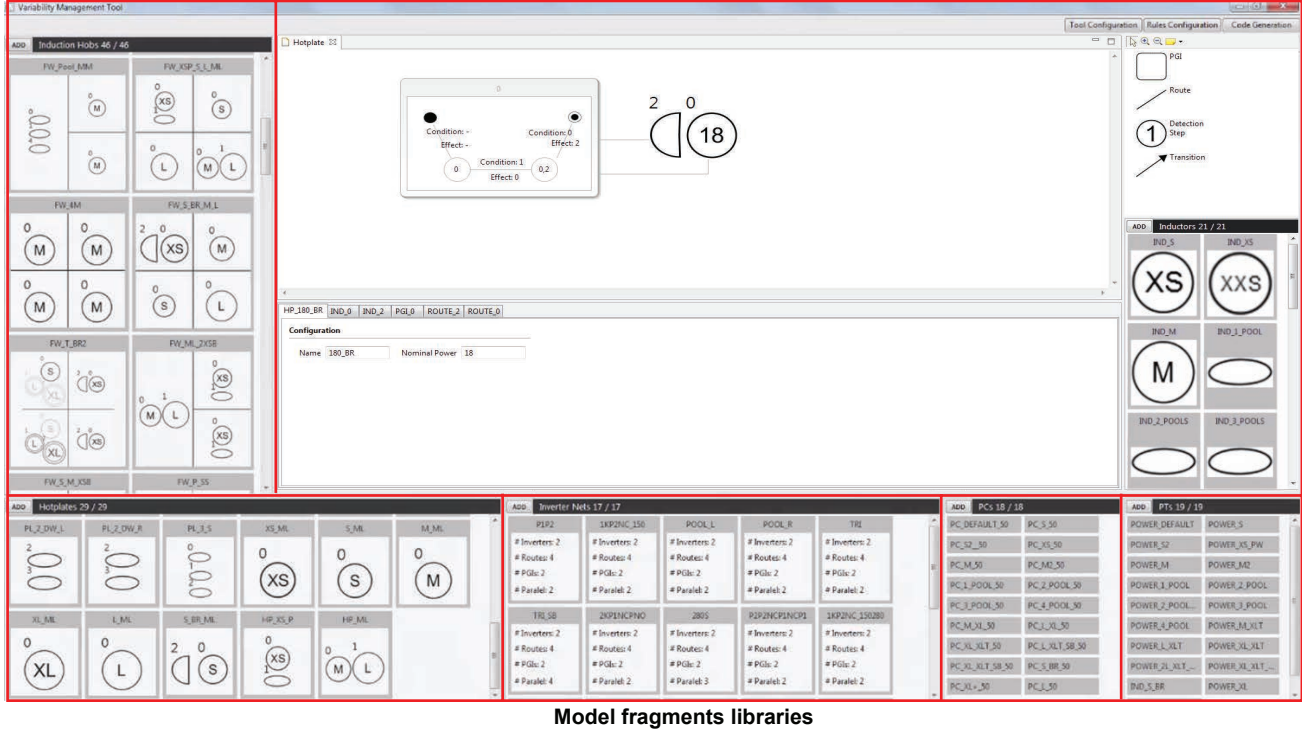


Figure 4: Resulting SPL for Induction Hobs domain

to create it. The substitution indicates that the Placement 1 can be substituted by the Replacement 1. As part of the definition of the substitution, links between the boundaries from the placement and the replacement are established. Therefore, when the fragment substitution is executed the elements can be updated properly and the model continues to conform to the metamodel. Similarly to previous step, if the substitution already exists there is no need to create it (see bottom of Figure 3).

At this point, the first difference ($diff1$) from the $diffset(Base Model, IH1)$ has been processed. Now, the steps 2.2, 2.3 and 2.4 are performed for the rest of differences of the $diffset$. For each difference, a placement, a replacement and the proper substitution of the placement by the replacement are obtained (created or retrieved if already exists). The iterations for $diff2$ and $diff3$ are not shown in Figure 2.

2.5 Resolution. When all the differences from the $diffset$ had been processed, the process is ready to create a resolution for the processed model (IH1). First, the process checks if the resolution already exists in the resolutions model. As the resolution does not exist, the process creates a new one (Resolution IH1). In this case, the process indicates that the resolution of IH1, involves the Substitution 1 (substitution of Placement 1 by Replacement 1) corresponding to the first $diff$ processed. Similarly, substitutions for the rest of the differences of the $diffset$ are included in this resolution.

This five-step process is repeated for all the models from the input (except for the base model). After executing $IH1$

to SPL , comes the execution of $IH2$ to SPL . The result is an SPL populated with all the models from the input family. Bottom part of Figure 2 shows the output of the *Model Family to SPL* operation. There is a base model and two library models conforming to the IHDSL. In addition, there are placements defined over the base model, and replacements defined over the library models. Moreover, substitutions are defined referencing placements and replacements; resolutions that generate each of the models received as input have been created based on those substitutions.

With the above process we obtain a CVL-based SPL capable of generating exactly the same models provided as input of the process. However, the commonalities and variabilities among the products are now explicitly formalized in terms of CVL. In addition, the *Product Model to SPL* process presented can be used to further evolve the variability of the SPL , decomposing new products and expressing them in terms of CVL. Next section presents the application of the approach to our industrial partner and the set of evolution scenarios encountered.

4. CASE STUDY: INDUCTION HOBS

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

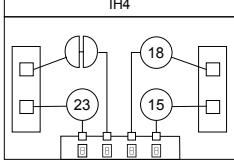
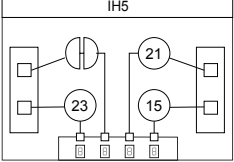
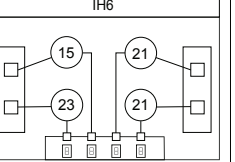
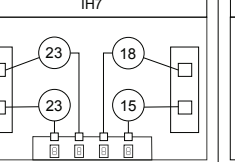
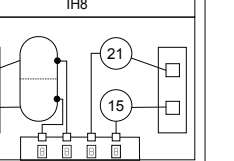
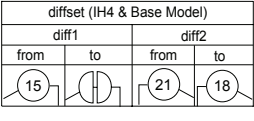
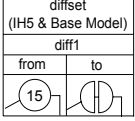
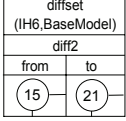
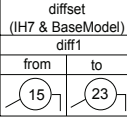
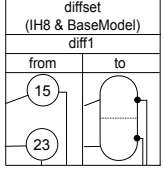
	Already existing model	Reuse existing variability	New substitution	New replacement	New placement																	
Product Model																						
diffsets generated																						
Product Model to CVL-SPL result	<table border="1" data-bbox="237 537 423 642"> <tr><td>Resolution 3</td></tr> <tr><td>Substitution 4</td></tr> <tr><td>Placement 1 Replacement 4</td></tr> <tr><td>Substitution 5</td></tr> <tr><td>Placement 2 Replacement 3</td></tr> </table>	Resolution 3	Substitution 4	Placement 1 Replacement 4	Substitution 5	Placement 2 Replacement 3	<table border="1" data-bbox="496 558 683 621"> <tr><td>NEW Resolution</td></tr> <tr><td>Substitution 4</td></tr> <tr><td>Placement 1 Replacement 4</td></tr> </table>	NEW Resolution	Substitution 4	Placement 1 Replacement 4	<table border="1" data-bbox="748 558 935 621"> <tr><td>NEW Resolution</td></tr> <tr><td>NEW Substitution</td></tr> <tr><td>Placement 4 Replacement 1</td></tr> </table>	NEW Resolution	NEW Substitution	Placement 4 Replacement 1	<table border="1" data-bbox="992 558 1195 621"> <tr><td>NEW Resolution</td></tr> <tr><td>NEW Substitution</td></tr> <tr><td>Placement 1 NEW Replacement</td></tr> </table>	NEW Resolution	NEW Substitution	Placement 1 NEW Replacement	<table border="1" data-bbox="1243 558 1455 621"> <tr><td>NEW Resolution</td></tr> <tr><td>NEW Substitution</td></tr> <tr><td>NEW Placement NEW Replacement</td></tr> </table>	NEW Resolution	NEW Substitution	NEW Placement NEW Replacement
Resolution 3																						
Substitution 4																						
Placement 1 Replacement 4																						
Substitution 5																						
Placement 2 Replacement 3																						
NEW Resolution																						
Substitution 4																						
Placement 1 Replacement 4																						
NEW Resolution																						
NEW Substitution																						
Placement 4 Replacement 1																						
NEW Resolution																						
NEW Substitution																						
Placement 1 NEW Replacement																						
NEW Resolution																						
NEW Substitution																						
NEW Placement NEW Replacement																						

Figure 5: Five Scenarios of SPL evolution

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 comparisons among models are implemented based on EMF-Compare², which is an Eclipse framework to compare instances of EMF models. To build the frontend of the SPL we have used the Graphical Modeling Project (GMP)³, a framework that provides a set of generative components and runtime infrastructures for developing graphical editors based on EMF. Finally, to add variability management capabilities to the graphical editor we have integrated the CVL tool from Sintef [5], a CVL prototype implementation that can be integrated into editor.

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 [8] approach, where each IH has been modeled modifying a copy of the most similar IH present in the collection. For instance, a modification includes taking some elements from other induction hobs and customize them (if necessary, sometimes the elements do not require further customization). Therefore, the variability present among the models has not been explicitly defined, resulting in a set of models with implicit variability among its members. 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.

Figure 4 presents the resulting SPL tool that makes use of the variability information obtained applying the *Product Model to SPL* process. Top left part presents the Induction Hobs that have already been derived from the SPL. The set of products is the same as the one used as input; however, those induction hobs have been expressed in terms of the reusable model fragments extracted through the *Product Model to SPL* process. Bottom part presents the libraries of model fragments, holding the 102 replacement model frag-

ments obtained by the approach. When deriving new products, the model fragments presented by the libraries can be reused. Finally, top right part presents the editor area, where product models can be derived and customized.

The tool contains the variability information extracted from the set of product models used as input. However, this information is extended when new product models are derived reusing existing model fragments (as the variability model needs to include the new product) and when new reusable model fragments are needed (the fragments need to be added to the model fragment libraries). For instance, one of our industrial partner engineer's creates a new empty model and populates it reusing elements from the library. Then, the engineer customizes some elements of the induction hob model using the editor and saves it. The *Product Model to SPL* process is automatically executed to include the new induction hob into the SPL, which can lead to an increment in the variability that is defined in the SPL or in the reusable model assets available to derivate further products.

Figure 5 presents five different examples that illustrates five different situations encountered when adding new models to the SPL. Each column presents one of the five examples. First row present the product model that is going to be added to the SPL. Second row shows the diffset generated when each model is compared with the Base Model (see bottom of Figure 2). Third row presents a summary of the changes that the application of *Product Model to SPL* produces over the CVL models. Next subsections present the five different scenarios.

4.1 Already existing model

First column is an example of the addition of a model that already exists in the SPL. The comparison between IH4 and the Base model produces a set of two differences (second row). When performing steps 2.2, 2.3 and 2.4, the placement, replacement and substitutions necessary to model diff1 already exists in the CVL models. Diff1 corresponds to already existing Substitution 4 (substitute Placement 1 by Replacement 4). Therefore, no placement, replacement or substitution is created for diff1. The same happens with diff2, that corresponds to Substitution 5 (sub-

¹<http://www.eclipse.org/modeling/emf/>

²<https://www.eclipse.org/emf/compare/index.html>

³<http://eclipse.org/modeling/gmp/>

stitute Placement 2 by Replacement 3). During the creation of the resolution model (step 2.5), the process detects that the resolution already exists in the SPL (Resolution 3 composed of Substitution 4 and Substitution 5). Therefore, no resolution is created as part of step 2.5.

When the step 2.5 does not involve the creation of a new resolution model (as in this scenario), denotes that the model being processed is already part of the SPL. The process *Product Model to SPL* automatically skips the inclusion of this model in order to avoid duplicates. By means of this scenario, we avoid the inclusion of redundancy into the SPL.

4.2 Model reusing existing variability

Second column is an example of the addition of a model that reuses the variability already defined in the SPL to generate a new product model. The comparison between IH5 and the Base model produces a set of one difference (second row). Execution of steps 2.2, 2.3 and 2.4 detects that diff1 corresponds to Substitution 4 (substitute Placement 1 by Replacement 4). During step 2.5 the resolution model does not exist in the SPL, therefore, a new resolution model that includes Substitution 4 is created.

When *Product Model to SPL* does not create any substitution means that already existing variability is being used to create a new product model. However, if the resolution model does not exist in the SPL, a new resolution including the substitutions identified for each diff is created. By means of this scenario, we have created a new product reusing existing variability.

4.3 Model requiring a new substitution

Third column shows the addition of a model that needs the creation of a new substitution in order to be included into the SPL. The comparison between IH6 and the Base model produces a set of only one difference (second row). Then, during step 2.2 a placement for diff1.from is identified (Placement 1). Similarly, during step 2.3 a replacement for diff1.to is identified (Replacement 4). However, during step 2.4 no existing substitution is identified, therefore a new one is created. Then, during step 2.5 a new resolution is created, holding the new substitution created in previous step.

Sometimes the placement, replacement and substitution for a given diff already exists in the SPL (as in previous scenario) while other times only the placement and replacement exists and a new substitution is created (as in this scenario). However, in both cases we are reusing already existing model fragments to create new product models. By means of this scenario we show how the existing variability is reused in the creation of new product models.

4.4 Model requiring a new replacement

Fourth column is an example of the addition of a model that requires the creation of a new replacement in order to be formalized and included into the SPL. The comparison between IH7 and the base model produces a set of one difference (second row). Step 2.2 determines that diff1.from correspond to the already existing Placement 1. By contrast, Step 2.3, determines that there is no replacement corresponding to diff1.to in the SPL models, therefore it is created. As a new replacement has been created in step 2.3, step 2.4 creates a new substitution of the Placement 1 by the just created replacement. Finally step 2.5 creates a new resolution model including the new substitution.

When the step 2.3 involves the creation of a new replacement, the next step 2.4 will always require the creation of a new substitution (as the substitution involves a new created placement, it cannot exist in the SPL). By means of this scenario, the variability defined in the SPL has been increased, including a new replacement that now is available for the construction of other models.

4.5 Model requiring a new placement

Fifth column is an example of the addition of a model that requires the creation of a new placement. The comparison between IH8 model and the base model returns a set of one difference (second row). Then, step 2.2 detects that there is no placement corresponding to diff1.from; therefore a new placement is defined over the base model. Then, in step 2.3 a new replacement defined by diff1.to is created in the SPL. As part of step 2.4, a new substitution (that substitutes the new placement by the new replacement) is created. Finally, the resolution model including the just created substitution is created as part of step 2.5.

If the step 2.2 involves the creation of a new placement, then, a new replacement (step 2.3) and a new substitution (step 2.4) will be also created. It is important to notice that during the inclusion of IH8 model into the SPL a new replacement has been created. This replacement overlaps with other existing replacements (Replacement 1 and Replacement 3), as it is defined over the same model elements as other existing placements. However, this situation does not poses a threat to the stability of the SPL models. Substitution in CVL can be restricted, to avoid situations where two overlapping placements try to be replaced. Therefore, it is safe to define overlapping placements as long as the restrictions among them are correctly defined.

5. RELATED WORK

There are several research efforts in existing literature towards the automation of the variability formalization among a set of products. However, most of them are focused on generating Feature Models (FMs) and not address CVL particularities. For instance, [2] present an approach to reverse engineering and evolve architectural FMs. In particular, they focus on plugin-based systems, projecting variability and technical constraints of plugin dependencies into an architectural FM. In [1], the authors presents a reverse-engineering tool to extract variability data from web configurators and transform them into structured data (for instance, a feature model) in a semi-automated way. The tool incorporates a component that explores the configuration space simulating users' configuration actions in order to generate more variable data to be extracted.

Other research efforts rely on the source code of the products in order to extract the variability model. In [11] the authors present a tool-supported approach for reverse engineering FMs from different sources, such as Makefiles, pre-processor declarations, and documentation. They focus on identifying parents and combine logic formulas and descriptions as complementary sources of information. In addition, [14] 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 produce feature candidates that are manually pruned (to remove non-relevant candidates). However, these approaches rely on the source code level as

input for the process, focusing in the generation of Feature Models. By contrast, our approach deals with the particularities of the CVL and is applied at model level.

In [10], the authors propose a generic framework for mining legacy product lines and automating their refactoring to contemporary feature-oriented SPLE approaches. In [7] the authors present MoVaC, an approach to identify and analyse commonalities and variability among a set of models, with the focus on the visualization of the results. In [12] the authors propose an approach to synthesize a SPL from the comparison of a set of models. The variability is extracted from the set of models and then a CVL model for the SPL is proposed. The approach is further refined in [13] to enable the inclusion of new models to the SPL. As output, a CVL model for the SPL is proposed to be manually enhanced. We further extend those works, automatically selecting a base model among the input models based on the metric desired. In addition, we have validated the approach building a SPL for an industrial environment, extracting the variability of a set of real induction hob models. Furthermore, we present the five different evolution scenarios encountered during the validation and how the approach handles them in order to evolve the variability of the SPL.

6. CONCLUSIONS

We have presented the *Model Family to SPL process*, capable of automating the formalization of the variability among a given set of similar product models. In addition, the generated SPL can be further extended in order to increase the variability specification. The presented approach has been tooled within the Eclipse environment using already existing technologies such as EMF Runtime, EMF Compare and GMP. Then, the approach has been validated with our industrial partner. Finally, we have presented the five different evolution scenarios encountered when evolving the variability specification by our industrial partner.

However, our current implementation has some limitations. For instance, the concrete syntax used to represent each of the elements from the library is not automatically produced. Therefore, some customization regarding the concrete syntax has been performed in order to present the resulting SPL tool. We plan to provide means for automating the generation of a graphical syntax following a generative approach (similar to the Graphical Modeling Project).

CVL materialization generates product models from resolution models. However, the graphical editor shows diagrams that need to be automatically generated for each resolved model. Therefore, the position of each graphical element needs to be calculated by custom layouts that automatically position each element in the correct place. Nevertheless, these limitations constitute our future work.

7. REFERENCES

- [1] E. Abbasi, M. Acher, P. Heymans, and A. Cleve. Reverse engineering web configurators. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on*, pages 264–273, Feb 2014.
- [2] 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.
- [3] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wařowski. A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13*, pages 7:1–7:8, New York, NY, USA, 2013. ACM.
- [4] D. Dhungana, T. Neumayer, P. Grunbacher, and R. Rabiser. Supporting evolution in model-based product line engineering. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 319–328, 2008.
- [5] F. Fleurey, Ø. Haugen, B. Møller-Pedersen, G. K. Olsen, A. Svendsen, and X. Zhang. A generic language and tool for variability modeling. *Technical Report SINTEF A13505*, 2009.
- [6] 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.
- [7] J. Martinez, T. Ziadi, J. Klein, and Y. le Traon. Identifying and visualising commonality and variability in model variants. In J. Cabot and J. Rubin, editors, *Modelling Foundations and Applications*, volume 8569 of *Lecture Notes in Computer Science*, pages 117–131. Springer International Publishing, 2014.
- [8] 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.
- [9] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.
- [10] 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.
- [11] S. She, R. Lotufo, T. Berger, A. Wařowski, 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.
- [12] X. Zhang, O. Haugen, and B. Moller-Pedersen. Model comparison to synthesize a model-driven software product line. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 90–99, Aug 2011.
- [13] X. Zhang, O. Haugen, and B. Moller-Pedersen. Augmenting product lines. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, volume 1, pages 766–771, Dec 2012.
- [14] 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.