Addressing Metamodel Revisions in Model-Based Software Product Lines [*]

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

San Jorge University, SVIT Research Group, Spain[1]
University of Oslo, Department of Informatics, Norway[2]
Østfold University College, Department of Information Technology, Norway[3]
jfont@usj.es larcega@usj.es oysteinh@ifi.uio.no ccetina@usj.es

Abstract

Metamodels evolve over time, which can break the conformance between the models and the metamodel. Model migration strategies aim to co-evolve models and metamodels together, but their application is not fully automatizable and is thus cumbersome and error prone. We introduce the Variable MetaModel (VMM) strategy to address the evolution of the reusable model assets of a model-based Software Product Line. The VMM strategy applies variability modeling ideas to express the evolution of the metamodel in terms of commonalities and variabilities. When the metamodel evolves, the models continue to conform to the VMM, avoiding the need for migration. We have applied both the traditional migration strategy and the VMM strategy to a retrospective case study that includes 13 years of evolution of our industrial partner, an induction hobs manufacturer. The comparison between the two strategies shows better results for the VMM strategy in terms of model indirection, automation, and trust leak.

Categories and Subject Descriptors D.2.13 [*Software Engineering*]: Reusable Software—Reuse Models

Keywords Model-based Software Product Lines, Variability Modeling, Model and Metamodel Co-evolution

1. Introduction

Model-Driven Development aims to shift the focus of software development from coding to modeling. Metamodels are used to formalize a set of concepts and the relationships among those concepts. A model conforms to a metamodel if it is expressed by the terms that are encoded in the metamodel.

Model-based Software Product Lines enable a planned reuse of software components in products that are within the same scope [14]. Commonalities and variabilities among the products are formalized into a set of models (and metamodels) using a variability language; either feature models [2] (the de facto standard for variability modeling) or Common Variability Language (CVL) [8], (recommended for adoption as a standard by the Architectural Board of the Object Management Group). Although the details are different, all share the idea of modeling commonalities and variabilities among the different products.

Similar to other software components, metamodels evolve over time [7]; however, changes that are introduced in the evolved metamodel can invalidate the models that conform to the previous version of the metamodel. To address this issue, migration strategies [3, 10, 12, 15, 17] propose co-evolving models and metamodels together to maintain consistency.

However, even though migration strategies have proven to be successful in model-based approaches, their application is not fully automatizable and can be cumbersome and error prone in large systems. Evolution is particularly critical for a successful adoption of model-based Software Product Lines (SPLs) [16].

We believe that the ideas of variability modeling can also be applied at the metamodel level to address the evolution of SPLs and at the same time avoid the issues involved with migration strategies. Our contribution is the Variable MetaModel (VMM) strategy, which enables the evolution of the metamodel without breaking conformance. In VMM, each metamodel evolution is expressed in terms of metamodel commonalities and variabilities. As a result, already existing models continue to conform to the created VMM, avoiding the need for migration and its related issues.

First, we build a retrospective case study of the evolution undergone by our industrial partner (BSH) over the last 13 years regarding the evolution of their models and metamodels. BSH is the leading manufacturer of home appliances in Europe and its induction department produces induction hobs (under the brands of Bosch and Siemens) following an MDD approach [9].

We then apply a migration strategy to the case study, migrating the models whenever a metamodel change that breaks the conformance between models and metamodels arises. Migration strategies involve the following three issues: 1) model migration introduces indirection to the models; 2) some of the steps of the migration strategy need human assistance; 3) the trust gained by models (over years of use) is lost when they are migrated.

Finally, we also apply the VMM strategy to the retrospective case study and compare both strategies (VMM and migration). The comparisons shows that the VMM strategy achieves better results than migration in terms of the three issues related to migration: 1) VMM avoids the need for migration (and the indirection introduced); 2) some of the steps of the migration strategy require human assistance while in the VMM strategy those steps are auto-

[*] Project carried out by the Research Group SVIT T92 (Consolidated Group for Applied Research). Partially supported by the Aragonian Government and the European Social Fund “Building Europe from Aragon”.

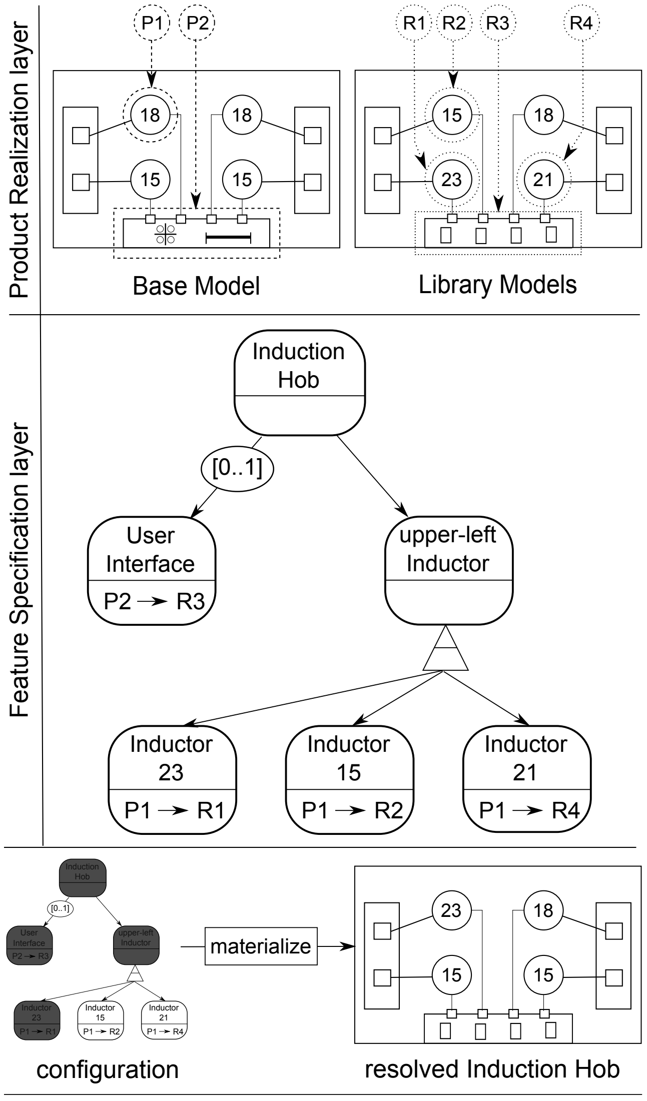


Figure 1. CVL applied to IH-DSL

matic; 3) the trust gained by models remains the same in the VMM strategy (since the model does not need to change).

2. The Induction Hobs Domain and CVL

Induction hobs use electromagnetism to generate heat that is transferred to the cookware. Traditionally, stoves feature four rounded areas that become hot when turned on. Therefore, the first Induction Hobs (IHs) created provided similar capabilities. However, the induction hob domain is constantly evolving due to the possibilities provided by the induction phenomena and the electronic components that are present.

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 real-time measurements of the consumption of the IH.

The Domain Specific Language used by our industrial partner to specify the Induction Hobs (IHDSL) is composed of 46 meta-classes, 74 references among the meta-classes 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 (top-left corner of Figure 2).

The bottom-right corner of Figure 1 shows an Induction Hob with the graphical representation of the IHDSL. It is composed of two power modules (vertical rectangles on both sides of the IH). Each of them holds two inverters (squares), which are in charge of providing the electrical supply required to generate the magnetic field. Inverters are connected to the inductors (circles), which are the elements where the magnetic field is generated. The number inside each inductor represents the diameter of the inductor. The line that connects inverters and inductors represents the channel, which transfers energy from the inverter to the inductor. The user interface of an IH has controllers to configure the power level of each inductor (the horizontal rectangle at the bottom of the IH).

The Common Variability Language (CVL) is a DSL for modeling variability in any model of any DSL based on Meta-Object Facility (MOF), which is an OMG specification to define a universal metamodel for describing modeling languages. CVL defines variants of the base model by replacing parts of the base model with model replacements that are found in a library.

The variability specification in CVL is divided across two different layers: the feature specification layer (where variability is specified following a feature model syntax [2]); and the product realization layer (where the 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. The top-left corner of Figure 1 shows the Base Model, which is a complete IH model with four inductors and a slider user interface.

The elements of the base model subject to variations are the **placement fragments** (hereinafter placements). In the top-left corner of Figure 1 there are two placements defined over the Base Model: P1, which is defined over the top-left inductor; and P2, which is defined over the user interface.

To define alternatives for a placement, we use a replacement library, which is a model described in the same DSL as the Base Model. Each alternative for a placement in the Base Model is a **replacement fragment** (hereinafter replacement). In the top-right corner of Figure 1 there are four replacements that are defined over the library model: three inductor replacements (R1, R2 and R4) and a user interface replacement (R3).

CVL defines variants of the base model by means of **fragment substitutions** (i.e. the substitution of placements by replacements). The middle part of Figure 1 shows the Feature specification layer with the substitutions. P2 can be substituted by R3 (this substitution is optional) and P1 can be substituted by R1, R2, or R4. The materialization operation of CVL executes the substitutions that are selected and produces a variation of the base model where the placements have been substituted by the replacements selected. The bottom part of Figure 1 shows an example of configuration (over the feature specification) and materialization where P1 has been substituted by R1 and P2 has been substituted by R3.

For simplicity throughout the rest of the paper, we will show the placements superimposed on the base model, even though they are defined in a separate model. Likewise, the replacements defined in the replacements library will be shown separately from the rest of the model where they are defined.

3. SPL Evolution Formalized by CVL

This section presents the retrospective case study that was extracted from the evolution of our industrial partner's models and metamodels over the last 13 years. Although the evolution data provided involves all the elements present in the initial DSL, for simplicity and

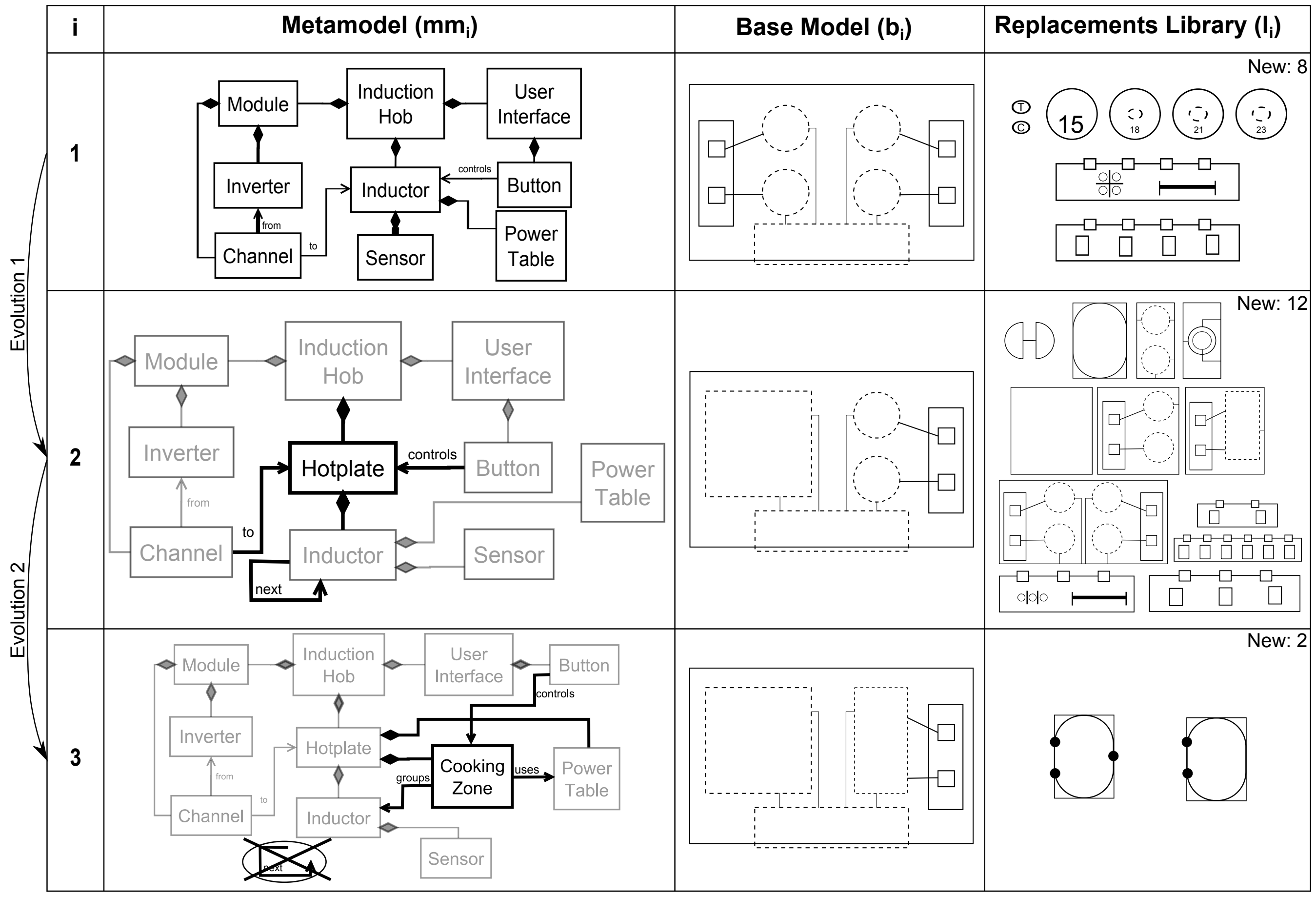


Figure 2. Model generations of the CVLSPL

due to intellectual property rights concerns, we are going to focus on the evolution related to the inductor concept.

Let MM be the set of all models that conform to the MOF language (i.e., the set of all metamodels). Let M be the set of all models. If m_i is in M and mm_i is in MM then $C(m_i, mm_i)$ means that m_i conforms to mm_i. Let $CVLSPL$ be the set of all CVL-based product lines. One such product line, $cvlspl_i$, is denoted as follows:

$$\begin{aligned} CVLSPL = \;& MM \times M \times M \\ cvlspl_i = & <mm_i\ ,\ b_i\ ,\ l_i> \end{aligned} \tag{1}$$

where mm_i is the metamodel of the DSL (conforming to MOF), b_i is the base model (over which placements for the variable parts are defined), l_i is the library of replacements for those placements, and the conformance between models $C(b_i, mm_i)$ and $C(l_i, mm_i)$ is fulfilled. In addition, let i be a consecutive index that is assigned based on when models and metamodels are created. That is, we will refer to the *Generation* i of the base model, the *Generation* i of the metamodel, the *Generation* i of the library, and the *Generation* i of the CVLSPL.

We perform a $CVLSPL$ evolution (shift from one $cvlspl_i$ generation to the next generation, $cvlspl_{i+1}$) whenever there is a *breaking and unresolvable change* (hereinafter *breaking change*) [3] in the metamodel. Breaking changes break the conformance of models and metamodel in a way that cannot be resolved by automatic means [3] (e.g., the addition of a mandatory meta-class or a restriction in the multiplicities). There are other metamodel changes that do not break the conformance of models and metamodel (e.g., the addition of an optional class) or metamodel changes that can be resolved automatically by existing approaches [3, 10, 12, 15, 17] (e.g., eliminating a property). However, in this work we will focus on the evolution triggered by breaking changes.

Figure 2 shows a summary of the CVLSPL generations and the evolutions performed. Specifically, we present three CVLSPL generations: the first row shows $cvlspl_1$, which includes the concept of inductor; the second row shows $cvlspl_2$, which includes the concept of Hotplate; the third row shows $cvlspl_3$, which includes the concept of cooking zone. The figure shows the breaking changes that were overcome by our industrial partner, such as the addition or removal of meta-elements.

Evolution 1 (from $cvlspl_1$ to $cvlspl_2$) is triggered by a new concept called Hotplate (see the first and second rows of Figure 2). A Hotplate consists of a group of inductors that can work together. There is a hierarchy (*next* relationship) among the inductors; some must be turned on before their subordinates are turned on. Therefore, we need to control the whole Hotplate (two inductors) with just one user interface controller, so the controller will now act over hotplates instead of inductors. This is reflected in the metamodel mm_2 (see the second row, first column).

There are also modifications at the model level. A new placement is created over the base model b_2 to enable substitutions of the new hotplate replacements. In addition, new replacements (l_2) that instantiate the hotplate concept are created; for example, the split hotplate (formed by two inductors, one main and one auxil-

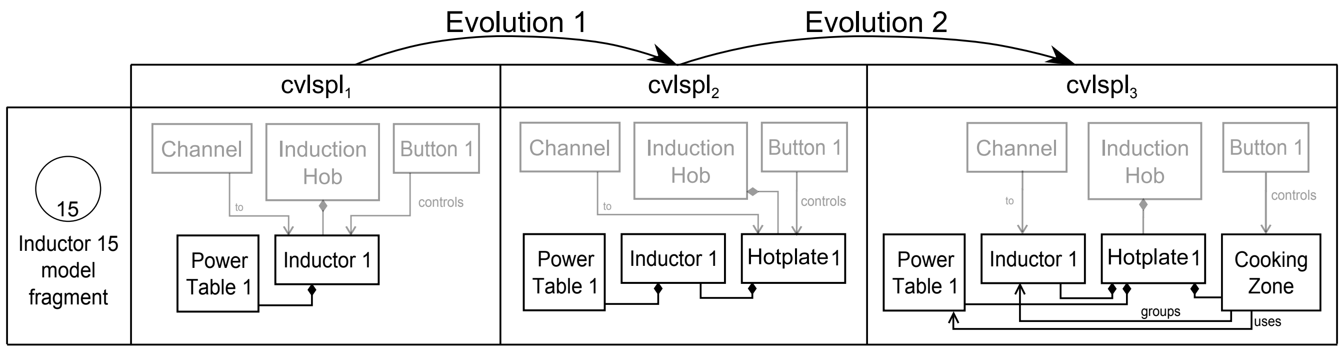


Figure 3. Migration Strategy steps

iary) or the double hotplate (formed by two inductors, requiring twice the space and power as the rest of hotplates).

Evolution 2 (from $cvlspl_2$ to $cvlspl_3$) is triggered by a new concept called cooking zone (see the second and third rows of Figure 2). Cooking zones improve the hotplate by introducing the ability to heat two different pieces of cookware at the same time and with different power levels. Now each hotplate will have cooking zones, which will be controlled by the user interface controller. As the number of combinations of inductors that are working at the same time increases, the power table is now aggregated by the hotplate, and the cooking zones use it. By means of this modification, several hotplates will share the same power tables (when the inductor configurations are equivalent). Furthermore, the hierarchy that is present among inductors is now controlled by the cooking zone (one cooking zone having the main inductor and another cooking zone having both inductors); therefore, the relationship *next* is removed from the metamodel (mm_3).

A new placement to include hotplates on both sides is created over the base model b_3. Similarly, new replacements that exercise the new concept of cooking zone are created (l_3). For instance, the pool hotplate has four inductors that are divided into two different cooking zones, which are controlled by two different buttons.

4. Motivation of the Approach

The evolution presented in Section 3 needs to be properly supported by the metamodels that are used by our industrial partner to formalize their SPL. Some of the changes presented can be addressed without breaking the conformance between the models and the metamodel, such as the creation of new model fragments or the addition of new optional elements to the metamodel. However, when we perform a breaking change to the metamodel (e.g., the hotplate and cooking zone concepts), the conformance between the models and the metamodel is lost.

Traditional migration strategies [3, 10, 12, 15, 17] propose migrating all of the models to conform to the new version of the metamodel. The migration of the SPL can be achieved by the following steps: Given a metamodel change, 1) the metamodel is upgraded to a new version introducing the new concept; 2) a model-to-model (M2M) transformation that migrates models from one version to another is created (by means of one of the existing approaches in the literature: manual specification [15], operator-based [12, 17] or metamodel matching [3, 10]); 3) existing replacements are migrated (by executing the M2M transformation obtained from step 2) to conform to the new generation of the metamodel; 4) if some common parts that are present in the Base Model have become variable, the user creates placements over the base model and extracts the model fragments as replacements; 5) new replacements are created to instantiate the new concepts that have been incorporated into the metamodel.

Let E_{mig} be the operation used to evolve a $cvlspl_i$ from a given generation i to the next generation $(i+1)$ following the migration strategy. The operation is defined as follows:

$$\begin{array}{rcl} E_{mig}: & CVLSPL & \longrightarrow \quad CVLSPL \\ E_{mig}\,(< mm_i, b_i, l_i >) & = & < mm_{i+1}, b_{i+1}, l_{i+1} > \\ & \text{where} & M2M(L_i) = L_{i+1} \end{array} \qquad (2)$$

Figure 3 presents the evolution of a model fragment following a migration strategy. Each column shows the same fragment (Inductor 15) for each of the $cvlspl_i$ generations. Although its functionality remains the same, the model is augmented to conform to each generation metamodel. In **Generation 1**, the replacement of an inductor of size 15 is represented by 2 metamodel classes (Inductor and Power Table) and can be connected to a channel and controlled by a button. In **Generation 2**, the model fragment is migrated to conform to mm_2. Hotplate 1 now aggregates the inductor and is the one controlled by the button. In this generation we need 3 classes (we add the Hotplate) to model the same functionality. In **Generation 3**, we need to include a cooking zone (enabling groups inside the same hotplate), so the model is now composed of four model elements. The three versions of the model fragment represent the same functionality: a heating element of size 15 that is connected with a channel and controlled from a button. However, there is an increase in model complexity.

Specifically, the migration of models from our industrial partner involves three related issues: **indirection**, where there is an increase in the number of elements used to model the same element of the induction hob (as in this example); **automation**, since the migration of the models cannot be performed automatically, an engineer needs to generate the M2M transformation and make decisions when applying it; **trust leak**, the modification of the model fragments (through the migrations) decreases the trust gained by those models during that generation. The fragments need to be modified to be adapted to the new metamodel, not to improve its functionality, and the modification is regarded as unnecessary and error prone. The domain of our industrial partner is constantly evolving, but the original elements are still present in new IHs. New kinds of heating elements or strategies may appear, but the simplest inductors (e.g., the inductor of size 15) are still an important part of modern IHs.

5. The Variable MetaModel Strategy Applied to the Case Study

In order to avoid the need for migration when a new generation is created, we want to build a new metamodel that supports both generations: the Variable MetaModel (VMM). For instance, mod-

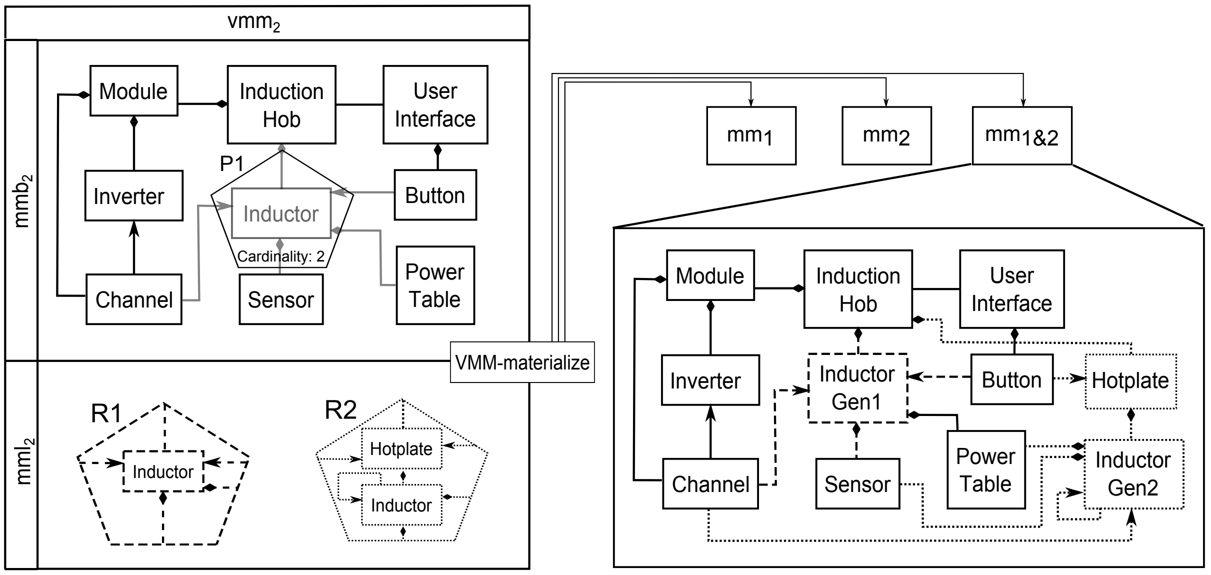


Figure 4. VMM and VMM-materialize

els that conform to Generation 1 and models that conform to Generation 2 will also conform to this VMM. A model that contains replacements from both generations will conform to the VMM.

The VMM is the result of applying CVL at the metamodel level; we have a base model in a given DSL (in this case, MOF) with placements defined over it and a library of replacements. VMM is defined as follows:

$$\begin{aligned} VMM = \ & MM \ \times MM \\ vmm_i = & < mmb_i \ , mml_i > \end{aligned} \tag{3}$$

where mmb_i is the base model at the metamodel level and mml_i is the library of replacements at metamodel level.

Similarly to CVL at the model level, we can materialize models that conform to the given DSL (in this case, MOF). Let G be the set of all generations and let $\mathcal{P}(G)$ be its power set. We define the $VMMmat$ (VMM Materialization) operation as follows:

$$\begin{aligned} VMMmat\colon \quad & VMM \quad \times \mathcal{P}(G) \longrightarrow MM \\ VMMmat(< mmb_i, mml_i >, \quad & g) \quad = mm_g \\ & where\ g \neq \emptyset \end{aligned} \tag{4}$$

That is, given a vmm_i where i generation is included in G and selecting a non-empty generation set g, $VMMmat$ retrieves the mm_g for the $cvlspl_g$ of the given generation set g.

Figure 4 (left) shows an example of VMM, the vmm_2 for generation 2. The top-left corner shows the base model (mmb_2). It is the metamodel from $cvlspl_1$, with a placement (P1) defined over the inductor. In addition, the bottom-left corner of Figure 4 shows the replacements library (mml_2), which contains two different replacements: R1 (in dashed lines) defined over the $cvlspl_1$ metamodel; R2 (in dotted lines) defined over the $cvlspl_2$ metamodel.

Figure 4 (right) shows the models produced with the vmm_2 presented. The materialization of CVL produces models that conform to the same language that the base model and replacements conform to; therefore, in this case the produced models will conform to MOF. With the library that is available (two replacements), we can produce three different models: 1) mm_1 (the metamodel of $cvlspl_1$) with a substitution of P1 by R1; 2) mm_2 (the metamodel of $cvlspl_2$) with a substitution of P1 by R2; 3) $mm_{1\&2}$ (a new metamodel with the concepts from the mm_1 and the mm_2 metamodels) with the substitution of P1 by R1 and P1 by R2.

The cardinality property of placements in CVL enables the creation of $mm_{1\&2}$. In other words, a placement can be substituted more than once. The first time that a placement is substituted, the existing references of the placement are replaced. The second time that the same placement is substituted, new references that are analogous to the existing ones need to be created. For instance, the aggregation of Inductors reference in mm_i is duplicated into an aggregation of Inductor Gen1 (in dashed lines) and aggregation of Hotplate (in dotted lines) in the $mm_{1\&2}$.

The $mm_{1\&2}$ metamodels contains concepts from both $cvlspl_1$ and $cvlspl_2$ at the same time. To achieve this, VMM renames the elements that conflict (e.g., Inductor from mm_1 and from mm_2). The advantages of this $mm_{1\&2}$ is that any model that conforms to mm_1 also conforms to $mm_{1\&2}$ and any model that conforms to mm_2 also conforms to $mm_{1\&2}$. In other words, $mm_{1\&2}$ is used when materializing IH models that contain replacements from both libraries (l_1 and l_2) and the resulting model conforms to $mm_{1\&2}$.

The vmm_2 enables the materialization of mm_1 and mm_2 that are used directly by the engineers to create new replacements. By doing so, the replacements created will conform to a specific generation, and will not include unnecessary indirection. If the functionality required for a particular replacement can be achieved with the expressiveness of a previous generation, that metamodel will be used.

Furthermore, if the engineers try to create new replacements using the $mm_{1\&2}$ directly, they could end up creating models that do not conform to either mm_1 or to mm_2. Therefore, we need to keep the original metamodels (mm_1 and mm_2) in order to enable the creation of new replacements.

5.1 Steps of the VMM Strategy

The evolution of a $cvlspl_i$ following the VMM-strategy is denoted as follows:

$$\begin{aligned} E_{VMM}\colon \quad & CVLSPL \quad \longrightarrow \quad VMM \\ & < mm_i, b_i, l_i > \longrightarrow < mmb_{i+1}, mml_{i+1} > \end{aligned} \tag{5}$$

$VMMmat$ is used with the generated vmm_i to retrieve the different CVLSPL generations needed by the company.

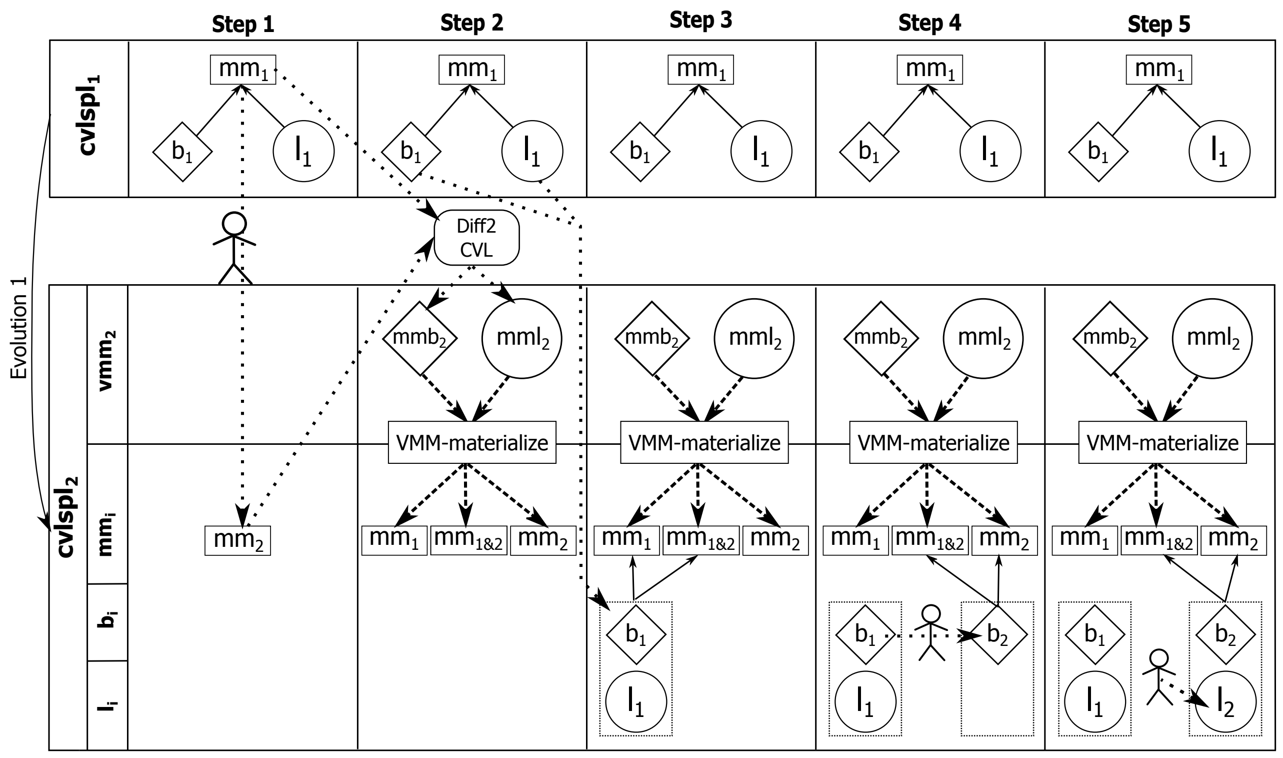


Figure 5. Steps of the VMM Strategy

Figure 5 shows the method to perform the Evolution 1 from $cvlspl_1$ to $cvlspl_2$. Each of the columns of the tables represent one step in the application of the VMM strategy. The top part shows the $cvlspl_1$ with its base model b_1 (depicted as a diamond), its metamodel mm_1 (depicted as a rectangle), and its fragment library l_i (depicted as a circle). The bottom part shows the $cvlspl_2$, the first row shows vmm_2, and the second row shows the $cvlspl_2$.

Step 1 shows the edition of the metamodel by the user. The mm_1 metamodel is edited to include the new concepts of the next generation (Hotplate), resulting in the mm_2 metamodel.

Step 2 shows our Diff2CVL operation, which is used to spot the differences between the two metamodels and to describe them in terms of a base model and replacements. Diff2CVL is built upon EMFCompare[1]. This is an eclipse plugin that provides generic support for any kind of metamodel in order to compare and merge models. The common parts of the two metamodels (mm_1 and mm_2) are included in the mmb_2 and placements are created over it for the differences between mm_1 and mm_2. Furthermore, replacements that contain these differences are created and included in the mml_2. The $VMMmat$ operation can be applied to vmm_i to obtain mm_1, mm_2, and $mm_{1\&2}$.

In Step 3, the l_1 and b_1 from $cvlspl_1$ are copied without any modification to be used in $cvlspl_2$. Both conform to the materialized mm_1, and they also conform to the materialized $mm_{1\&2}$.

In Step 4, some common parts of the base model (b_1) may become variable because of the new concepts introduced in Generation 2. In that case, the engineer edits the base model b_1 (that has been copied in the previous step) from the $cvlspl_2$ to extract the variable parts as replacements.

In Step 5, the engineer creates new replacements that instantiate the new concepts of this generation (Hotplate) and includes them in l_2. These new replacements conform to mm_2, and they also conform to $mm_{1\&2}$.

Following the above steps, we can evolve the SPL from one generation to the next, while avoiding the need for migrating existing fragments. Then, when the engineer wants to create new replacements, the engineer will be able to use the metamodel of just one generation and not the $mm_{1\&2}$. As a result, the engineer can create replacements for the most recent generation (using mm_2) to instantiate the new concepts of that generation. In contrast, the engineer can use the previous generation metamodel (mm_1) to create replacements that do not exercise the expressiveness provided by the new generation, avoiding the overcharge of the model (as the case of the motivating example, see Section 4). When materializing an IH model containing replacements from both generations (l_1 and l_2), the resulting IH model will conform to $mm_{1\&2}$.

In addition, the recursion capabilities of CVL enable us to create placements inside a replacement and hence apply the VMM strategy to further generations. That is, when creating the next generation, the step 2 of the process could end up in the creation of a new replacement that includes previously defined placements (if the replacement is not common for both metamodels).

5.2 Resulting Models after Applying VMM Strategy

Figure 6 shows an overview of our industrial partner's $CVLSPL$ models (rows) after applying the VMM strategy to manage Evolution 1 and Evolution 2 (columns).

In **Evolution 1** a new concept (hotplate) is introduced (see the first and second columns). This concept affects the inductor, which is now aggregated by the hotplate; therefore, we apply the method explained above to perform Evolution 1. Diff2CVL produces a base model (mmb_2) that contains a placement (P1) with cardinality 2 (i.e., it can be replaced up to two times). Diff2CVL also produces mml_2, which contains two replacements: R1 (which holds the

[1] https://www.eclipse.org/emf/compare/

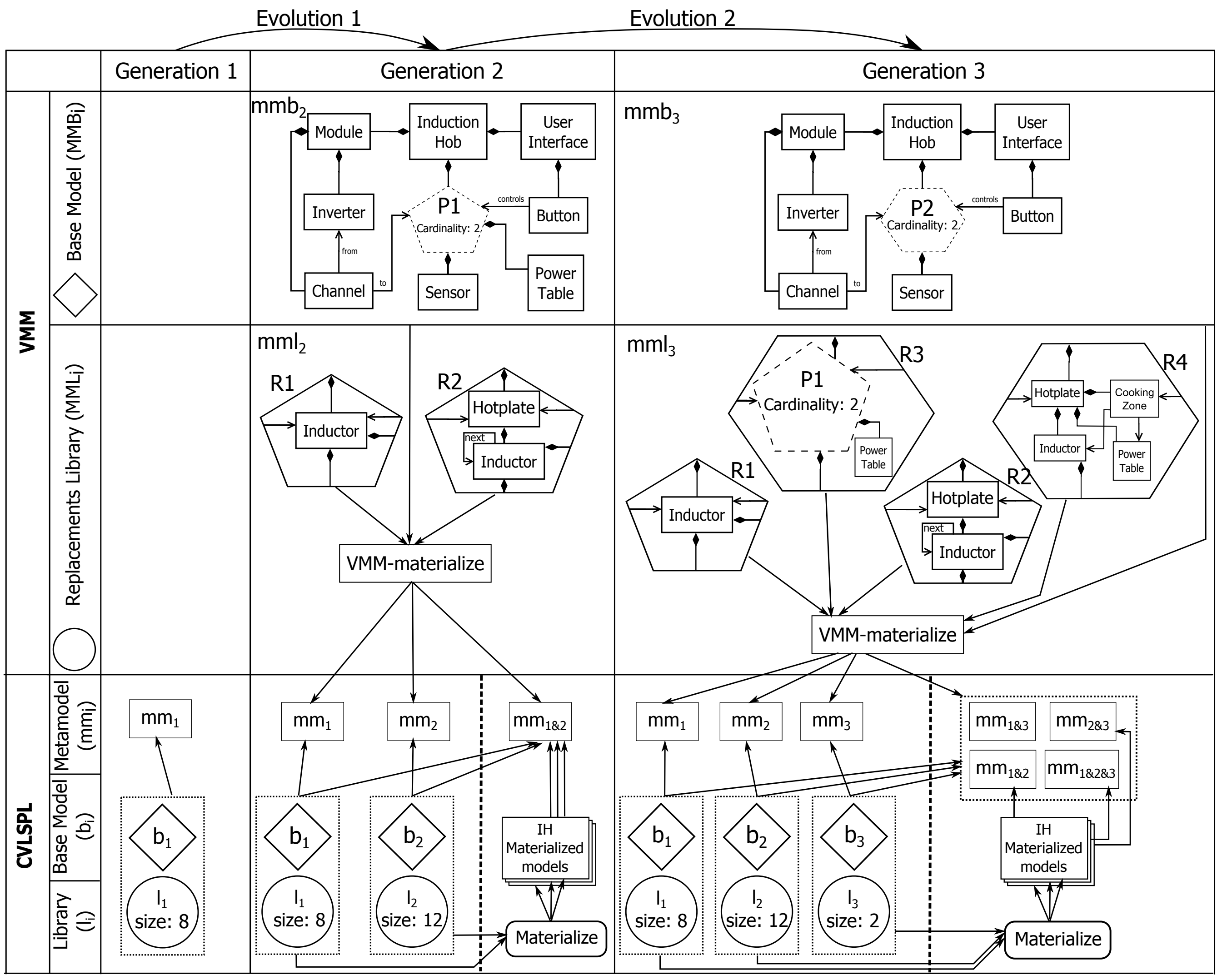


Figure 6. IHDSL Metamodel level for each generation of the CVLSPL

particularities of Gen1) and R2 (which holds the particularities of Gen2). $VMMmat$ operation can be applied to those models to produce three different metamodels: 1) the substitution of P1 by R1 produces mm_1; 2) the substitution of P1 by R2 produces mm_2; 3) the substitution of P1 twice, by R1 and by R2 produces $mm_{1\&2}$.

When creating new fragments, the engineer must stick to only one generation in order to create a valid fragment. In other words, fragments must conform to a specific metamodel generation, either mm_1 or mm_2. As a result the engineer can create replacements only using concepts from mm_1, avoiding the indirection introduced by the migration strategy (see Section 4).

When materializing an IH model that contains replacements from both generations (l_1 and l_2), the resulting IH model conforms to $mm_{1\&2}$. Overall, vmm_2 enables the materialization of IH models with replacements from both generations (l_1 and l_2), while at the same time allowing the creation of fragments pertaining to one generation (either conforming to mm_1 or to mm_2).

In **Evolution 2** a new *breaking change* that introduces the concept of cooking zones occurs (see the second and third columns). Similarly to Evolution 1, we apply the method to perform Evolution 2 (from Generation 2 to Generation 3).

The CVL capabilities of recursion (placements inside replacements) and cardinalities over the placements applied to the metamodel level have proven to provide enough expressiveness to overcome all of the evolution situations of our industrial partner over 13 years. In addition, the VMM strategy of this work enables our industrial partner's engineers to derive products by means of replacements from any generation, while avoiding the disadvantages of migrating the replacements after each evolution.

5.3 Derivation of SPL Products after Applying VMM

The VMM strategy has been tooled within the Eclipse environment and integrated into our industrial partner's SPL. The resulting tool is used by our industrial partner (BSH, the leading manufacturer of home appliances in Europe) to generate the firmware of their Induction Hobs (sold under the brands of Bosh and Siemens). An example of the resulting tool in action can be seen here [2]. This section present an example of using the SPL evolved with the VMM strategy: an engineer of our industrial partner deriving a new product.

[2] http://www.carloscetina.com/variablemetamodel.htm

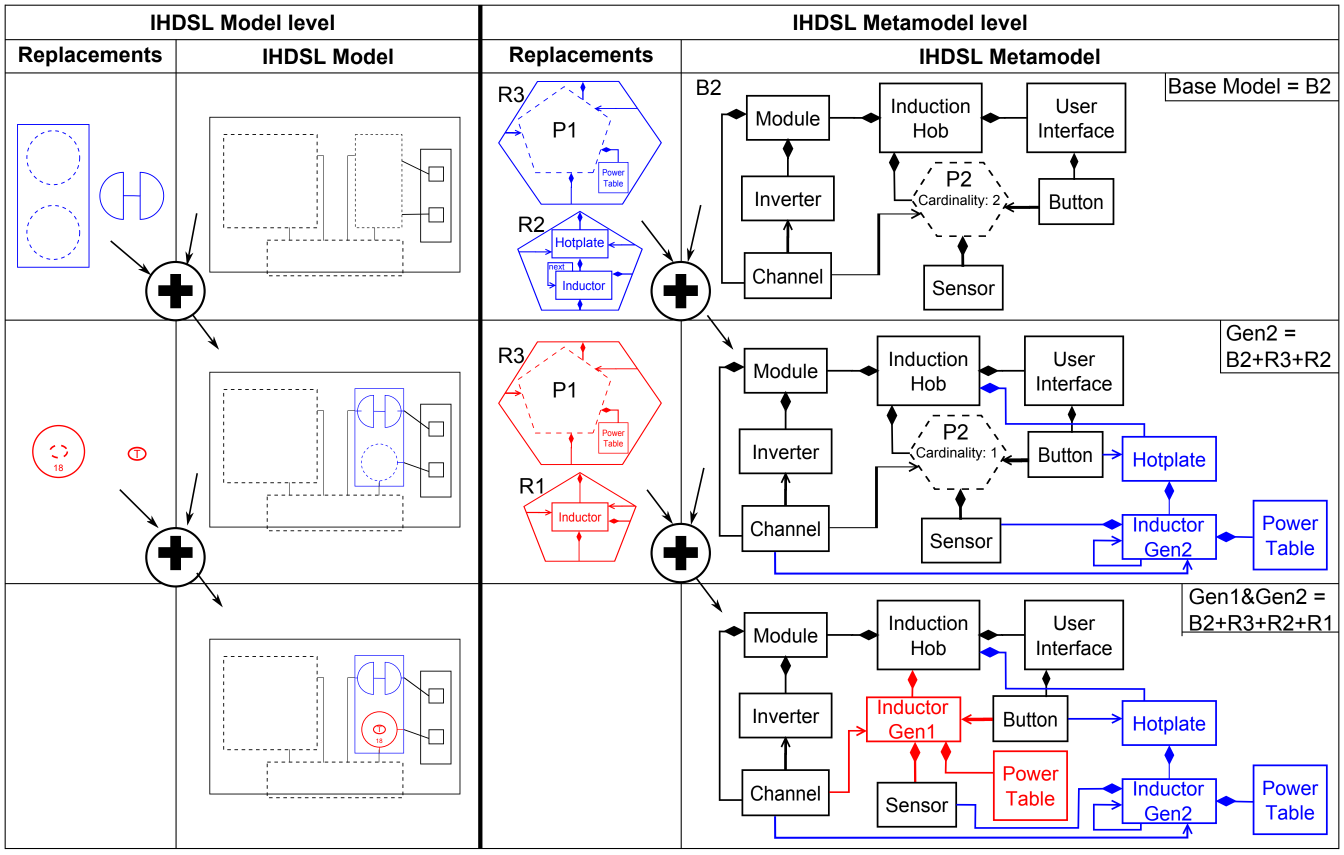


Figure 7. Fragment substitutions to derive a SPL product

The engineer will act at the model level, choosing which replacements should be substituted in the base model and building the Induction Hob model; in the meantime, at the metamodel level, the metamodel is built up automatically reflecting those model level substitutions. Each time a replacement is chosen by the engineer (at model level), the replacement (at metamodel level) corresponding to the replacement chosen by the engineer at the model level will be automatically substituted in the base model metamodel (only if it is the first occurrence of that generation).

Figure 7 show an example of the derivation when the SPL is in Generation 3. At the model level (the first and second columns), the engineer chooses the replacements (the first column) for the placements of the base model (the second column), while at the metamodel level (the third and fourth columns), the metamodel replacements (the third column) are automatically substituted for the placements of the Base Model (the fourth column). Note that the metamodel level elements presented in Figure 7 (the third and fourth columns) and the metamodel level elements presented in Figure 6 (the third column) are the same.

The first row in Figure 7 shows the first substitution of the product derivation: the engineer can use replacements from the three different generations available. In this case, the engineer is going to use replacements from the second generation (the first column). The base model of the current generation (the second column) is used. The metamodel level has the replacements R2 and R3 (third column) that correspond to the model level replacements, and the metamodel base B2 (fourth column) with all the common elements from all of the generations.

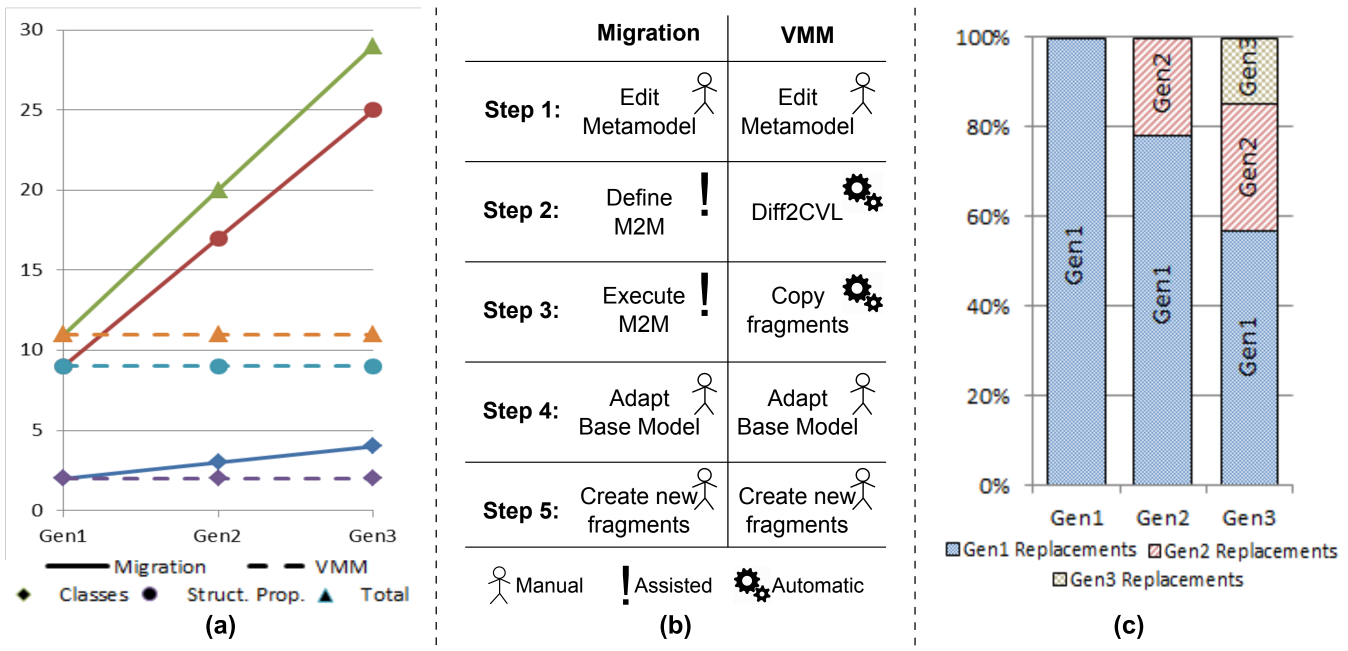
The second row in Figure 7 shows the result of the first fragment substitution. The fragments chosen by the engineer have been substituted at the model level (the second column). At metamodel level, corresponding fragments have been automatically substituted (the fourth column), resulting in the Generation 2 metamodel (Gen2). Now, if more model level replacements from Generation 2 are added, the metamodel does not vary (it only varies the first time that a generation is used). We repeat the operation with more replacements: this time they belong to Generation 1. At the metamodel level, the corresponding metamodel level replacements R1 and R3 are used.

The third row shows the results of the second fragment substitution. The model now has elements from two SPL generations; therefore, the metamodel has automatically been increased to be the combination of those two generations (Gen1&Gen2) maintaining the conformance between the model (the second column) and the metamodel (the fourth column). The engineer then performs more fragment substitutions until all the placements of the IH model are substituted; the metamodel is automatically increased as necessary.

The VMM strategy of this work enables our industrial partner's engineers to derive products by means of replacements from any generation, while avoiding the disadvantages of migrating the replacements after each evolution. The following Section discusses the advantages and disadvantages of each of the strategies, taking into account the experience acquired from our industrial partner.

6. Discussion

We have applied both strategies to the retrospective of 13 years of our industrial partner's SPL models. In this paper, we only show a simplification of the evolution related to the inductor concept even though we have applied it to all of the concepts. This involves about



	Migration	VMM
Step 1:	Edit Metamodel	Edit Metamodel
Step 2:	Define M2M	Diff2CVL
Step 3:	Execute M2M	Copy fragments
Step 4:	Adapt Base Model	Adapt Base Model
Step 5:	Create new fragments	Create new fragments

Figure 8. Comparison between Migration and VMM Strategy

32 different IH models composed of approximately 72 different model replacements (each of them composed of multiple model elements). The average number of model elements of a fragment replacement is 43, while the average number of elements of an IH model is about 470. Figure 8 shows a summary of the comparison obtained from the collaboration with our industrial partner of both, the migration strategy and the VMM strategy, in terms of three dimensions: (a) indirection, (b) automation and, (c) trust leak.

6.1 Indirection

Indirection refers to an increase in model elements in order to conform to an evolved metamodel while keeping the same functionality. For instance, the inductor that migrates into a hotplate and then into a cooking zone (see Section 4).

Figure 8 (a) shows the comparison of both strategies in terms of the indirection that is present in the replacements. The graph shows the number of model elements (classes and structural properties) used in each generation to represent an inductor. In the migration strategy (solid lines), the inductor grows from a total of 11 elements in Gen 1 to a total of 29 elements in Gen 3. This growth trend is common for all of the concepts studied in this work. Although it is out of the scope of this paper, there are transformations based on the metamodel to transform IHDSL models into code, and this indirection requires modifications and produces an increase in the complexity of the transformations and the code generated. In contrast, the VMM strategy (dashed lines) avoids the migration of replacements, and the number of elements needed to represent the inductor concept (11) remains the same over all of the generations.

6.2 Automation

Depending on the degree of involvement of the user, the execution of the steps of both strategies can be either manual, assisted, or automatic. A step is automatic when it is done without user intervention; it is assisted when user must help in the process; and it is manual when the whole process is performed by the user.

Figure 8 (b) shows the comparison of the two strategies in terms of automation for each of the steps of the strategies. Step 1 (Edit Metamodel) is the same for both strategies and must be performed manually. Step 2 is different; the migration strategy requires the definition of a M2M transformation. With the options that are available (manual [15], operator-based [12, 17] or metamodel matching [3, 10]), the process is, at best, assisted [3, 11]. In contrast, in the VMM Strategy Step 2 (Diff2CVL) is fully automatizable, (CVL applied to the model and the metamodel level enabled us to resolve all kind of changes presented by [3] in an automatic way). Step 3 in the migration strategy is the execution of the M2M transformation. Breaking changes (e.g., the addition of obligatory properties) are not automatically resolvable ([3, 11]), so the step needs to be assisted. In contrast, in the VMM strategy replacements are used "as is": no migration is required and only an automatic copy is performed. Finally Steps 4 (Adapt base model) and 5 (Create new replacements) are performed manually in both strategies.

6.3 Trust Leak

Models are used to produce code: once they have been used repeatedly on many IHs, they gain the trust of our industrial partner's engineers. However, when the replacements are modified, there is a loss in this trust on the part of the engineers, which has been reported as *trust leak.*

Figure 8 (c) shows a comparison of both strategies in terms of *trust leak*. The graph shows the weight of the replacements of each generation in relation to the total number of products created with the SPL (i.e., average percentage of replacements from each generation present in the induction hobs taking into account all the IHs derived from the SPL). For instance, using the migration strategy for Evolution 2 (from Gen2 to Gen3), the replacements that represent 83% of the total products built need to be migrated (58% of them twice, from Gen1 to Gen2 and then to Gen3). It turns out that the replacements from Generation 1 are the ones that are most frequently used to build IHs (in all generations), and they are also the ones that require more migrations when following the migration strategy. Therefore, they are the replacements that have the highest level of trust leak. In contrast, when using the VMM strategy there is no need to migrate replacements, thus avoiding the trust leak.

7. Related Work

To the best of our knowledge, there are no works that address the evolution of SPLs using variability modeling ideas at the metamodel level; However, there are research efforts on SPL evolution that can complement model-based SPL evolution.

In [1] Batory et al. present the AHEAD model, based on the step-wise refinement paradigm, enables the synthesization of multiple complex programs from a simple program. In AHEAD the software is expresed as nested sets of equations describing feature refinements. The composition function (specific for each kind of asset) is used to stack the refinements applied to the base program to produce the different variants. However we do not focus on how to specify variants of the base product, the main focus in our approach is to avoid the migration of the models from one generation to the next by applying variability at the metamodel level.

Dhungana et al. [6] present an approach that is based on model fragments applied at the model level. They provide tool support for the automated detection of changes, facilitating metamodel evolution and propagating changes in the domain to already existing variability models. However, in contrast to our approach, they do not use fragments at the metamodel level having to update their fragments when changes occur at the metamodel level.

Deng et al. [5] argue that adding new requirements to a model-based Product Line Architecture (PLA) often causes invasive modifications to the PLA‘s component frameworks and DSLs. To address these modifications, they show how structural-based model transformations help maintain the stability of domain evolution by automatically transforming domain models. Although the details are different, their approach is similar to the *migration strategy* with support of model transformations. However, our work shows that, in the case of a CVLSPL, the *VMM strategy* offers the best results.

Creff et al. [4] propose an incremental evolution by extension of the product line. They aim to benefit from the investments supplied during the product derivation and *re-invest* them into the SPL models. Specifically, they introduce an assisted feedback algorithm to extend the SPL to emerging product derivation requirements. We believe that their feedback algorithm could be tailored to help in the detection for the need of new metamodel changes (new SPL Generations) when product derivations occur, triggering our *VMM strategy* to address the evolution at the metamodel level.

Passos et al. [13] developed a vision of software evolution that is based on a feature-oriented perspective. They provided a feature-oriented project management and system development platform that supports traceability and analyses. In our work, the SPL is specified by means of base models, fragment substitution and metamodel expressiveness. However, we can represent the variability model of our industrial partner's SPL by means of a feature model, therefore strategy can benefit from the analysis and traceability of the work of Passos et al. [13].

8. Conclusions

The CVL capabilities of recursion (placements inside replacements) and cardinalities over the placements applied to the metamodel level have proven to provide enough expressiveness to overcome all the evolution situations of our industrial partner over 13 years. In addition, the VMM strategy of this work enables our industrial partner's engineers to derive products by means of replacements from any generation, while avoiding the disadvantages of migrating the replacements after each evolution.

This work indicates that the VMM achieves better results than the migration strategy in domains like the domain of our industrial partner in terms of indirection, automation, and trust leak. Furthermore, using already existing variability management approaches (like CVL) enables us to bring efforts from the variability research community to address the evolution challenge. Nevertheless, there are still open issues (e.g., evolutions that turn variabilities into commonalities) that will be addressed in our work in the future.

References

[1] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 187–197, Washington, DC, USA, 2003. IEEE Computer Society.

[2] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems Journal*, 35(6):615–636, 2010.

[3] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. In *Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE*, pages 222–231, 2008.

[4] S. Creff, J. Champeau, J.-M. Jézéquel, and A. Monégier. Model-based product line evolution: an incremental growing by extension. In *16th International Software Product Line Conference*, SPLC '12, NY, USA, 2012. ACM.

[5] G. Deng, D. C. Schmidt, A. Gokhale, J. Gray, Y. Lin, and G. Lenz. Evolution in model-driven software product-line architectures. *Designing Software-Intensive Systems: Methods and Principles*, 2008.

[6] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software*, 83(7):1108–1122, 2010.

[7] J.-M. Favre. Meta-model and model co-evolution within the 3d software space. In *In Procedings of the International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA) at ICSM*, pages 98–109, Amsterdam, Netherlands, September 2003.

[8] 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.

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

[10] K. Garcés, F. Jouault, P. Cointe, and J. Bézivin. Managing model adaptation by precise detection of metamodel changes. In R. F. Paige, A. Hartman, and A. Rensink, editors, *Model Driven Architecture - Foundations and Applications*, volume 5562 of *Lecture Notes in Computer Science*, pages 34–49. Springer Berlin Heidelberg, 2009.

[11] B. Gruschko, D. Kolovos, and R. Paige. Towards synchronizing models with evolving metamodels. In *Proceedings of the International Workshop on Model-Driven Software Evolution*, 2007.

[12] M. Herrmannsdoerfer, S. Benz, and E. Juergens. Cope - automating coupled evolution of metamodels and models. In S. Drossopoulou, editor, *ECOOP 2009 Object-Oriented Programming*, volume 5653 of *Lecture Notes in Computer Science*, pages 52–76. Springer Berlin Heidelberg, 2009.

[13] L. Passos, K. Czarnecki, S. Apel, A. Wasowski, C. Kästner, J. Guo, and C. Hunsen. Feature-oriented software evolution. In *7th International Workshop on Variability Modelling of Software-intensive Systems*, Italy, 2013. ACM.

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

[15] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. Polack. An analysis of approaches to model migration. In *Proc. Joint MoDSE-MCCM Workshop*, pages 6–15, 2009.

[16] M. Svahnberg and J. Bosch. Evolution in software product lines: Two cases. *Journal of Software Maintenance*, 11(6):391–422, Nov. 1999.

[17] G. Wachsmuth. Metamodel adaptation and model co-adaptation. In E. Ernst, editor, *ECOOP 2007 Object-Oriented Programming*, volume 4609 of *Lecture Notes in Computer Science*, pages 600–624. Springer Berlin Heidelberg, 2007.