Incremental Reconfiguration of Product Specific Use Case Models for Evolving Configuration Decisions

Ines Hajri[1], Arda Goknil[1], Lionel C. Briand[1], and Thierry Stephany[2]

[1] SnT Centre for Security, Reliability and Trust, University of Luxembourg
{ines.hajri, arda.goknil, lionel.briand}@uni.lu
[2] International Electronics & Engineering (IEE), Contern, Luxembourg
thierry.stephany@iee.lu

Abstract. [Context and motivation] Product Line Engineering (PLE) is increasingly common practice in industry to develop complex systems for multiple customers with varying needs. In many business contexts, use cases are central development artifacts for requirements engineering and system testing. In such contexts, use case configurators can play a significant role to capture variable and common requirements in Product Line (PL) use case models and to generate Product Specific (PS) use case models for each new customer in a product family. **[Question/Problem]** Although considerable research has been devoted to use case configurators, little attention has been paid to supporting the incremental reconfiguration of use case models with evolving configuration decisions. **[Principal ideas/results]** We propose, apply, and assess an incremental reconfiguration approach to support evolving configuration decisions in PL use case models. PS use case models are incrementally reconfigured by focusing only on the changed decisions and their side effects. In our prior work, we proposed and applied Product line Use case modeling Method (PUM) to support variability modeling in PL use case diagrams and specifications. We also developed a use case configurator, PUMConf, which interactively collects configuration decisions from analysts to generate PS use case models from PL models. Our approach is built on top of PUM and PUMConf. **[Contributions]** We provide fully automated tool support for incremental configuration as an extension of PUMConf. Our approach has been evaluated in an industrial case study in the automotive domain, which provided evidence it is practical and beneficial.

Keywords: Product Line Engineering, Use Case-Driven Development.

1 Introduction

Product Line Engineering (PLE) is becoming common practice in many domains such as automotive and avionics, due to the increasing complexity of software systems that warrant better support for reusable software artifacts. In such domains, many business contexts are use case-driven where use cases are the main artifacts driving requirements engineering and system testing practices [1] [2] [3]. This is also the case for the industrial context of our work, IEE [4], a leading supplier of embedded systems in the automotive domain. The current development practice at IEE is use case-driven and based on clone-and-own reuse [5]. To develop a new product in a new project, IEE

analysts elicit requirements as a use case diagram and its accompanying use case specifications. For each new customer of the product, they need to clone the current models, and negotiate variabilities with the customer to produce new use case models. This is a manual, error prone, and time-consuming practice since variability information is not explicitly represented.

The need for PLE support in the context of use case-driven development has already been acknowledged and several product line use case modeling and configuration approaches have been proposed [6] [7] [8]. Existing approaches rely on feature modeling, including establishing and maintaining traces between features and use case models [9]. Due to limited resources, IEE, as well as other software development companies, find such additional traceability and maintainability effort to be impractical. In addition, existing use case configurators (e.g., [6] [7] [8]) do not support incremental reconfiguration of use case models resulting from changes in configuration decisions, e.g., a selected variant use case being unselected.

In practice, for example at IEE and for a variety of reasons, analysts manually assign traces from the configured use case models to other software and hardware specifications as well as to the customers' requirements documents for external systems [10]. Furthermore, configuration decisions frequently change, resulting in the reconfiguration of Product Specific (PS) use case models. When the use case models are reconfigured for all decisions, including unchanged and unaffected decisions, manually assigned traces are lost. The analysts need to reassign all the traces after each reconfiguration. It is therefore vital to enable the incremental reconfiguration of use case models focusing only on changed decisions and their side-effects. With such support, the analysts could then reassign traces only for the parts of the reconfigured models impacted by decision changes. Our main motivation is to preserve the unimpacted parts of the PS use case models for evolving configuration decisions, thus avoiding manual effort during reconfiguration such as manual updating of traces from PS models to other documents.

In our previous work [11], we proposed and assessed the Product line Use case modeling Method (PUM) to support variability modeling in Product Line (PL) use case diagrams and specifications, without making use of feature models, thus avoiding unnecessary modeling and traceability overhead. PUM includes existing PL extensions for use case diagrams [12] [13] and, for modeling variability in use case specifications, we introduced new extensions for the Restricted Use Case Modeling method (RUCM) [14]. Building on this, we developed a use case-driven configuration approach [15] supporting three crucial activities. First, the analyst is guided to make configuration decisions in an appropriate order. Second, the consistency of configuration decisions is ensured by automatically identifying contradicting decisions. Third, PS use case diagram and specifications are automatically generated from PL models and configuration decisions. Our configuration approach is supported by a tool, *PUMConf*, integrated with IBM DOORS.

In this paper, we propose, apply and assess an incremental reconfiguration approach, based on PUM and PUMConf, to support the evolution of configuration decisions for PL use case models. We do not address here evolving PL use case models, which is an entirely different problem and needs to be treated in a separate approach. In our proposed solution, the PS use case diagram and specifications are incrementally reconfigured by focusing only on the changed configuration decisions and their side effects. To

do so, we implemented a model differencing pipeline which identifies decision changes to be used in the regeneration of PS models. There are two sets of decisions: (i) the set of previously made decisions used to initially generate the PS use case models and (ii) the set of decisions including decisions changed after the initial generation of the PS models. Our approach compares the two sets to incrementally regenerate the PS use case models. We extended our configurator, *PUMConf*, to fully automate our approach. We also report an industrial case study demonstrating its applicability and benefits.

This paper is structured as follows. In Section 2, we discuss the related work. Section 3 provides a short overview of the background on *PUM* and *PUMConf*, proposed in our previous work, on which this paper builds. In Section 4, we provide an overview of the approach. Sections 5 and 6 provide the details of the core technical parts of our approach. Sections 7 and 8 present our tool support and industrial case study along with results and lessons learned. We conclude the paper in Section 9.

2 Related Work

Several use case-driven configuration approaches were proposed in the literature (e.g., [6] [7] [8]). These approaches do not support incremental reconfiguration of use cases for changes in configuration decisions. There are also more general configuration approaches that can be customized to configure PS use case models. For instance, DOPLER [16] supports capturing variability information as a variability model, and modeling any type of artifact as asset models. Variability and asset models are linked by using trace relations. Heider et al. [17] [18] propose an approach as an extension of DOPLER to identify the impact of changes of variability information on products. For a change in a variability model of a product line, the approach identifies whether configuration decisions for the existing products need to be changed as well. Then, it reconfigures all the products in the product line and also compares the reconfigured products with the previous version to inform the analysts about the differences in the products. However, it focuses on changes in variability information, not changes in decisions. It is also not incremental, limiting its applicability, as the reconfiguration encompasses all the decisions, not only the affected ones.

Considerable attention in the model-driven engineering research community has been given to incremental model generation/transformation for model changes (e.g., [19] [20] [21]), and this line of work has inspired initiatives in many software engineering domains. For instance, Vogel et al. [22] use incremental model transformation techniques for synchronizing runtime models by integrating a general-purpose model transformation engine into their runtime modeling environment. Bidirectional model transformations are employed by Eramo et al. [23] to support the synchronization and interoperability of architecture models for architecture model changes. Alternatively, we could also have employed a generic model transformation engine and language to implement the incremental generation of PS use case models. Compared to model transformation languages, in terms of loading, matching and editing text in natural language, Java provides much more flexibility for handling plain text use case specifications. As a result, we used Java to implement the generation of PS use case models in our prior work [15], and also to implement the incremental reconfiguration of PS models as

a model differencing and reconfiguration pipeline (see Section 4). To the best of our knowledge, our approach is the first work which supports incremental reconfiguration of PS use case models for evolving configuration decisions in a product family.

3 Background

In this section we give the background information about elicitation of PL use case diagram and specifications (Section 3.1), and our configuration approach (Section 3.2).

In the rest of the paper, we use Smart Trunk Opener (STO) as a case study. STO is a real-time automotive embedded system developed by IEE. It provides automatic, hands-free access to a vehicle's trunk, in combination with a keyless entry system. In possession of the vehicle's electronic remote control, the user moves her leg in forward and backward directions at the vehicle's rear bumper. STO recognizes the movement and transmits a signal to the keyless entry system, which confirms that the user has the remote. This allows the trunk controller to open the trunk automatically.

3.1 Elicitation of Variability in PL Use Cases

Elicitation of PL use case models is based on the Product line Use case modeling Method (PUM) [11]. In this section, we give a brief description of the PUM artifacts.

Use Case Diagram with PL Extensions For use case diagrams, we employ the PL extensions proposed by Halmans and Pohl [12] [13] since they support explicit representation of variants, variation points, and their dependencies (Fig. 1). We do not introduce any further extensions.

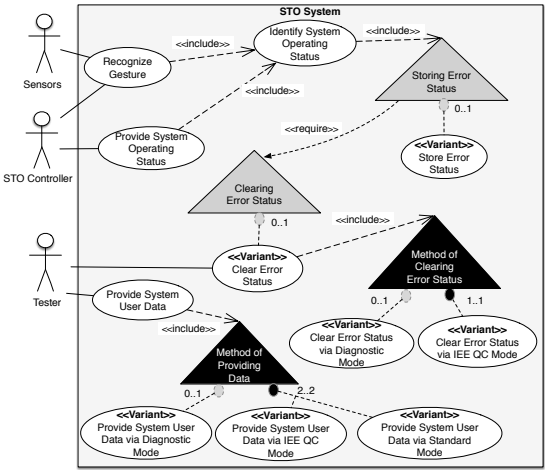

Fig. 1. Part of the PL Use Case Diagram for STO

A use case is either *Essential* or *Variant*. Variant use cases are distinguished from essential (mandatory) use cases, i.e., mandatory for all the products in a product family, by using the 'Variant' stereotype. A variation point given as a triangle is associated to one, or more than one use case using the 'include' relation. The mandatory variation points indicate where the customer has to make a selection for a product (the black triangles in Fig. 1). A 'tree-like' relation, containing a cardinality constraint, is used to express relations between variants and variation points, which are called *variability relations*. The relation uses a [min..max] notation in which *min* and *max* define the minimum and maximum numbers of variants that can be selected for the variation point. A variability relation is optional where ($min = 0$) or ($min > 0$ and $max < n$); n is the number of variants in a variation point. A variability

relation is mandatory where ($min = max = n$). Optional and mandatory relations are depicted with light-grey and black filled circles, respectively (Fig. 1). For instance, the 'Provide System User Data' essential use case has to support multiple methods of providing data where the methods of providing data via IEE QC mode and Standard mode are mandatory. In addition, the customer can select the method of providing data via diagnostic mode. In STO, the customer may decide the system does not store the errors determined while the operating status is being identified (see the 'Storing Error Status' optional variation point in Fig. 1). The extensions support the dependencies *require* and *conflict* among variation points and variant use cases [13]. Based on *require* in Fig. 1, the selection of the variant use case in 'Storing Error Status' implies the selection of the variant use case in 'Clearing Error Status'. Further variability information is given in PL use case specifications. For instance, only PL use case specifications indicate in which flows of events a variation point is included.

Restricted Use Case Modeling (RUCM) and its Extensions This section introduces the RUCM template and its PL extensions which we proposed. RUCM provides restriction rules and keywords constraining the use of natural language [14]. Since RUCM was not designed for PL modeling, we introduced some PL extensions (see Table 1). In RUCM, use cases have basic and alternative flows (Lines 2, 8, 13, 16, 22, 27, 33 and 38). In Table 1, we omit some alternative flows and some basic information such as actors and pre/post conditions.

Table 1. Some STO Use Cases in the extended RUCM

1	**USE CASE** Recognize Gesture
2	**1.1 Basic Flow**
3	1. INCLUDE USE CASE Identify System Operating Status.
4	2. The system VALIDATES THAT the operating status is valid.
5	3. The system REQUESTS the move capacitance FROM the sensors.
6	4. The system VALIDATES THAT the movement is a valid kick.
7	5. The system SENDS the valid kick status TO the STO Controller.
8	**1.2 <OPTIONAL>Bounded Alternative Flow**
9	RFS 1-4
10	1. IF voltage fluctuation is detected THEN
11	2. RESUME STEP 1.
12	3. ENDIF
13	**1.3 Specific Alternative Flow**
14	RFS 2
15	1. ABORT.
16	**1.4 Specific Alternative Flow**
17	RFS 4
18	1. The system increments the OveruseCounter by the increment step.
19	2. ABORT.
20	
21	**USE CASE** Identify System Operating Status
22	**1.1 Basic Flow**
23	1. The system VALIDATES THAT the watchdog reset is valid.
24	2. The system VALIDATES THAT the RAM is valid.
25	3. The system VALIDATES THAT the sensors are valid.
26	4. The system VALIDATES THAT there is no error detected.
27	**1.4 Specific Alternative Flow**
28	RFS 4
29	1. INCLUDE <VARIATION POINT: Storing Error Status>.
30	2. ABORT.
31	
32	**USE CASE** Provide System User Data
33	**1.1 Basic Flow**
34	1. The tester SENDS the system user data request TO the system.
35	2. INCLUDE <VARIATION POINT : Method of Providing Data>.
36	
37	<**VARIANT**>**USE CASE** Provide System User Data via Standard Mode
38	**1.1 Basic Flow**
39	V1. <OPTIONAL>The system SENDS calibration TO the tester.
40	V2. <OPTIONAL>The system SENDS sensor data TO the tester.
41	V3. <OPTIONAL>The system SENDS trace data TO the tester.
42	V4. <OPTIONAL>The system SENDS error data TO the tester.
43	V5. <OPTIONAL>The system SENDS error trace data TO the tester.

A basic flow describes a main successful path that satisfies stakeholder interests. It contains use case steps and a postcondition (Lines 3-7, 23-26 and 39-43). A step can be one of the following interactions: an actor sends a request or data to the system (Line 34); the system validates a request or data (Line 4); the system replies to an actor with a result (Line 7). The system can alter its internal state (Line 18). The inclusion of an-

other use case is given in a step with the keyword '*INCLUDE USE CASE*' (Line 3). The keywords are written in capital letters. '*VALIDATES THAT*' (Line 4) indicates a condition that must be true to take the next step, otherwise an alternative flow is taken.

An alternative flow describes other scenarios, both success and failure. It always depends on a condition in a specific step of the basic flow. RUCM has *specific*, *bounded* and *global* alternative flows. A specific alternative flow refers to a step in the basic flow (Lines 13, 16, and 27). A bounded alternative flow refers to more than one step in the basic flow (Line 8), while a global one refers to any step in the basic flow. '*RFS*' is used to refer to reference flow steps (Lines 9, 14, 17, and 28). Bounded and global alternative flows begin with '*IF .. THEN*' for the conditions under which they are taken (Line 10). Specific alternative flows do not necessarily begin with '*IF .. THEN*' since a guard condition is already indicated in their reference flow steps (Line 4).

Our extensions are (i) new keywords for modeling interactions in embedded systems and (ii) new keywords for modeling variability. The keywords '*SENDS .. TO*' and '*REQUESTS .. FROM*' are to distinguish system-actor interactions (Lines 5, 7, 34, and 39-43). We introduce the notion of variation point and variant, complementary to the extensions in Section 3.1, into RUCM. Variation points can be included in basic or alternative flows with the keyword '*INCLUDE <VARIATION POINT : ... >*' (Lines 29 and 35). Variant use cases are given with the keyword '*<VARIANT >*' (Line 37).

Some variability cannot be captured in PL use case diagrams due to the required level of granularity for product configuration. To model such variability, as part of our extensions, we introduce optional steps, optional alternative flows and a variant order of steps. Optional steps and alternative flows begin with '*<OPTIONAL>*' (Lines 8 and 39-43). We use 'V' before any step number to express variant step orders (Lines 39-43).

3.2 Configuration of PS Use Case Models

PUMConf relies on variability information given in the PL use case diagram and specifications. The user selects (1) variant use cases in the PL diagram and (2) use case elements in the PL specifications, to generate the PS models.

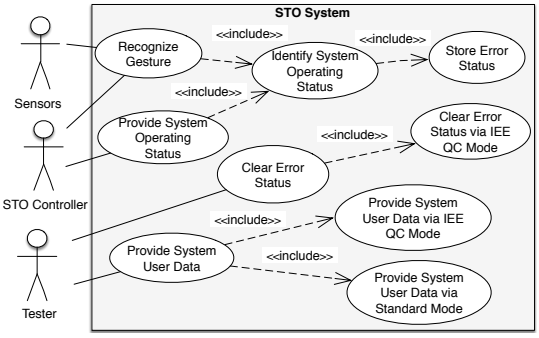

Fig. 2. Generated PS Use Case Diagram

The user makes decisions for the variation points in Fig. 1. PUMConf automatically generates the PS use case diagram from the PL diagram and the diagram decisions (see Fig. 2 generated from Fig. 1). For instance, based on the decision for *Method of Providing Data* in Fig. 1, PUMConf creates *Provide System User Data via IEE QC Mode*, *Provide System User Data via Standard Mode* and two *include* relations in Fig. 2.

In Table 1, there are two variation points (Lines 29 and 35), one variant use case (Lines 37-43), five optional steps (Lines 39-43), one optional alternative flow (Lines

8-12), and one variant order group (Lines 39-43). The user selects only three optional steps with the order *V3*, *V1*, and *V5*. The optional alternative flow is not selected.

The PS specifications are automatically generated from the PL specifications and the diagram and specification decisions. (see Table 2 generated from Table 1). For instance, based on the diagram decision for *Method of Providing Data* in Fig. 1, PUMConf creates two include statements for *Provide System User Data via Standard Mode* and *via IEE QC Mode* (Lines 31 and 34 in Table 2), a validation step (Line 30), and a specific alternative flow where *Provide System User Data via IEE QC Mode* is included (Lines 32-35). The validation step checks if the precondition of *Provide System User Data via Standard Mode* holds. If it holds, *Provide System User Data via Standard Mode* is executed in the basic flow (Line 31). If not, the alternative flow is taken to execute *Provide System User Data via IEE QC Mode* (Lines 32-35). Selected optional steps and alternative flows are included in the PS specifications, while variant order groups are ordered (Lines 39-41).

Table 2. Some of the Generated PS Specifications

1	**USE CASE** Recognize Gesture
2	**1.1 Basic Flow**
3	1. INCLUDE USE CASE Identify System Operating Status.
4	2. The system VALIDATES THAT the operating status is valid.
5	3. The system REQUESTS the move capacitance FROM the sensors.
6	4. The system VALIDATES THAT the movement is a valid kick.
7	5. The system SENDS the valid kick status TO the STO Controller.
8	**1.2 Specific Alternative Flow**
9	RFS 2
10	1. ABORT.
11	**1.3 Specific Alternative Flow**
12	RFS 4
13	1. The system increments the OveruseCounter by the increment step.
14	2. ABORT.
15	
16	**USE CASE** Identify System Operating Status
17	**1.1 Basic Flow**
18	1. The system VALIDATES THAT the watchdog reset is valid.
19	2. The system VALIDATES THAT the RAM is valid.
20	3. The system VALIDATES THAT the sensors are valid.
21	4. The system VALIDATES THAT there is no error detected.
22	**1.4 Specific Alternative Flow**
23	RFS 4
24	1. INCLUDE USE CASE Store Error Status.
25	2. ABORT.
26	
27	**USE CASE** Provide System User Data
28	**1.1 Basic Flow**
29	1. The tester SENDS the user data request TO the system.
30	2. The system VALIDATES THAT 'Precondition of Provide System User Data via Standard Mode'.
31	3. INCLUDE Provide System User Data via Standard Mode.
32	**1.2 Specific Alternative Flow**
33	RFS 2
34	1. INCLUDE Provide System User Data via IEE QC Mode.
35	2. ABORT.
36	
37	**USE CASE** Provide System User Data via Standard Mode
38	**1.1 Basic Flow**
39	1. The system SENDS the trace data TO the tester.
40	2. The system SENDS the calibration data TO the tester.
41	3. The system SENDS the error trace data TO the tester.

4 Overview of the Approach

The reconfiguration of PS models is implemented as a pipeline (Fig. 3). Configuration decisions are captured in a decision model during the decision-making process. The decision model conforms to a decision metamodel, described in our prior work [11]. PUMConf keeps two decision models, i.e., the decision model before changes (*M1* in Fig. 3) and the decision model after changes (*M2* in Fig. 3). Fig. 4 provides the decision metamodel and the two input decision models for the PL models in Fig. 1 and Table 1.

The pipeline takes the decision models, and the PS diagram and specifications as input. The PS models are reconfigured, as output, together with an impact report, i.e., list of reconfigured parts of the PS models. The pipeline has three steps given in Fig. 3.

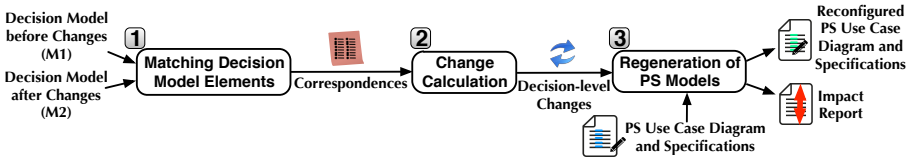

Fig. 3. Overview of the Model Differencing and Regeneration Pipeline

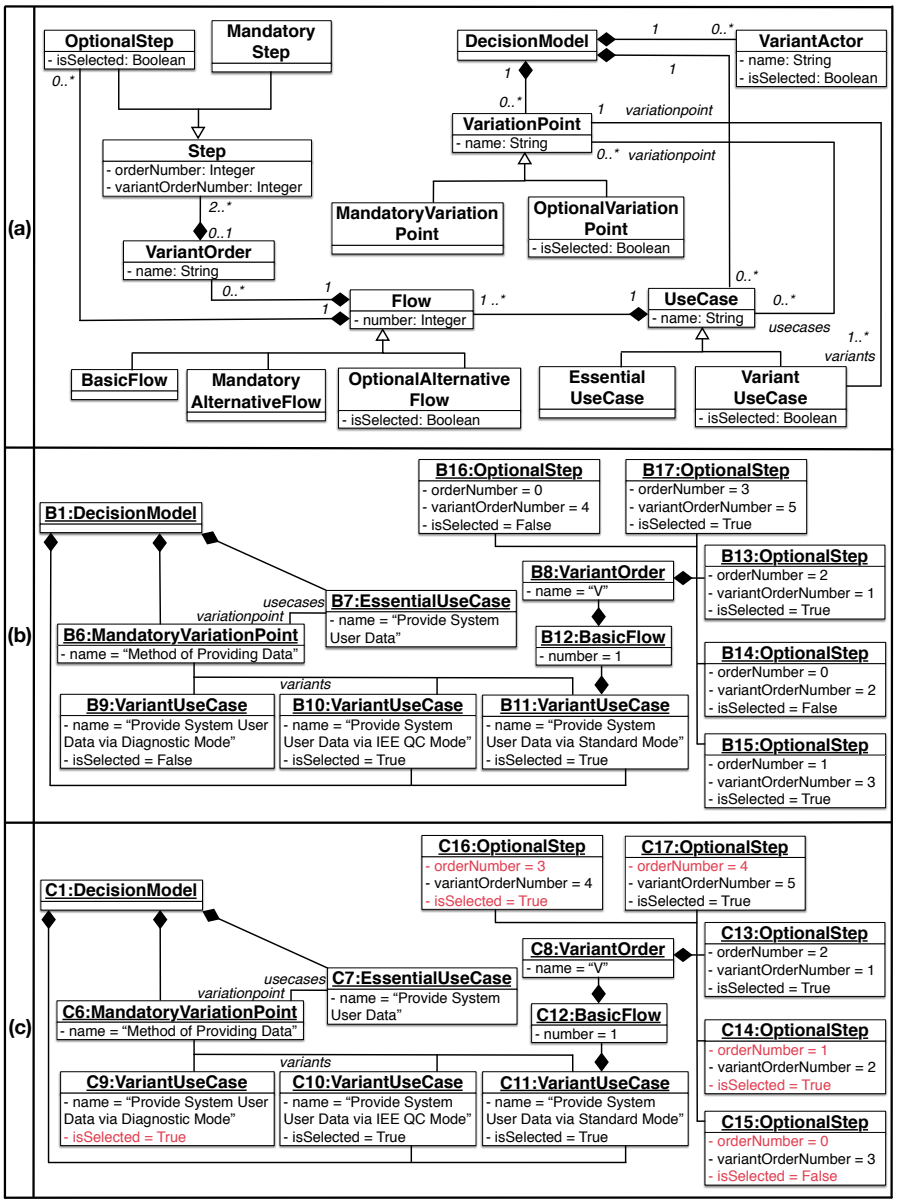

Fig. 4. (a) Decision Metamodel, (b) Example *M1*, and (c) Example *M2*

In Step 1, *Matching decision model elements*, the structural differencing of *M1* and *M2* is done by looking for the correspondences in *M1* and *M2*. To that end, we devise an algorithm that identifies the matching model elements in *M1* and *M2*. The output of Step 1 is the corresponding elements, representing decisions for the same variations, in *M1* and *M2* (Section 5).

The decision metamodel in Fig. 4(a) includes the main use case elements for which the user makes decisions (i.e., variation point, optional step, optional alternative flow, and variant order). In a variation point, the user selects variant use cases to be included for the product. For PL use case specifications, the user selects optional steps and alternative flows to be included and determines the order of steps (variant order). Therefore, the matching elements in Step 1 are the pairs of variation points and use cases including the variation points, the pairs of use cases and optional alternative flows in the use cases, and the triples of use cases, flows in the use cases, and optional steps in the flows.

In Step 2, *Change calculation*, decision-level changes are identified from the corresponding model elements (see Section 5). A set of elements in *M1* which does not have a corresponding set of elements in *M2* is considered to be a deleted decision, which we refer to as *DeleteDecision* in the decision-level changes. Analogously, a set of model elements in *M2* which does not have a corresponding set of elements in *M1* is considered to be added (*AddDecision*). Each set of corresponding model elements with non-identical attribute values (see the red-colored attributes in Fig. 4(c)) is considered to be a decision-level change of the type *UpdateDecision*. Alternatively, we could record changes during the decision-making process. However, the user might make changes cancelling previous changes or implying some further changes. In such a case, we would have to compute cancelled changes and infer new changes.

In Step 3, *Regeneration of PS models*, the PS use case diagram and specifications are regenerated only for the added, deleted and updated decisions (see Section 6). For instance, use cases selected in the deleted decisions are removed from the PS models, while use cases selected in the added decisions are added in the PS models.

5 Model Matching and Change Calculation

We devise an algorithm (see Fig. 5) for the first two pipeline steps, *Matching Decision Model Elements* and *Change Calculation*, in Fig. 3. The algorithm calls some *match* functions (Lines 7-9 in Fig. 5) to identify the corresponding model elements, which represent decisions for the same variations, in the input decision models. The *match* functions implement Step 1 in Fig. 3.

- ***matchDiagramDecisions*** returns the set of pairs (*variation point*, *use case*) matching in the decision models (*M1* and *M2*), which are capturing which variation points are included in the use cases involved in diagram decisions,
- ***matchFlowDecisions*** returns the set of pairs (*use case*, *optional alternative flow*) matching in the input decision models (*M1* and *M2*), which are capturing which optional alternative flows are in the use cases involved in flow decisions,
- ***matchStepDecisions*** returns the set of triples (*use case*, *flow*, *step*) matching in the input decision models (*M1* and *M2*), which are capturing which steps are in the flows of the use cases involved in step decisions.

The corresponding model elements in the example decision models in Fig. 4(b) and (c) are as follows (Lines 7-9 in Fig. 5):

- For decisions in the variation points, $U3 = \{(B6, B7), (C6, C7)\}$,
- For decisions in the optional alternative flows, $F3 = \{\emptyset\}$,
- For decisions in the use case steps, $S3 = \{(B11, B12, B13), (B11, B12, B14), (B11, B12, B15), (B11, B12, B16), (B11, B12, B17), (C11, C12, C13), (C11, C12, C14), (C11, C12, C15), (C11, C12, C16), (C11, C12, C17)\}$.

A variant use case in a variation point (vp) may include another variation point (vp'). Changing the decision for vp may imply another decision to be added or deleted for vp'. As part of Step 2, *Change Calculation*, the algorithm first identifies deleted and added diagram decisions by checking the pairs of variation points and use cases which exist only in one of the input decision models (($U1 \setminus U3$) and ($U2 \setminus U3$) in Lines 10-11). Similar checks are done for flow and step decisions in the specifications (Lines 10-11). For the decision models in Fig. 4, there is no deleted or added decision (($U1 \setminus U3 = \emptyset$), ($U2 \setminus U3 = \emptyset$), ($F1 \setminus F3 = \emptyset$), ($F2 \setminus F3 = \emptyset$), ($S1 \setminus S3 = \emptyset$), and ($S2 \setminus S3 = \emptyset$)).

Input: Initial decision model $M1$, New decision model $M2$
Output: Triple of sets of decision-level changes (ADD, DELETE, UPDATE)

1. Let a pair (vp, uc) denote cases where vp is a variation point and uc is a use case including vp
2. Let a pair (uc, fl) denote cases where uc is a use case and fl is an optional alternative flow in uc
3. Let a triple (uc, fl, st) denote cases where uc is a use case, fl is a flow in uc, and st is a step in fl
4. Let *U1* and *U2* be the sets of (vp, uc) in *M1* and *M2*
5. Let *F1* and *F2* be the sets of (uc, fl) in *M1* and *M2*
6. Let *S1* and *S2* be the sets of (uc, fl, st) in *M1* and *M2*
7. *U3* ← **matchDiagramDecisions**(*U1*, *U2*)
8. *F3* ← **matchFlowDecisions**(*F1*, *F2*)
9. *S3* ← **matchStepDecisions**(*S1*, *S2*)
10. *DELETE* ← $(U1 \setminus U3) \cup (F1 \setminus F3) \cup (S1 \setminus S3)$
11. *ADD* ← $(U2 \setminus U3) \cup (F2 \setminus F3) \cup (S2 \setminus S3)$
12. **foreach** $(k \in (U3 \cap U1))$ **do**
13. z ← **getMatchingDecision**(k, *U3*)
14. *SUC1* ← **getSelectedUseCases**(k, *M1*)
15. *SUC2* ← **getSelectedUseCases**(z, *M2*)
16. **if** $(SUC1 \neq SUC2)$ **then**
17. $UPDATE \leftarrow UPDATE \cup \{k\}$;
18. **end if**
19. **end foreach**
20. **foreach** $(t \in (F3 \cap F1))$ **do**
21. y ← **getMatchingDecision**(t, *F3*)
22. **if** $(t.fl.isSelected \neq y.fl.isSelected)$ **then**
23. $UPDATE \leftarrow UPDATE \cup \{t\}$
24. **end if**
25. **end foreach**
26. **foreach** $(u \in (S3 \cap S1))$ **do**
27. m ← **getMatchingDecision**(u, *S3*)
28. **if** $(u.st$ **is** $OptionalStep)$ **and** $(u.st.isSelected \neq m.st.isSelected)$ **then**
29. $UPDATE \leftarrow UPDATE \cup \{u\}$
30. **else**
31. **if** $(u.st.orderNumber \neq m.st.orderNumber)$
32. **then** $UPDATE \leftarrow UPDATE \cup \{u\}$
33. **end if**
34. **end if**
35. **end foreach**
36. **return** (*ADD*, *DELETE*, *UPDATE*)

Fig. 5. Algorithm for Steps 1 and 2 in Fig. 3

The matching pairs of variation points and their including use cases represent decisions for the same variation point ($(B6, B7)$ and $(C6, C7)$ in Fig. 4(b) and (c)). If the selected variant use cases for the same variation point are not the same in *M1* and *M2*, the corresponding decision in *M1* is considered as updated in *M2* (Lines 12-19). The variant use case *Provide System User Data via Diagnostic Mode* of the variation point *Method of Providing Data* is unselected in *M1* (*B6*, *B7* and *B9* in Fig. 4(b)), but selected in *M2* (*C6*, *C7* and *C9* in Fig. 4(c)). The diagram decision for the pair (*B6*, *B7*) in *M1* is identified as updated (Line 17). To identify updated specification decisions, the algorithm compares decisions across *M1* and *M2* that involve optional alternative flows, optional steps and

steps with a variant order (Lines 22-24, 28-30 and 31-33). In our example, the triples $(B11, B12, B14), (B11, B12, B15)$, $(B11, B12, B16)$, and $(B11, B12, B17)$ in Fig. 4 are identified as updated decisions.

6 Regeneration of PS Use Case Models

After all the changes are calculated by matching the corresponding model elements in the input decision models, the parts of PS use case models affected by the changed decisions are automatically regenerated (Step 3 in Fig. 3).

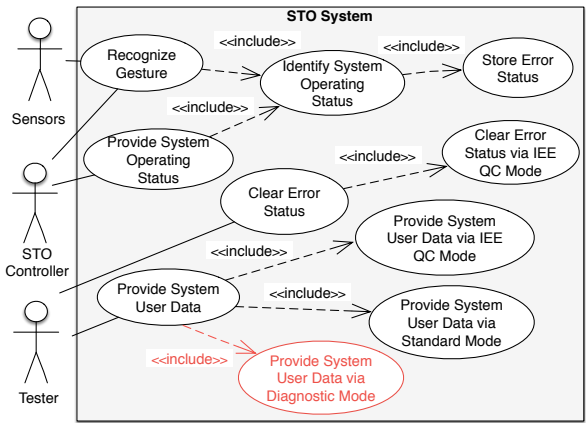

Fig. 6. Regenerated PS Use Case Diagram

Our approach first handles the diagram decision changes to reconfigure the PS use case diagram. For selected variant use cases in the added diagram decisions (i.e., in the pairs (*vp*, *uc*) in *ADD* in Line 36 in Fig. 5), we generate the corresponding use cases and *include* relations in the PS diagram. For selected variant use cases in deleted diagram decisions (i.e., in the pairs (*vp*, *uc*) in *DELETE* in Line 36), we remove the corresponding use cases and *include* relations from the PS diagram. If a selected variant use case is unselected in an updated diagram decision (i.e., in the pairs (*vp*, *uc*) in *UPDATE* in Line 36), we remove the corresponding use case from the PS diagram. For unselected variant use cases which are selected in the updated diagram decisions, the corresponding use cases and *include* relations are added to the PS diagram. Fig. 6 gives the regenerated parts of the PS use case diagram in Fig. 2 for *M1* and *M2* in Fig. 4.

There is no added or deleted diagram decision in *M1* and *M2* in Fig. 4. The decision for the variation point *Method of Providing Data* (i.e., $(B6, B7)$ in *UPDATE* in Line 36) is updated by selecting the variant use case *Provide System User Data via Diagnostic Mode*. Only the corresponding use case and its *include* relation are added to the PS diagram (red-colored in Fig. 6).

Changes for diagram and specification decisions are used to regenerate the PS specifications. For diagram decision changes, we add or delete the corresponding use case specifications. Table 3 provides the regenerated parts of the PS specifications in Table 2, for *M1* and *M2* in Fig. 4.

For the variation point *Method of Providing Data* included by the use case *Provide System User Data* (i.e., $(B6, B7)$), we have one updated diagram decision in which the unselected use case *Provide System User Data via Diagnostic Mode* is selected. The corresponding use case specification is added (Lines 24-29 in Table 3). A new specific alternative flow is also generated for the inclusion of the newly selected use case in the specification of the use case *Provide System User Data* (Lines 12-15, red-colored).

Table 3. Regenerated PS Use Case Specifications

1	**USE CASE** Provide System User Data
2	**1.1 Basic Flow**
3	1. The tester SENDS the user data request TO the system.
4	2. The system VALIDATES THAT 'Precondition of Provide System User Data via Standard Mode'.
5	3. INCLUDE Provide System User Data via Standard Mode.
6	**1.2 Specific Alternative Flow**
7	RFS 2
8	1. IF 'Precondition of Provide System User Data via IEE QC Mode' holds THEN
9	2. INCLUDE Provide System User Data via IEE QC Mode.
10	3. ABORT.
11	4. ENDIF
12	**1.3 Specific Alternative Flow**
13	RFS 2
14	1. INCLUDE Provide System User Data via Diagnostic Mode.
15	2. ABORT.
16	
17	**USE CASE** Provide System User Data via Standard Mode
18	**1.1 Basic Flow**
	~~1. The system SENDS trace data TO the tester.~~
19	1. The system SENDS sensor data TO the tester.
20	2. The system SENDS calibration TO the tester.
21	3. The system SENDS error data TO the tester.
22	4. The system SENDS error trace data TO the tester.
23	
24	**USE CASE** Provide System User Data via Diagnostic Mode
25	**1.1 Basic Flow**
26	1. The system SENDS the RAM data TO the tester.
27	2. The system SENDS the NVM data TO the tester.
28	3. The system SENDS the session response TO the tester.
29	4. The system SENDS the message length TO the tester.

The specification decision changes are about selecting optional alternative flows, optional steps and steps with a variant order (e.g., the triples $(B11, B12, B14)$, $(B11, B12, B15)$, $(B11, B12, B16)$, and $(B11, B12, B17)$ in Fig. 4(b)). The use case *Provide System User Data via Standard Mode* has two new steps in Lines 19 and 21 in Table 3 (i.e., $(B11, B12, B14)$, and $(B11, B12, B16)$ in Fig. 4(b)), while one of the steps (red-colored, strikethrough step) is removed (i.e., $(B11, B12, B15)$ in Fig. 4(b)). The step number of one of the steps is changed (Line 22, blue-colored) due to the change in the order of the steps with a variant order (i.e., $(B11, B12, B17)$ in Fig. 4(b)).

7 Tool Support

We implemented our approach as an extension of PUMConf [24] which has been developed as an IBM DOORS Plug-in. PUMConf uses GATE (`http://gate.ac.uk/`), an open source NLP framework, to annotate PL use case specifications to be used for (re)configuring PS use case specifications. PUMConf relies upon: (i) *IBM DOORS* to model PL use case specifications and (ii) *Papyrus* to model and save PL use case diagrams as a UML file. To load use cases from IBM DOORS, it uses DOORS Document Exporter, an API that exports the DOORS content as text files. The reconfiguration of PS use case models has been implemented as a Java application. The DOORS eXtension Language (DXL) is employed to load the configured PS specifications into DOORS. PUMConf is approximately 25K lines of code, excluding comments and third-party libraries. Additional details about PUMConf, including executable files and a screencast covering motivations, are available on the tool's website at `https://sites.google.com/site/pumconf/`.

8 Industrial Case Study

We evaluate our reconfiguration approach via reporting an industrial case study (STO).

Goal: Our goal was to assess, in an industrial context, the feasibility of using our approach. We assessed whether we could improve reuse and reduce manual effort by preserving unimpacted parts of PS use case models, when possible, and their manually assigned traces.

Study Context: STO was selected for the assessment of our approach since it was a relatively new project at IEE with multiple potential customers requiring different features. IEE provided their initial STO documentation, which contained a use case diagram, use case specifications, and supplementary requirements specifications describing non-functional requirements. To model the STO requirements according to our modeling method, PUM, we first examined the initial STO documentation and then worked with IEE engineers to build and iteratively refine our models [11] (see Table 4). Due to the confidentiality concerns, we do not put the entire case study online. However, the reader can download the sanitized example models from the tool's website.

Table 4. Product Line Use Cases in the Case Study

	# of use cases	**# of variation points**	**# of basic flows**	**# of alternative flows**	**# of steps**	**# of condition steps**
Essential Use Cases	11	6	11	57	192	57
Variant Use Cases	13	1	13	131	417	130

Table 5. Configuration Results for the Selected Product

Product	**# of Selected Variant Use Cases**	**# of Selected Optional Steps**	**# of Selected Optional Flows**	**# of Decided Variant Order**
P1	6	1	0	0

Table 6. Decision Change Scenarios

ID	**Change Scenario**	**Explanation**
S1	Update a diagram decision	Unselecting selected use cases
S2	Update and delete diagram decisions	Unselecting selected use cases, removing other decisions
S3	Update a diagram decision	Selecting unselected use cases
S4	Update and add diagram decisions	Selecting unselected use cases, implying other decisions
S5	Update a specification decision	Selecting unselected optional steps
S6	Update a diagram decision	Selecting unselected use cases
S7	Update a diagram decision	Unselecting selected use cases
S8	Update a specification decision	Updating the order of optional steps

Results and Analysis: By using PUMConf, we, together with the IEE analysts, configured the PS use case models for four products selected among the STO products IEE had already developed [15]. The IEE analysts made the decisions on the PL models using the guidance provided by PUMConf. Among the four products, we chose one product to be used for reconfiguration of PS models (see Table 5) because it was the most recent one in the STO product family with a properly documented change history. The IEE engineers identified 36 traces from the PS use case diagram and 278 traces

from the PS use case specifications to other software and hardware specifications as well as to the customers' requirements documents for external systems (see Fig. 7). We considered eight change scenarios derived from the change history of the initial STO documentation for the selected product (see Table 6).

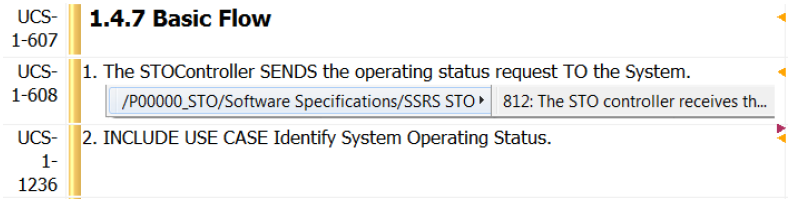

Fig. 7. An Example Specification with a Trace in DOORS

Some change scenarios contain individual decision changes such as selecting unselected use cases in a variation point, while some others contain a series of individual changes applied sequentially (see *S2* and *S4*). For instance, *S2* starts with unselecting *Clear Error Status* in Fig. 1, which automatically deletes the decision for the variation point *Method of Clearing Error Status* and implies another decision change, i.e., unselecting *Store Error Status*.

Table 7. Summary of the Reconfiguration of the PS Use Case Models for STO

		Decision Change Scenarios							
		S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**
PS Model Changes	**# of Added UCs**	0	0	1	4	0	1	0	0
	# of Deleted UCs	1	4	0	0	0	0	1	0
	# of Added UC Steps	0	0	53	140	3	85	0	0
	# of Deleted UC Steps	53	140	0	0	0	0	103	0
Traces for the PS Use Case Diagram	**# of Initial Traces**	36	34	25	27	36	36	38	38
	# of Deleted Traces During Reconfiguration	2	9	0	0	0	0	0	0
	# of Manually Added Traces After Reconfiguration	0	0	2	9	0	2	0	0
	# of Preserved Traces	34	25	25	27	36	36	38	38
	% of Preserved Traces	94.4	73.5	100	100	100	100	100	100
Traces for the PS Use Case Specifications	**# of Initial Traces**	278	265	218	231	278	287	298	278
	# of Deleted Traces During Reconfiguration	13	47	0	0	0	0	20	0
	# of Manually Added Traces After Reconfiguration	0	0	13	47	9	11	0	0
	# of Preserved Traces	265	218	218	231	278	287	278	278
	% of Preserved Traces	95.3	82.2	100	100	100	100	93.2	100

Table 7 provides a summary of the reconfiguration of the PS models for the change scenarios. After each change scenario, we ran PUMConf and checked the preserved and deleted traces. Our approach preserved all the traces for the unchanged parts of the PS models, while only the traces for the deleted parts of the PS models were removed. We had to manually assign traces only for the new parts of the PS models. In terms of saving traceability effort while reconfiguring, we can look at the percentages of traces from the use case diagram and the use case specifications that were preserved over all the change scenarios. From Table 7, we can see that between 73% and 100% (average $\approx$ 96%) of the use case diagram traces were preserved. Similarly, for the use

case specifications, trace reuse was between 82% and 100% (average $\approx$ 96%). We can therefore conclude that our automated approach to incremental reconfiguration leads to significant reuse and savings when updating traceability to other documents to account for changed configuration decisions.

Discussion: We also had semi-structured interviews with IEE engineers to better assess their perception. All interview participants agreed that the proposed approach could help reduce, by a substantial amount, the manual effort in terms of preserving traces for the unchanged parts of the PS use case models.

Threats to Validity: The main threat to the validity of our case study regards the generalizability of our conclusions. To mitigate this threat, we applied our approach to an industrial case study that includes nontrivial use cases in an application domain with many potential customers and numerous sources of variability. To limit threats to internal validity, we had many interviews with IEE engineers in the STO project to verify the correctness and completeness of the PL models and the reconfigured PS models.

9 Conclusion

Product line requirements need to be configured for each product. This paper presents an incremental reconfiguration approach for use case models in the context of product lines. Our main motivation is to preserve the unimpacted parts of the Product Specific (PS) use case models, when changing their configuration decisions, based on a careful analysis of the Product Line (PL) use case models. Our main goal is to avoid manual effort during reconfiguration due to the manual updating of traceability links from the PS use case models to other documents and artifacts, a common practice and requirement in industry. We therefore need to carefully determine which parts of the PS models remain unchanged and we do so by carefully analysis decision dependencies in PL models. We aim to incrementally reconfigure PS use case models by minimizing their changes based on a careful impact analysis of changed decisions. We performed a case study in the context of automotive embedded system development. The results suggest that our approach is practical and provides significant savings with respect to traceability updates during reconfiguration.

This work is an intermediate step to achieve our long term objective [25], i.e., change impact analysis and regression test selection in the context of use case-driven development and testing. Changes can also emerge in variability aspects of product line models, and they entail impact assessment on decisions for each individual product and may require reconfiguration and regression test selection in several products. Our plan for the next steps is to support change impact analysis to help analysts properly manage changes in PL use case models.

Acknowledgments. Financial support was provided by IEE and FNR under grants FNR/P10/03 and FNR10045046.

References

1. C. Nebut, F. Fleurey, Y. L. Traon, and J.-M. Jezequel, "Automatic test generation: A use case driven approach," *IEEE TSE*, vol. 32, no. 3, pp. 140–155, 2006.

2. C. Wang, F. Pastore, A. Goknil, L. C. Briand, and M. Z. Z. Iqbal, "Automatic generation of system test cases from use case specifications," in *ISSTA'15*, 2015, pp. 385–396.
3. ——, "UMTG: a toolset to automatically generate system test cases from use case specifications," in *ESEC/SIGSOFT FSE'15*, 2015, pp. 942–945.
4. "IEE (International Electronics & Engineering) S.A., http://www.iee.lu/."
5. P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001.
6. M. Eriksson, J. Borstler, and K. Borg, "The pluss approach - domain modeling with features, use cases and use case realizations," in *SPLC'05*, 2005, pp. 33–44.
7. A. Fantechi, S. Gnesi, G. Lami, and E. Nesti, "A methodology for the derivation and verification of use cases for product lines," in *SPLC'04*, 2004, pp. 255–265.
8. K. Czarnecki and M. Antkiewicz, "Mapping features to models: A template approach based on superimposed variants," in *GPCE'05*, 2005, pp. 422–437.
9. S. Sepulveda, A. Cravero, and C. Cachero, "Requirements modeling languages for software product lines: A systematic literature review," *IST*, vol. 69, pp. 16–36, 2016.
10. B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE TSE*, vol. 27, no. 1, pp. 58–93, 2001.
11. I. Hajri, A. Goknil, L. C. Briand, and T. Stephany, "Applying product line use case modeling in an industrial automotive embedded system: Lessons learned and a refined approach," in *MODELS'15*, 2015, pp. 338–347.
12. G. Halmans and K. Pohl, "Communicating the variability of a software-product family to customers," *SoSyM*, vol. 2, pp. 15–36, 2003.
13. S. Buhne, G. Halmans, and K. Pohl, "Modeling dependencies between variation points in use case diagrams," in *REFSQ'03*, 2003, pp. 59–69.
14. T. Yue, L. C. Briand, and Y. Labiche, "Facilitating the transition from use case models to analysis models: Approach and experiments," *TOSEM*, vol. 22, no. 1, 2013.
15. I. Hajri, A. Goknil, L. C. Briand, and T. Stephany, "Configuring use case models in product families," *SoSyM*, 2016.
16. D. Dhungana, P. Grünbacher, and R. Rabiser, "The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study," *ASE*, vol. 18, pp. 77–114, 2011.
17. W. Heider, R. Rabiser, and P. Grünbacher, "Facilitating the evolution of products in product line engineering by capturing and replaying configuration decisions," *STTT*, vol. 5, pp. 613–630, 2012.
18. W. Heider, R. Rabiser, D. Lettner, and P. Grünbacher, "Using regression testing to analyze the impact of changes to variability models on products," in *SPLC'12*, 2012, pp. 196–205.
19. D. Hearnden, M. Lawley, and K. Raymond, "Incremental model transformation for the evolution of model-driven systems," in *MODELS'06*, 2006, pp. 321–335.
20. I. Kurtev, M. Dee, A. Göknil, and K. van den Berg, "Traceability-based change management in operational mappings," in *ECMDA-TW'07*, 2007, pp. 57–67.
21. S. Jahann and A. Egyed, "Instant and incremental transformation of models," in *ASE'04*, 2004, pp. 362–365.
22. T. Vogel, S. Neumann, S. Hildebrandt, H. Giese, and B. Becker, "Incremental model synchronization for efficient run-time monitoring," in *MODELS Workshops*, 2009, pp. 124–139.
23. R. Eramo, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio, "A model-driven approach to automate the propagation of changes among architecture description languages," *SoSyM*, vol. 11, pp. 29–53, 2012.
24. I. Hajri, A. Goknil, L. C. Briand, and T. Stephany, "PUMConf: A tool to configure product specific use case and domain models in a product line," in *FSE'16*, 2016, pp. 1008–1012.
25. I. Hajri, "Supporting change in product lines within the context of use case-driven development and testing," in *Doctoral Symposium - FSE'16*, 2016, pp. 1082–1084.