PUMConf: A Tool to Configure Product Specific Use Case and Domain Models in a Product Line

Ines Hajri, Arda Goknil, Lionel C. Briand
SnT Centre for Security, Reliability and Trust
University of Luxembourg, Luxembourg
{firstname.lastname}@uni.lu

Thierry Stephany
International Electronics & Engineering (IEE)
Contern, Luxembourg
{firstname.lastname}@iee.lu

ABSTRACT

We present PUMConf, a tool for supporting configuration that currently focuses on requirements and enables effective product line management in the context of use case-driven development. By design, it relies exclusively on variability modeling for artifacts that are commonly used in such contexts (i.e., use case diagram, specifications and domain model). For given Product Line (PL) use case and domain models, PUMConf checks the consistency of the models, interactively receives configuration decisions from analysts, automatically checks decision consistency, and generates Product Specific (PS) use case and domain models from the PL models and decisions. It has been evaluated on an industrial case study in the automotive domain.

CCS Concepts

•Software and its engineering → Software product lines;

Keywords

Product Line Engineering; Use Case-Driven Development.

1. INTRODUCTION

Product Line Engineering (PLE) is being widely adopted in industry due to the increasing complexity of software systems that warrant better support for reusable software artifacts. In various business contexts, use cases are central development artifacts and used for communicating requirements among stakeholders and for system test case generation [17] [18]. In environments where software development practice is strongly use case-driven, use case configurators can play a key role to capture variable requirements in Product Line (PL) use cases and to generate Product Specific (PS) use cases for each new customer in a product family.

We present a tool, PUMConf (Product line Use case Model Configurator), to support automated requirements configuration that guides stakeholders in making configuration decisions and automatically generates use case and domain models for the configured product. PUMConf is developed for use case-driven development environments within the context of our research in collaboration with IEE S.A. [1], a leading supplier of embedded software systems in the automotive domain. The motivation behind PUMConf is to provide a high degree of automation during configuration and to avoid unnecessary modeling overhead and complexity by relying exclusively on variability modeling for commonly used artifacts in use-case driven development, i.e., use case diagrams, use case specifications and domain models.

PUMConf builds on our previous work [9] where we proposed a use case-centric product line modeling method (PUM). Using PUM, variability is directly captured in the PL use case diagram, specifications and domain model without any feature model, at a level of granularity enabling precise communication with various stakeholders. PUMConf provides the following features: (i) the automated consistency checking of PL use case and domain models, (ii) the automated, interactive configuration support including consistency checking of configuration decisions, and (iii) the automated generation of PS use case and domain models from PL models and configuration decisions. Natural Language Processing (NLP) is employed to check the consistency of PL models. Our tool automatically infers new configuration decisions based on variation point-variant dependencies and prior decisions input by the analyst. The consistency checking of configuration decisions, i.e., determining contradicting decisions made for the variation points in the PL use case diagram, is based on mapping variation points, use cases and variant dependencies into propositional logic formulas. We developed our own consistency checking algorithm using these logic formulas. PUMConf automatically generates PS use case and domain models using a set of transformation rules implemented in Java. Our tool is integrated with an industrial requirements management tool: IBM DOORS.

In the remainder of this paper, we outline PUMConf's features and main components. We further highlight the findings from our evaluation of PUMConf over an industrial case study and a questionnaire study with IEE engineers.

2. RELATED WORK

Several configuration tools for scenario-based requirements have been proposed in the literature [2, 3, 4, 8, 6, 7, 16]. Most of these require that feature models be manually traced to requirements by analysts. This entails additional modeling and maintenance effort. Moon et al. [14, 13] present a configurator generating PS use cases without using any

feature model. However, the configurator relies on various matrices manually formed for use cases and primitive requirements, i.e., building blocks of complex requirements.

There are various generic configurators, e.g., DOPLER [5], C2O configurator [15], and SPLOT [12], developed for configuring variability models. Though these configurators could be employed, they require considerable effort and tool-specific knowledge to be customized for configuring use cases.

PUMConf does not require analysts to trace feature models to use case models. It relies exclusively on variability modeling within commonly used artifacts in use-case driven development. Furthermore, our tool checks decision consistency and orders decisions to facilitate the decision-making process, thus providing a high degree of automation.

3. TOOL OVERVIEW

PUMConf is the tool support of our configuration approach for use case-driven development, described in a recent research paper [10]. Figure 1 presents an overview of our tool. In Step 1, the analyst elicits the PL use case and domain models using (1) PL extensions for use case diagrams [11], (2) a structured form of use case specifications, i.e., Restricted Use Case Modeling (RUCM) [19], and its PL extensions, and (3) PL extensions for domain models [20]. The elicitation of PL models is performed according to our Product line Use case modeling Method (PUM) [9].

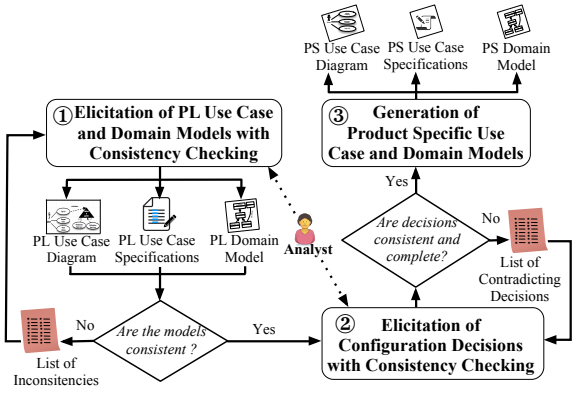

Figure 1: Tool Overview

Once the analyst captures variability in PL models, PUMConf automatically checks the consistency of the PL use case diagram, the PL use case specifications and the PL domain model. If any inconsistency is detected, e.g., a variation point specified in the diagram is missing in the specifications, the tool reports these inconsistencies. Another consistency checking task concerns the PL use case specifications, which should conform to the RUCM template.

In Step 2, PUMConf processes the PL models once they are deemed consistent. It interactively gets configuration decisions from the analyst and detects contradicting decisions made for variation points in the PL use case diagram.

Once all the configuration decisions are made, the tool proceeds to Step 3 with the generation of PS use case and domain models from the PL models. The generation is based on decisions and a set of transformation rules. In the rest of this section, we elaborate on each step in Figure 1.

3.1 Elicitation of PL Use Case Models

As a first step, the analyst manually elicits PL use case and domain models. To model variation points, variant use cases as well as their constraints and dependencies in the use case diagram, PUMConf employs the PL extensions proposed by Halmans and Pohl [11]. Figure 2 shows a part of the PL use case diagram for Smart Trunk Opener (STO), a real-time automotive embedded system developed by IEE.

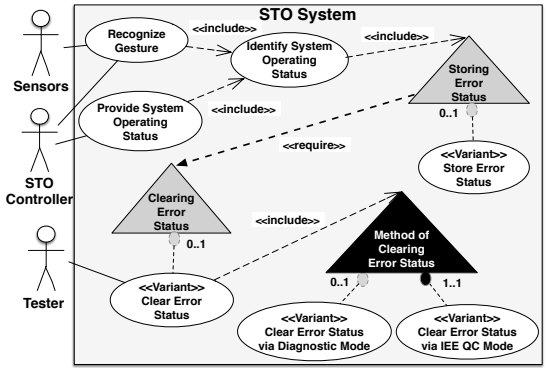

Figure 2: Part of the PL Use Case Diagram for STO

In Figure 2, there are three variation points (i.e., *Storing Error Status, Clearing Error Status and Method of Clearing Error Status*), four variant use cases, one *require* dependency between two variation points, and four *cardinality constraints* restricting the selection of variant use cases.

Table 1: A PL Use Case Specification for STO

1	**USE CASE** Identify System Operating Status
2	**1.1 Basic Flow**
3	1. The system VALIDATES THAT the watchdog reset is valid.
4	2. The system VALIDATES THAT the RAM is valid.
5	3. The system VALIDATES THAT the sensors are valid.
6	4. The system VALIDATES THAT no error is detected.
7	**1.1 Specific Alternative Flow**
8	RFS 1
9	1. The system sets WatchdogError as detected.
10	2. RESUME STEP 2.
11	Postcondition: The WatchdogError has been detected.
12	**1.4 Specific Alternative Flow**
13	RFS 4
14	1. **INCLUDE VARIATION POINT:** Storing Error Status.
15	2. ABORT.
16	Postcondition: There are some errors detected.

The analyst models the PL use case specifications according to RUCM and its PL extensions [9] (see Table 1). RUCM is based on restriction rules and keywords constraining the use of natural language. The PL extensions consist of new keywords (e.g., *INCLUDE VARIATION POINT* in Line 14) to capture variability in specifications. Moreover, they are used to model variability that cannot be represented in the PL use case diagram, e.g., optional use case steps and variant step order.

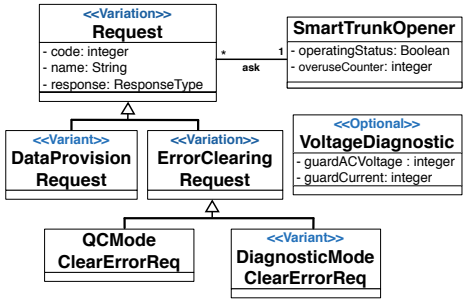

Figure 3: Part of the PL Domain Model for STO

To model variant domain entities, PUMConf employs the *Variation*, *Variant*, and *Optional* stereotypes proposed by

Ziadi and Jezequel [20] (Figure 3). The *Variant* and *Variation* stereotypes specify variability associated with an inheritance hierarchy, while variant entities which are not part of any inheritance hierarchy are stereotyped as *Optional*.

PUMConf uses NLP to automatically check (1) if the PL specifications conform to the RUCM template and its extensions, (2) if the PL use case diagram is consistent with its specifications, and (3) if the PL domain model is consistent with the PL use case specifications.

3.2 Elicitation of Configuration Decisions

PUMConf guides the analyst in making decisions for the PL use case diagram, specifications, and domain model.

3.2.1 Configuration Decisions for PL Diagrams

The analyst makes decisions for variation points in the PL diagram (see Figure 4). A decision is about selecting, for the product, variant use cases in the variation point.

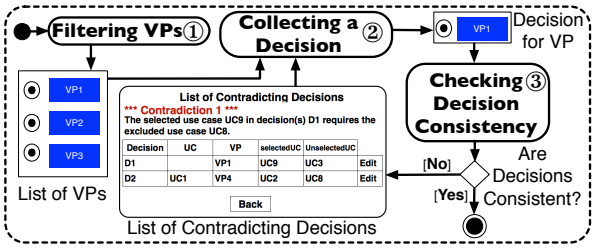

Figure 4: Overview of Making a Diagram Decision

Before each decision, PUMConf filters out variation points included by variant use cases (*Filtering VPs*) since the analyst can make a decision for these variation points only if the including variant use case is selected. For instance, in Figure 2, *Method of Clearing Error Status* is not considered if *Clear Error Status* is not selected (see Figure 5(a)).

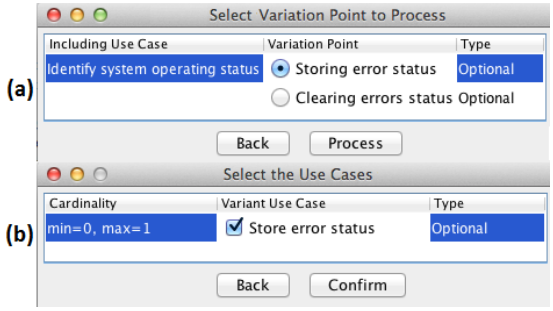

Figure 5: UI for Collecting a Decision for the PL Use Case Diagram

PUMConf receives a diagram decision from the analyst (*Collecting a Decision*). In Figure 5(b), *Store Error Status* in *Storing Error Status* is selected by the analyst.

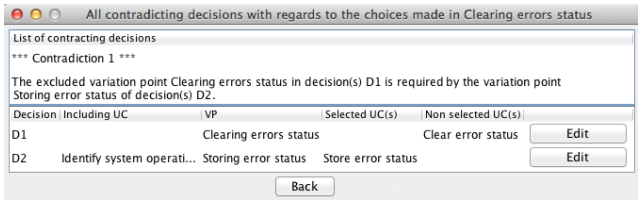

Figure 6: UI for Resolving Contradicting Decisions

After each decision, PUMConf traverses the PL diagram to determine previous decisions contradicting the current decision (*Checking Decision Consistency*). Two or more configuration decisions may contradict each other if they result in violating some variation point and variant dependency constraints (i.e., *require* and *conflict*). If there is any contradiction, the analyst is expected to update one or more decisions to resolve the contradiction (see Figure 6). PUMConf employs a consistency checking algorithm based on mapping variation points, use cases and variant dependencies to propositional logic formulas. For a given decision regarding a variation point in the PL diagram, the algorithm infers further, implicit decisions and only checks the satisfaction of the propositional formulas derived from the dependencies of the variation point [10].

In Figure 6, a contradiction between the decisions in *Storing Error Status* and *Clearing Error Status* is reported. The upper part of the user interface provides an explanation for the contradiction, while the bottom part lists the decisions involved in the contradiction, with an Edit button to update the corresponding decision.

3.2.2 Configuration Decisions for PL Specifications

PUMConf processes the PL use case specifications, using NLP, to retrieve variability information, i.e., optional steps, optional alternative flows, and variant order, in essential and variant use cases selected in the PL diagram. The analyst is asked to make decisions (e.g., selecting the appropriate optional steps) for the retrieved variability information.

3.2.3 Configuration Decisions for PL Domain Model

Our tool collects decisions for all optional and variant domain entities to generate the PS domain model. First, the analyst makes decisions for optional entities. Then, for each variation entity, the appropriate variant entities are selected.

3.3 Generation of PS Use Case Models

After all the decisions are made, the PS use case and domain models are generated from the PL models and the configuration decisions. The generation of the PS models are implemented as a set of transformation rules in Java.

3.3.1 Generation of PS Use Case Diagrams

PUMConf takes the PL use case diagram and the diagram decisions as input, and generates, using the transformation rules, the PS use case diagram as output (see Figure 7).

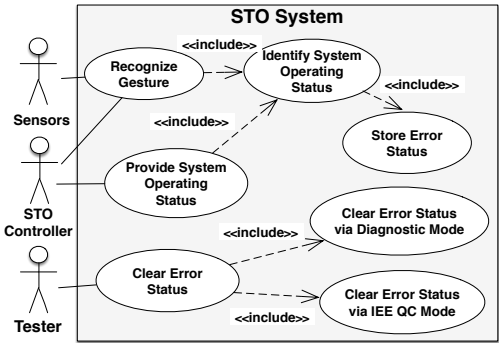

Figure 7: Part of the PS Use Case Diagram for STO

Example transformation rules for PL use case diagrams are as follows: (i) selected variant use cases become essential in the PS diagram, (ii) selected variant use cases are included by use cases including the corresponding variation points, and (iii) variation points are removed in the PS diagram.

3.3.2 Generation of PS Use Case Specifications

PUMConf takes the PL specifications and the diagram and specification decisions as input to generate the PS specifications as output (see Table 2). The generated PS use case specifications contain (1) selected variant use cases included in the flows of use cases (Line 14 in Table 2) (2) selected optional steps and alternative flows, and (3) decided orders for steps with a variant order.

Table 2: A PS Use Case Specification for STO

1	**USE CASE** Identify System Operating Status
2	**1.1 Basic Flow**
3	1. The system VALIDATES THAT the watchdog reset is valid.
4	2. The system VALIDATES THAT the RAM is valid.
5	3. The system VALIDATES THAT the sensors are valid.
6	4. The system VALIDATES THAT no error is detected.
7	**1.1 Specific Alternative Flow**
8	RFS 1
9	1. The system sets WatchdogError as detected.
10	2. RESUME STEP 2.
11	Postcondition: The WatchdogError has been detected.
12	**1.4 Specific Alternative Flow**
13	RFS 4
14	1. **INCLUDE USE CASE: Store Error Status.**
15	2. ABORT.
16	Postcondition: There are some errors detected.

3.3.3 Generation of PS Domain Models

PUMConf takes the PL domain model and the corresponding decisions as input, and generates the PS domain model as output (see Figure 8). The generated PS domain model includes all the selected optional and variant domain entities as well as mandatory entities.

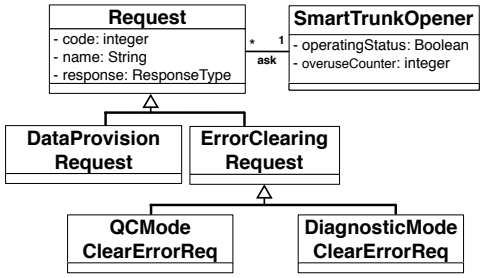

Figure 8: Part of the PS Domain Model for STO

4. EVALUATION

PUMConf has been evaluated in an industrial context for which a case study, i.e., STO, and a questionnaire study were reported in [10]. We applied PUMConf to the functional requirements of STO. Using our modeling method PUM [9], we modeled seven variation points, thirteen variant use cases, and seven variant dependencies in the PL use case models for STO. We used PUMConf to configure the PS use case and domain models of STO for four clients. All the generated PS models were confirmed by the IEE analysts to be correct and complete. The PL models that we derived using PUM were sufficient to make all the configuration decisions needed in PUMConf to generate the correct and complete PS models for the considered STO products.

To evaluate the output of PUMConf, we had a semi-structured interview with seven industrial participants. The participants held various roles (e.g., development manager and system engineer) and all had substantial experience in software development. The interview was preceded by a presentation illustrating our modeling method PUM, the PUMConf steps, and a tool demo. We also organized three hands-on sessions where the participants could use PUMConf.

We handed out two questionnaires to assess PUM and PUMConf in terms of adoption effort, expressiveness, comparison with current practice, and tool support. The results of the questionnaire showed that all participants agreed about the expressiveness and simplicity of PUM. They also agreed about the useful guidance provided by PUMConf for configuring PS models. The participants mentioned that the effort required to adopt PUMConf is reasonable although more practice and training were still needed to become familiar with the tool. They also stated that PUMConf should be extended to capture non-functional requirements.

5. IMPLEMENTATION & AVAILABILITY

PUMConf has been implemented as a DOORS Plug-in. This Plug-in activates the user interfaces of PUMConf and provides the features *consistency checking of PL artifacts* and *configuration of PS models*. We use GATE (http://gate.ac.uk/), an open source NLP framework, to annotate use case specifications. The NLP output contains the annotated use case steps. The annotations are used to check the consistency of the specifications, the diagram, and the domain model. PUMConf uses the same annotations to match the transformation rules for the generation of PS models.

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 generation of PS models has been implemented as a Java application. The DOORS eXtension Language (DXL) is employed to load the generated PS use case specifications into IBM DOORS.

PUMConf is approximately 17K 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/

6. CONCLUSION

We presented a tool, PUMConf, to support the configuration of requirements in a use case-driven development context. More specifically, it automatically generates PS use case and domain models from PL models. The key characteristics of our tool are (1) the consistency checking of PL artifacts by relying on Natual Language Processing, (2) the automated and interactive configuration support based on variability modeling for commonly used modeling artifacts, i.e., use case diagrams, specifications and domain models, and (2) the automatic generation of PS use case and domain models from PL models and configuration decisions. PUMConf has been evaluated over an industrial case study. The evaluation shows that our tool is practical and beneficial to configure PS use case and domain models in industrial settings. In the future, we plan to extend PUMConf to support regression test selection and change impact analysis in the context of use case-driven development and testing. We further plan to conduct more case studies to better evaluate the practical utility and usability of the tool.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[1] IEE (International Electronics & Engineering) S.A., http://www.iee.lu/.

[2] M. Alférez, J. Santos, A. Moreira, A. Garcia, U. Kulesza, J. Araújo, and V. Amaral. Multi-view composition language for software product line requirements. In *SLE'09*, pages 103–122, 2009.

[3] R. Bonifácio and P. Borba. Modeling scenario variability as croscutting mechanisms. In *AOSD'09*, pages 125–136, 2009.

[4] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *GPCE'05*, pages 422–437, 2005.

[5] D. Dhungana, P. Grünbacher, and R. Rabiser. The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering*, 18:77–114, 2011.

[6] M. Eriksson, J. Borstler, and K. Borg. Managing requirements specifications for product lines - an approach and industry case study. *Journal of Systems and Software*, 82:435–447, 2009.

[7] M. Eriksson, H. Morast, J. Borstler, and K. Borg. The pluss toolkit - extending telelogic doors and ibm-rational rose to support product line use case modeling. In *ASE'05*, pages 300–304, 2005.

[8] 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*, pages 255–265, 2004.

[9] 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*, pages 338–347, 2015.

[10] I. Hajri, A. Goknil, L. C. Briand, and T. Stephany. Configuring use case models in product families. *Software and Systems Modeling*, 2016.

[11] G. Halmans and K. Pohl. Communicating the variability of a software-product family to customers. *Software and Systems Modeling*, 2:15–36, 2003.

[12] M. Mendonca, M. Branco, and D. Cowan. S.P.L.O.T. - software product lines online tools. In 761-762, editor, *OOPSLA'09*, 2009.

[13] M. Moon and K. Yeom. An approach to develop requirement as a core asset in product line. In *ICSR'04*, pages 23–34, 2004.

[14] M. Moon, K. Yeom, and H. S. Chae. An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *IEEE Transactions on Software Engineering*, 31(7):551–569, 2005.

[15] A. Nöhrer and A. Egyed. C2O configurator: a tool for guided decision-making. *Automated Software Engineering*, 20:265–296, 2013.

[16] A. K. Thurimella and D. Janzen. Metadoc feature modeler: A plug-in for IBM rational DOORS. In *SPLC'11*, pages 313–322, 2011.

[17] 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*, pages 385–396, 2015.

[18] C. Wang, F. Pastore, A. Goknil, L. C. Briand, and M. Z. Z. Iqbal. UMTG: a toolset to automatically generate system test cases from use case specifications. In *ESEC/SIGSOFT FSE'15*, pages 942–945, 2015.

[19] T. Yue, L. C. Briand, and Y. Labiche. Facilitating the transition from use case models to analysis models: Approach and experiments. *ACM Transactions on Software Engineering and Methodology*, 22(1), 2013.

[20] T. Ziadi and J.-M. Jezequel. Product line engineering with the uml: Deriving products. In *Software Product Lines*. Springer, 2006.