Modeling bCMS Product Line Using Feature Model, Component Family Model and UML

Shuai Wang, Shaukat Ali

Certus Software V&V Center, Simula Research Laboratory, Norway

{shuai, shaukat}@simula.no

Abstract. In the context of Model-Based Engineering (MBE) of product lines, effort required to develop models can be significantly reduced by applying systematic product line modeling and configuration methodologies. Our previous work presented models of bCMS developed using AspectSM, a UML profile for Aspect-Oriented Modeling (AOM), which was defined to model crosscutting behaviors using extended UML state machines, with the objectives of minimizing modeling effort and the learning curve for modeling crosscutting behavior. However, such approach still requires users to be familiar with specific expertise and concepts on various UML behavior models. In this paper, we extend our previous work using Feature Model (FM) and Component Family Model (CFM) to model bCMS product line. More specifically, a FM is designed and developed to capture all variations points for bCMS product line and a CFM is built to provide an abstraction layer on top of the configurable state machines. With our current methodology, a user doesn't need to acquire expertise on behavioral modeling and can simply configure models for a product by selecting features in FM and configuring provided attributes in CFM.

1 Selection of the Approach

In this section, we first present the classification of the selected approach (Section 1.1) followed by a brief overview of the approach (Section 1.2).

1.1 Classification of the Approach

Our proposed approach is a mix of aspect-oriented, feature-oriented, and object-oriented paradigms. Feature modeling is used to capture variability and commonality of product lines, aspect-oriented modeling is used for modeling crosscutting behaviors on UML state machines, whereas standard UML state machines (object-oriented) are used for modeling functional behavior of product lines.

The approach is specifically designed to support model-based testing of new products in a product line and hence can be classified as an approach in the validation and verification phase of software development.

In this paper, we modeled software product line and also assumed some new varia-

tion points in addition to the mentioned in the description of the case study.

1.2 Brief Overview of the Approach

In our previous work, we proposed a product line modeling and configuration methodology that systematically captures various types of behavioral variability of a product line using behavioral models, including standard UML state machines modeling functional behaviors and aspect state machines modeling non-functional behaviors [1]. Results of applying this methodology to a product line of Video Conferencing Systems (VCSs) (called Saturn developed by Cisco Systems, Norway) revealed that the modeling effort required could be significantly reduced as compared to an approach where a new product has to be modeled from the scratch [1].

While investigating the adoption of this methodology in Cisco, we discovered that test engineers are required to have expertise on developing aspect state machines and Object Constraint Language (OCL) constraints—two key artifacts required to configure a product. Moreover, test engineers are also required to be familiar with concepts of UML class diagrams, UML state machines, aspect class diagrams, and aspect state machines—four key UML diagrams used for modeling functional and non-functional behaviors of the Saturn product line. To ease the adoption of the MBE approach in industry, it is essential to seek a solution that can shield test engineers from all the above-required modeling expertise.

To address such challenges, we propose an extension to our previously modeling and configuration methodology using Feature Model (FM) and Component Family Model (CFM). More specifically, a FM is developed to capture variabilities of a product line and a CFM is designed to associate its configurable parameters to model elements of a large number of behavioral models developed as part of our previous work [1]. Test engineers are only required to perform selection and configuration through the front-end for FM, such that all relevant behavioral models can be selected and configured automatically, based on the links built between FM and CFM, and between CFM and the repository of behavioral models. With our extended methodology, most of the details of the behavioral models are hidden from a test engineer, which is not the case for our previous work [1]—test engineers need to create, configure and select behavioral models. To compare with our previous work, this extended methodology significantly reduces the complexity of configuration; thereby significantly reducing required effort and cost (e.g., in terms of training). In addition, the extended methodology does not require test engineers to have expertise of UML modeling, aspect modeling, and OCL constraints specification, which based on our study and experience of working with our industrial partner, is an obstacle to apply MBT in practice. The results of evaluation showed that our methodology significantly reduces the complexity of configuration; thereby reducing required modeling effort. Moreover, the need to acquire expertise of modeling is also eliminated [2].

In this paper, we models bCMS product line using the above-mentioned product line methodology based on the existing behavioral models (e.g., UML class diagrams, state machines, aspect state machines and aspect diagrams) we developed in [3].

2 Background

In this section, we briefly introduce aspect state machines (Section 2.1), followed by describing feature model and component family model (Section 2.2 and 2.3).

2.1 Aspect State Machines(AspectSM)

AspectSM [4] is a UML profile supporting the modeling of system robustness behavior—a very common type of crosscutting behavior in communication and control systems. Though AspectSM was originally defined to support scalable, model-based, robustness testing, including test case and oracle generation, it is applicable to model crosscutting behaviors and AspectSM in particular simply relies on UML state machines to do it all. In AspectSM, the core functionality of a system is modeled as one or more standard UML state machines (called base state machines). Crosscutting behavior of the system (e.g., robustness behavior) is modeled as aspect state machines using the AspectSM profile. The AspectSM profile specifies stereotypes for all features of Aspect-Oriented Modeling (AOM), in which the concepts of *Aspect*, *Joinpoint*, *Pointcut*, *Advice*, and *Introduction* are the most important ones and are defined as stereotypes. More details and examples can be found in [1].

2.2 Feature Model (FM)

Feature modeling is a hierarchical modeling approach for capturing commonalities and variabilities in product line [5-7]. FM can be represented as a 2-*tuple (features, constraints)* with four types of features, namely *mandatory*, *optional*, *alternative* and *or*. A *mandatory* feature means it must be included if its father feature is included in the current selection. The selection of an *optional* feature is optional even if its father feature is included. A father feature with a set of *alternative* features describes that only one of the *alternative* features can be included if their father feature is included. A father feature with a set of *or* features means at least one of the *or* features is included if their father feature is included. In addition, FM contains cross-tree constraints which are supplementary relationships among unrelated features. There are two kinds of such constraints, namely *require* and *mutually exclusive*. A *require* relation among two features (a *source* and a *target*) means if the *source* feature is included into the current selection, the *target* feature must also be included. A *mutually exclusive* relation has the opposite meaning, saying that if the *source* feature is included then the *target* feature cannot be included into the current selection.

2.3 Component Family Model (CFM)

A CFM is used to represent how products are assembled and generated in a product line by modeling relations among software architectural elements [8]. CFM can be represented as *a 4-tuple (components, parts, source elements, restrictions). Components* are named entities organized into a tree-like structure that can be of any depth. Each *component* represents one or more functional elements of the products in product

line (e.g. C functions, Java classes). *Parts* are named and typed entities. Each *part* belongs to a *component* and contains one or more *source elements*. A *part* can be associated with given programming language features, classes or objects, but it can also be associated with other key elements. A *source element* is an unnamed but typed entity. *Source elements* are usually used to determine how the source code for the specified element is generated. *Restrictions* specify conditions under which a *component*, *part* or *source element* may be excluded from a final selection [8, 9].

3 Methodology

In this section, we first briefly present the behavioral model repository for modeling the system behaviors for a product line (Section 3.1). Second, we present our methodology that is based on Feature Model (FM) and Component Family Model (CFM) for bCMS product line. Since our context is related with bCMS, we will call our FM as FM_b and CFM as CFM_b. More specifically, FM_b is first presented to capture the variabilities of bCMS product line (Section 3.2) followed by CFM_b to associate its configurable parameters to model elements of a large number of behavioral models and related OCL constraints (Section 3.3). Afterwards, we present the configuration process for a variant (Section 3.4)

3.1 Behavioral Model Repository

Suppose, we have a product line *P* that has a set of products ; where *np* is the total number of products in *P*. To capture the behaviors of all the products, in our previous work [3], we developed a configurable product line Behavioral Model Repository (*BMRepository*).

$$BMRepository = \{SM, CD, ASM, OCL\}$$

$SM = \{sm_1, sm_2, \dots, sm_{n_{sm}}\}$ is a set of UML state machines in the repository and each sm_i is used to model a functional behavior of the product line. An example of such behavior includes the fire station coordinator behavior, which is modeled as a set of state machines in the context of bCMS product line. n_{sm} is the total number of state machines in the repository.

$CD = \{cd_1, cd_2, \dots, cd_{n_{cd}}\}$ contains a set of UML class diagrams capturing the structure of the system including its Application Programming Interfaces (APIs), state variables, software and hardware configurations. . Notice that each sm_i is linked to exactly one cd_j and cd_j may have a set of associated state machines from *SM*. n_{cd} is the total number of UML class diagrams in the repository.

$ASM = \{asm_1, asm_2, \dots, asm_{n_{asm}}\}$ is a set of aspect state machines modeling system behaviors including functional and non-functional behaviors. A typical example of non-functional behavior in the bCMS product line is the performance behavior. n_{asm} is the total number of aspect state machines specifying functional and non-functional behaviors in the repository.

$OCL = \{ocl_1, ocl_2, \dots, ocl_{n_{ocl}}\}$ is a set of OCL constraints for configuration. An

OCL constraint can be written to configure to configure corresponding state machines, aspect state machines. n_{ocl} is the total number of OCL constraints in the repository.

3.2 Feature Model for bCMS (FM_b)

Functionalities and non-functionalities of a product line *P* can be represented as $FM\_b = \{f_1, f_2, f_3, \dots, f_{nf}\}$, where nf is the total number of features for *P*. Each functionality or non-functionality is associated with a feature f_i in FM_b. Notice that the types of features in FM_b can be *mandatory*, *optional*, *alternative* and *or* as discussed in Section 2.2. Notice that for now, all the features are named as the same as description in the bCMS document since it is more comprehensible for users to configure these variation points (The names of features can be changed in any way users can better understand).

A set of cross-tree constraints is added to the FM_b since functionalities may be related to each other. All the constraints can be represented as $CONS = \{cons_1, cons_2, cons_3, \dots, cons_{ncons}\}$, where $ncons$ is the number of constraints. Each $cons_i$ can be either *require* or *mutually exclusive*, i.e., $cons_i$ can be represented as $cons_i = require\ (f_m, f_n)$ or $cons_i = exclusive\ (f_m, f_n)$, where f_m is the source feature and f_n is the target feature (Section 2.2).

The variants can be configured by performing different selections of the features in FM_b, i.e., a specific variant can be represented as a subset of features. We developed the FM_b for bCMS product line, which contains 40 features (10 *mandatory* feature, 11 *optional* feature, 13 *alternative* feature and 6 *or* feature) and 6 *require* constraints in total. Besides, we need to mention that building FM_b is one-time manual effort since the bCMS product line doesn't change significantly. A complete FM_b is shown as Fig. 1 in appendix and more details can be consulted in [10].

3.3 Component Family Model for bCMS (CFM_b)

Our CFM_b is represented as $CFM\_b = \{c_1, c_2, c_3, \dots, c_n\}$ comprising of a set of components, where n is the number of components. Each component represents a behavior task and can be hierarchically decomposed into parts representing various behavioral models (e.g., class diagrams) $c_i = \{pa_{i1}, pa_{i2}, pa_{i3}, \dots, pa_{in}\}$, where in is the number of parts belonging to c_i. Each part pa_{ij} can represent one behavior, such as fire station coordinator at the same time being associated with a set of state machines (*SM*) and aspect state machines (*ASM*) in the repository. Meanwhile, each part consists of a set of attributes representing different information for configuration (e.g., number of crisis): $A\_pa_{ij} = \{a_{ij1}, a_{ij2}, a_{ij3}, \dots, a_{ijn}\}$, where ijn is the number of attributes belonging to pa_{ij}.. Notice that all these attributes are associated with a set of relevant configurable attributes in their corresponding class diagrams, state machines and aspect state machines for configuration.

Afterwards, restrictions are assigned to components or parts, which constrain relations between components or parts in CFM_b and features in FM_b. Notice that each

component or part can be linked with one or more features in FM_b via restrictions (i.e., Each component or part can have any number of restrictions). A component or part cannot be included into the final selection for a product unless its restrictions evaluate to true. In general, our CFM_b for bCMS includes 15 components and 20 parts with 26 restrictions and 10 attributes. A complete CFM_b is shown as Fig. 2 in appendix and more details can be consulted in [10].

3.4 Configuration Process for a Variant

For each variant, the following two steps are involved for configuration: 1) selecting a set of relevant features in FM_b for a variant; 2) configuring the selected attributes in CFM_b as the result of step 1). Afterwards, relevant behavioral models (e.g., class diagrams and state machines) will be selected and configured automatically in the repository. A concrete example for configuring a variant is shown as Fig. 3 in appendix and can be consulted in [10] for more details.

References

1. Ali, S., Yue, T., Briand, L. C., and Walawege, S.: A product line modeling and configuration methodology to support model-based testing: an industrial case study. In Proceedings of the International Conference Model Driven Engineering Languages and Systems (MODELS). pp. 726-742, 2012.
2. Wang, S., Ali, S., Yue, T., and Liaaen, M.: Using Feature Model to Support Model-Based Testing of Product Lines: An Industrial Case Study. In Proceedings of the International Conference of Software Quality (QSIC). pp. 75-84, 2013.
3. Ali, S: Modeling bCMS using AspectSM. In Proceedings of the MODELS workshop Comparing Modeling Approach (CMA), 2012.
4. Ali, S., Briand, L.C., Hemmati, H.: Modeling Robustness Behavior Using Aspect-Oriented Modeling to Support Robustness Testing of Industrial Systems. Software & Systems Modeling (SOSYM), 11(4), pp.633-670, 2012.
5. Benavides, D., Segura, S., and Cortés, A. R.: Automated analysis of feature models 20 years later. A literature review. Information Systems. (35), 615–636, 2010.
6. Wang, S., Gotlieb, A., Ali, S., and Liaaen, M.: Automated Selection of Test Cases using Feature Model: An Industrial Case Study. In Proceedings of the International Conference of Model-Driven Engineering Languages and Systems (MODELS), pp. 237-253, 2013.
7. Wang, S., Gotlieb, A., Liaaen, M., and Briand, L.C.: Automatic Selection of Test Execution Plans from a Video Conference System Product Line. In Proceedings of the MODELS workshop VARiability for You (VARY' 12), pp. 32-37, 2012.
8. Pure systems GmbH: Variant management with pure::variants. Technical white paper. Available from http://web.pure- systems.com. 2003.
9. Pure systems GmbH: Pure::Variants User's Guide. Available from http://web.pure-systems.com, 2011.
10. Wang, S., and Ali, S.: Modeling Specification for bCMS Product Line using Feature Model, Component Family Model and UML. Repository for Model-Driven Development (ReMoDD). Available from: http://www.cs.colostate.edu/remodd/v1/content/modeling-specification-bcms-product-line-using-feature-model-component-family-model-and-uml, 2013.

Appendix

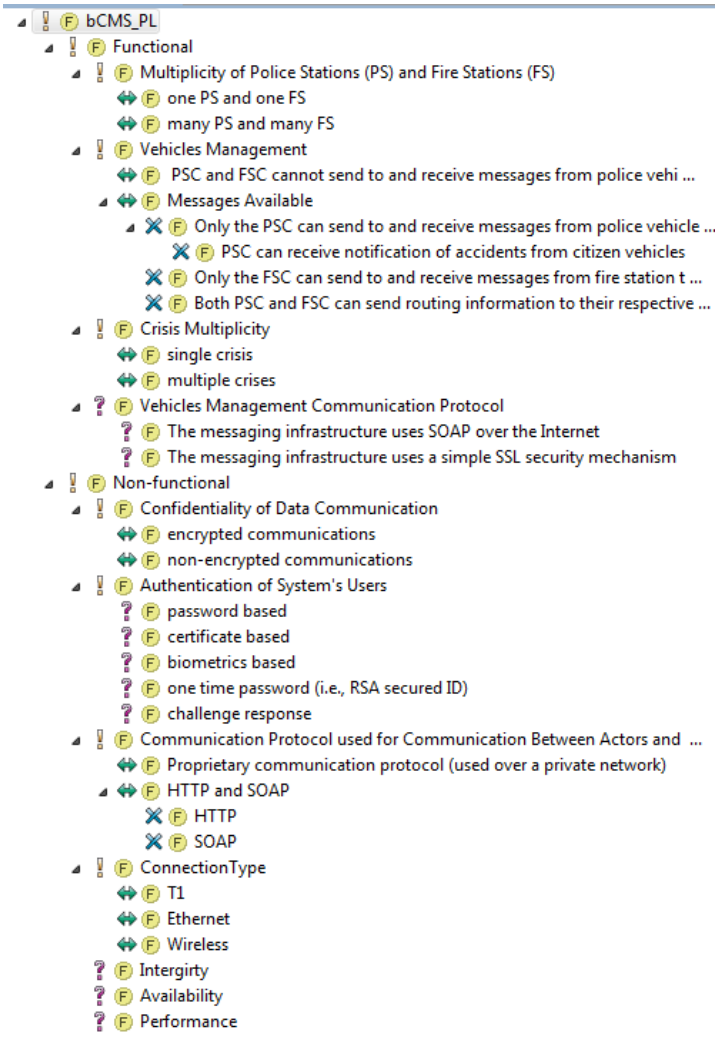


Fig. 1. Feature model for bCMS product line (FM_b)

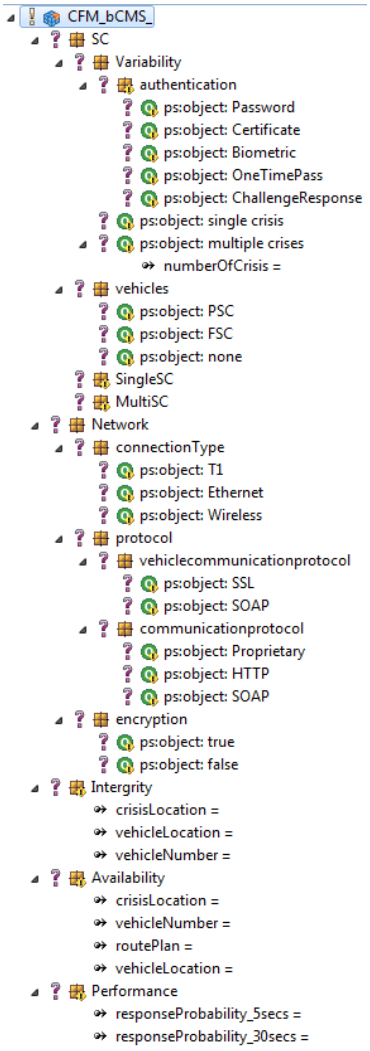
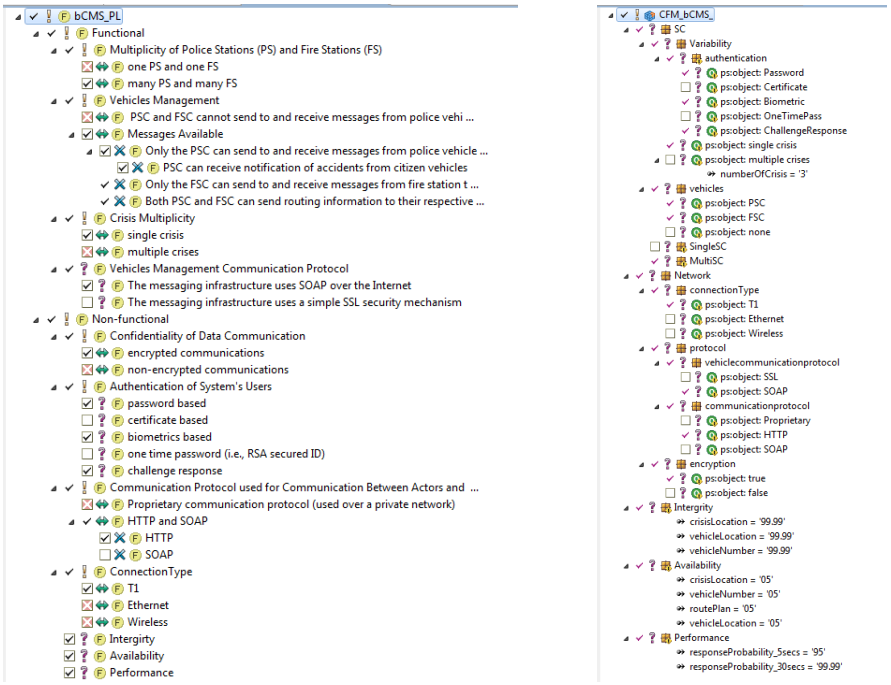


Fig. 2. Component family model for bCMS product line (CFM_b)



(a) Selection of features for a variant **(b)** Relevant component family model

Fig. 3. An example for configuration process for a variant