Formal Modeling of Product-Line Variant Requirements

Shamim Ripon, Sk. JahirHossain, Keya Azad, and Mehidee Hassan

***Abstract*—Formal verification of variant requirements has gained much interest in the software product line (SPL) community. Feature diagrams are widely used to model product line variants. However, there is a lack of precisely defined formal notation for representing and verifying such models. This paper presents an approach to modeling and verifying SPL variant feature diagrams using first-order logic. It provides a precise and rigorous formal interpretation of the feature diagrams. Logical expressions can be built by modeling variants and their dependencies by using propositional connectives. These expressions can then be validated by any suitable verification tool. A case study of a Computer Aided Dispatch (CAD) system variant feature model is presented to illustrate the verification process.**

***Index Terms*—Product line, reuse, first-order logic, variants.**

I. Introduction

Software product line is a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or missions and that are developed from a common set of core assets in a prescribed way [1]. The main idea of software product line is to explicitly identify all the requirements that are common to all members of the family as well as those that vary among products in the family. Common requirements are easy to handle but problem arises from the variant requirements. Different variants might have dependencies on each other. Tracing multiple occurrences of any variant and understanding their mutual dependencies are major challenges during domain modeling. While each step in modeling variants may be simple but problem arises when the volume of information grows. As a result, the impact of variant becomes ineffective on domain model. Therefore, product customization from the product line model becomes unclear and it undermines the very purpose of domain model.

This paper presents our work-in-progress logic verification approach for variant requirements of software product line. In our earlier work [2] we have shown how a `Unified Tabular' representation along with a decision table can be augmented with feature diagram to overcome the hurdles of variant management during an explosion of variant dependencies. However, defining such table involves manual handling of variants and hence, formal verification is not directly admissible for such approach. This paper uses first-order logic to represent product line variants and their dependencies. First we use extended versions of UML to model product line variants. The logical representation of the feature model is then presented using propositional logic allowing us to logically verify the models. We present a case study of Computer Aided Dispatch (CAD)[1] system product line.

In the remainder of the paper, Section 2 gives an overview of the CAD domain model and how variants of the CAD domain are modeled. How UML can be used to model variants using UML extensions is presented in Section 3. The logical definitions of variant models and their dependencies are presented in Section 4. Finally, we conclude our paper and outline our future plans in Section 5.

II. CAD Overview

A Computer Aided Dispatch system (CAD) is a mission-critical system that is used by police, fire and rescue, health service, port operation, taxi booking and others. Fig. 1 depicts a basic operational scenario and roles in a Police CAD system.

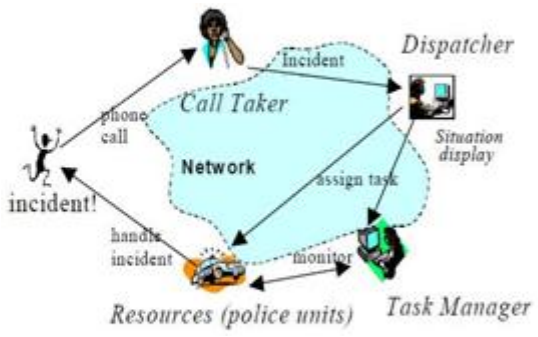


Fig. 1. Basic operational scenario in a CAD system for police

When an incident has occurred, a caller reports the incident to the command and control center of the police unit. A Call Taker in the command and control center captures the details about the incident and the Caller, and creates a task for the incident. There is a Dispatcher in the system whose task is to dispatch resources to handle any incident. The system shows the Dispatcher a list of un-dispatched tasks. The Dispatcher examines the situation, selects suitable Resources (e.g. police units) and dispatches them to execute the task. The Task Manager monitors the situation and at the end, closes the task. Different CAD members have different resources and tasks for their system. At the basic operational level, all CAD systems are similar. Some of the variants identified in CAD domain are: (i) *Call taker and dispatcher roles* (ii)

Manuscript received July July 20, 2012; revised September 15, 2012.
The authors are with the Department of Computer Science and Engin (e-mail: dshr@ewubd.edu).

[1]The CAD case study is adopted from Software Engineering Research group, Computer Science, National University of Singapore, http://xvcl.comp.nus.edu.sg/xvcl/cad/CAD.html

Validation (iii) *Un-dispatched task selection rule*etc.

A. *Modeling Variants*

Feature modeling is an integral part of the FODA method and the Feature Oriented Domain Reuse Method (FORM) [4]. Features are represented in graphical form as trees. The internal nodes represent the variation point and leaves represent the values of the variation points, known as variants. The root node of a feature tree always represents the domain whose features are modeled. The remaining nodes represent features which are classified into three types: *Mandatory*, *Optional*, and *Alternative*. Mandatory features are always part of the system. Optional features may be selected as a part of the system if their parent feature is in the system. Alternative features, on the other hand, are related to each other as a mutually exclusive relationship. There are more relationships between features. Or-feature [5] connects a set of optional features with a parent feature, either common or variant. Feature diagram also depicts the interdependencies among the variants which describes the selection of one variant depends on the selection of the dependency connected variants. A CAD feature diagram is illustrated in Fig. 2.

III. Modeling Variants in UML

Feature models are widely used in domain analysis to model the common as well as variant requirements of the application domain. However, the semantics of a domain are not fully expressed by feature models. As a result, there is a need for other notations to support feature models which can enhance the meaning of thedomain concept. The Unified Modeling Language (UML), a standardized notation for describing object-oriented models, can be used with feature model to depict the domain concept properly.UML is targeted at modeling single system rather than system families. In order to use UML diagrams to represent the model of the system family simple extension mechanisms [11] of UML,namely stereotypes and tagged values are used here. The stereotype <<variant>> designates a model element as a variant and the tagged values are used to keep trace of the models and their corresponding variant elements. It is claimed that adding only thestereotype <<variants>> does not represent the types of variants and proposed another extension where thenotion of *variation point* is used to make variation point visible in use case diagram, represented as a triangleand variant is used to make variant in use cases explicitly.

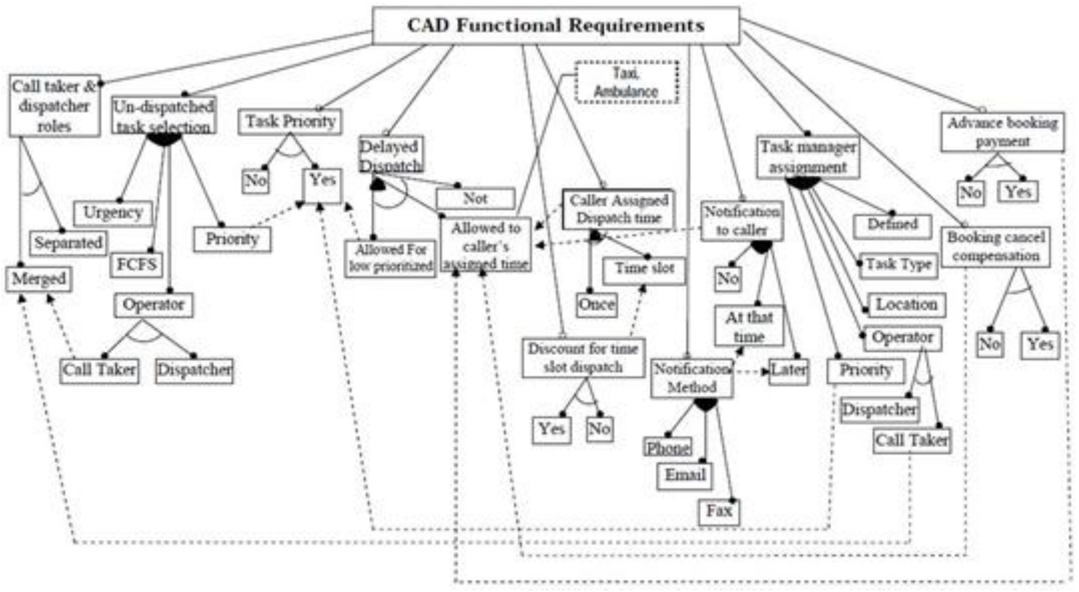


Fig. 2. CAD feature diagram with dependencies.

Fig. 3 illustrates the use case diagram added with variants of 'Create Task' activity. An *exclude* denotes that when one feature is selected other related feature cannot be selected. A *requires* relation indicates that when there is a relation from one feature (source) to another (target), then if the source feature is selected the target feature has to be selected as well. UML activity diagrams are used to identify the workflow of any activity. As use cases are the source of information for creating activity diagrams, whenever there is change occurs in use cases due to using <<include>> or <<extend>>, then corresponding activity diagrams should be updated. The activity diagram of creating a task is shown in Fig. 4.

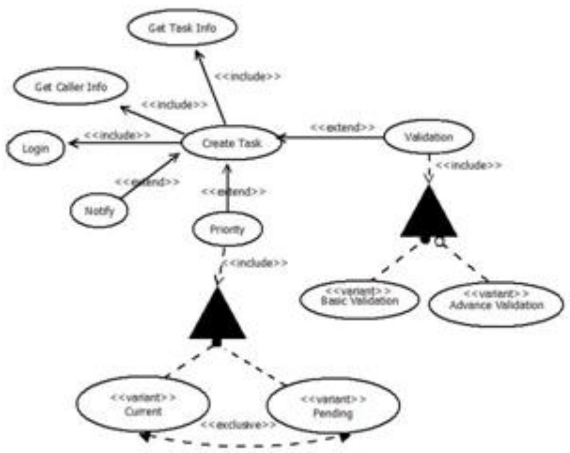


Fig. 3. Create task Use case diagram with variants.

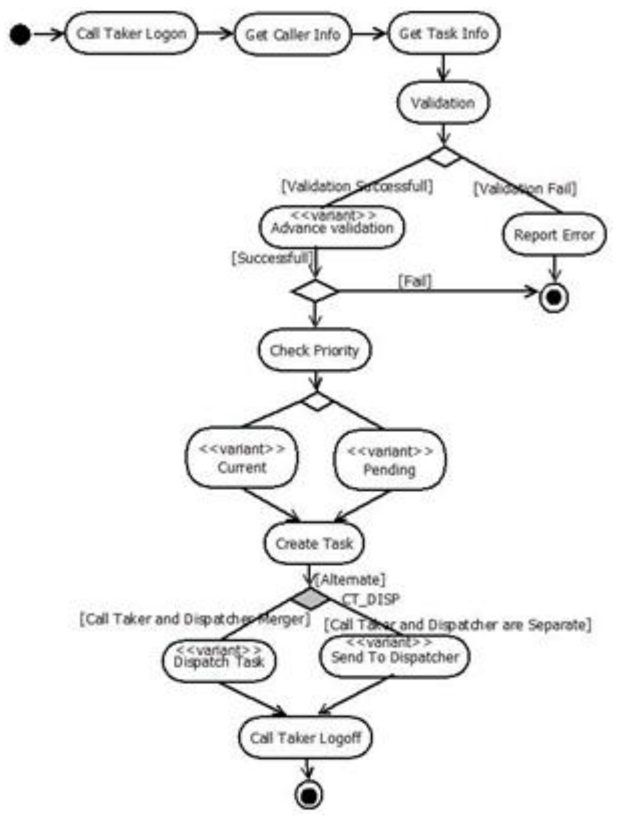


Fig. 4. Create task activity diagram with variants.

IV. Logic Representation

A feature model is a hierarchically arranged set of features. The relationships between a parent (variation point) feature and its child features (variations) are categorized as follows:

- *Mandatory*: A mandatory feature is included if its parent feature is included.
- *Optional*: An optional feature may or may not be included if its parent is included.
- *Alternative*: One and only one feature from a set of alternative features are included when parent feature is included.
- *Optional Alternative*: One feature from a set of alternative features may or may not be included if parent in included.
- *Or*: At least one from a set of *or* feature is included when parent is included.
- *Optional Or*: One or more optional feature may be included if the parent is included.

The logical notations of these features are defined in Fig. 5.

A feature model can be considered as a graph consists of a set of sub-graphs. Each sub-graph is created separately by defining a relationship between the variation point (v_i) and the variants ($v_{i,j}$) by using the expressions shown in Fig. 5. For brevity, a partial feature graph is drawn from CAD feature model in Fig. 6. The complexity of a graph construction lies in the definition of dependencies among variants. When there is a relationship between cross-tree (or cross hierarchy) variants (or variation points) we denote it as a dependency. Typically dependencies are either *inclusion* or *exclusion*: if there is a dependency between p and q, then if p is included then q must be included (or excluded). Only inclusion dependencies are shown in this paper. Dependencies are drawn by dotted lines (e.g., from $v_{2.3.1}$to $v_{1.1}$).

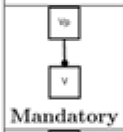
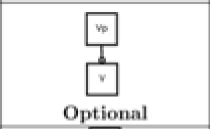
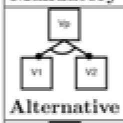
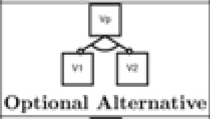
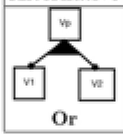
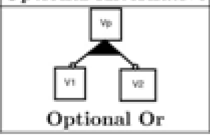
Type	Logic Expression	Type	Logic Expression
 Mandatory	$v_p \Leftrightarrow v$	 Optional	$v \Rightarrow v_p$
 Alternative	$v_p \Leftrightarrow (v_1 \oplus v_2)$	 Optional Alternative	$(v_1 \oplus v_2) \Rightarrow v_p$
 Or	$v_p \Leftrightarrow (v_1 \vee v_2)$	 Optional Or	$(v_1 \vee v_2) \Rightarrow v_p$

Fig. 5. Logical notations for feature models

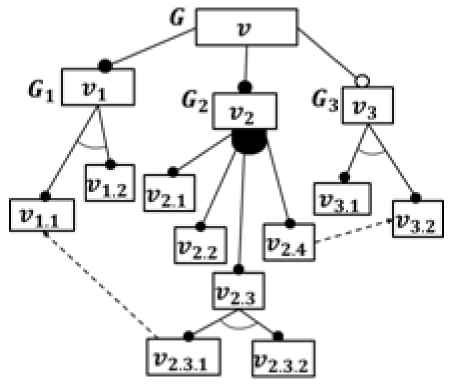


Fig. 6. A partial CAD feature graph using symbolic notations

A. Analysis of Variants

Automatic analysis of variants is already identified as a critical task [6]. Various operations of variant analysis are suggested in [7], [8]. Our logical representation can define and validate a number of such analysisoperations. The validation of a product line model is assisted by its logical representation. While constructinga single system from a product line model we assign TRUE (T) value to selected variants and FALSE (F) tothose not selected. After substituting these values to product line model, if TRUE value is evaluated, we call the model as valid otherwise the model is invalid. A product graph is considered to be valid if the mandatory sub-graphs are evaluated to TRUE.

Example 1**:** Suppose the selected variants are $v_1, v_{1.1}, v_2, v_{2.3}, v_{2.3.1}, v_{2.4}, v_3andv_{3.2}$. We check the validity of the subgraphs G_1, G_2 and G_3 by substituting the truth values of the variants of the subgraphs

$$G_1: (v_{1.1} \oplus v_{1.2}) \Leftrightarrow v_{1.2} \ = (T \oplus F) \ \Leftrightarrow T \ = T$$

$$G_2: v_2 \Leftrightarrow v_{2.1} \vee \ v_{2.2} \vee \ v_{2.3} \vee v_{2.4} = v_2 \Leftrightarrow v_{2.1} \vee \ v_{2.2} \vee \left((v_{2.3.1} \oplus \ v_{2.3.2}) \Leftrightarrow \ v_{2.3}\right) \vee v_{2.4}$$

$$= T \Leftrightarrow T \vee F \vee \left((T \oplus F) \Leftrightarrow T\right) \vee T \quad = T$$

$$G_3: (v_{3.1} \oplus \ v_{3.2}) \Leftrightarrow v_3 = (F \oplus T) \Leftrightarrow T \ = T$$

As the sub-graphsG_1, G_2, and G_3 are evaluate to TRUE, the product model is valid. However, variant dependencies are not considered in this case. Dependencies among variants are defined as additional constraints which must be checked separately apart from checking the validity of the subgraphs.

Evaluating the dependencies of the selected variants, we get ,
Dependency: $(v_{2.3.1} \Rightarrow v_{1.1}) \wedge (v_{2.4} \Rightarrow v_{3.2}) = (T \Rightarrow T) \wedge (T \Rightarrow T) = T$

It concludes that the selected features from the feature model create a valid product.

Example 2: Similar to Example 1, suppose the selected variants are $v1, v2, v2.1, v2.3, v2.3.1, v2.4$, and $v3$. Initially, neither $v1.1$ nor $v3.2$ is selected. However, there is inclusion dependency between $v2.3.1$and $v1.1$, and between $v2.4$ and $v3.2$ and the dependent variants are not selected. Therefore, the whole product model becomes invalid. To handle such scenarios where dependency decision can be propagated, a set of rules has been defined using first-order logic. One of the rules indicates that if there is an inclusion dependency between x and y and if x is selected then y will be selected. Due to inclusion dependency, both $v1.1$ and $v3.2$ will be automatically selected and the product graph will be evaluated to TRUE resulting in a valid model. It indicates how the model supports decision propagation. Inconsistency, dead featureetc.

V. Conclusions

This paper presented an approach to formalizing and verifying SPL variant models by using formal reasoning techniques. We provided formal semantics of the feature models by using first-order logic and specified the definitions of six types of variant relationships. Examples are provided describing various analysis operations, such as validity, inconsistency, dead feature detection etc. We are currently working towards answering all the analysis questions mentioned in [7], [8].

In contrast to other approaches [9], [10], our proposed method defines across-graph variant dependencies as well as dependencies between variation point and variants. These dependencies are defined as additional constraints while creating sub-graphs from the feature graph. Comparing to that presentation, our definition relies on first-order logic which can be directly applied in many verification tools. Currently, we are encoding our logical representation and predicate rules in Alloy [3]. It will allow us to automatically model check and analyse the logical representation.

References

[1] P. Clements and L. Northrop, "Software product lines: Practices and patterns," *3rd ed. Addison-Wesley Professional*, 2001.

[2] S. Ripon, "A unified tabular method for modeling variants of software product line," *SIGSOFT Software Engineering Notes*, vol. 37, May 2012.

[3] D. Jackson, "Alloy: A lightweight object modeling notation," *ACM Trans. Softw. Eng. Methodol.*, vol. 11, pp. 256-290, April 2002.

[4] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh, "FORM: A feature-oriented reuse method with domain-specific reference architectures," *Ann. Softw. Eng.*, vol. 5, pp. 143-168, January 1998.

[5] K. Czarnecki and U. W. Eisenecker, "Generative programming: methods, tools, and applications," New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000.

[6] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University Software Engineering Institute November 1990.

[7] D. Benavides, S. Corte, A. Ruiz, P. Trinidad, and S. Segura, "A survey on the automated analyses of feature models," in *JISBD*, 2006, pp. 367-376.

[8] D. Benavides, S. Segura, S. Cort'e, and A. Ruiz, "Automated analysis of feature models 20 years later: A literature review," *Inf. Syst.*, vol. 35, pp. 615-636, 2010.

[9] M. Mannion, "Using First-Order Logic for Product Line Model validation," in *Proceedings of the Second International Conference on Software Product Lines*, London, UK, 2002, pp. 176-187.

[10] W. Zhang, H. Zhao, and H. Mei, "A propositional logic-based method for verification of feature models," in *Formal Methods and Software Engineering*. vol. 3308, J. Davies, W. Schulte, and M. Barnett, Eds., ed: Springer Berlin / Heidelberg, 2004, pp. 115-130.

[11] G. Halmans and K. Pohl, "Communicating the variability of a software product family to customers," *Software and Systems Modeling*, vol. 2, pp. 15-36, Springer, Hamburg, March 2003.