

# An Aspect-Oriented Approach for Use Case Based Modeling of Software Product Lines

Stéphane S. SOMÉ, Pauline ANTHONYSAMY

SITE, University of Ottawa, Ottawa, Canada.  
Email: [ssome@site.uottawa.ca](mailto:ssome@site.uottawa.ca)

Received May 29<sup>th</sup>, 2009; revised June 22<sup>nd</sup>, 2009; accepted June 27<sup>th</sup>, 2009.

## ABSTRACT

*Software Product Line Development advocates software reuse by modeling common and variable artefacts separately across members of a family of products. Aspect-Oriented Software Development aims at separation of concerns with “aspects” to increase modularity, reusability, maintainability and ease of evolution. In this paper, we apply an aspect-oriented use case modeling approach to product line system modeling. A use case specification captures stakeholders concerns as interactions between a system and its actors. We adapt our previous work with the introduction of a “variability” relationship for the expression of variabilities. This relationship is used to model variable and common behaviours across a family of products as use cases. A variability composition mechanism enables building of executable behaviour models for each member of a product line family by integrating common elements with the applicable variable elements.*

**Keywords:** Product Lines, Use Cases, Aspects, Requirements Modeling

## 1. Introduction

The importance of a *Software Product Line (SPL)* emerged from the field of software reuse when developers realized that reusing development artefacts such as requirements, designs, and components across different members of a product family significantly reduces cost, effort and time. According to Clements *et al.* [1], a software product line is defined as “*a set of software intensive systems sharing common, managed set of features that satisfy specific needs of a particular market segment and that are developed from a common set of core assets in a prescribed way*”. However, effectiveness of a software product line does not solely depend on reuse capability but also on how *commonalities* and *variabilities* of a product line are managed and modeled from the requirements phase to the implementation phase.

Use cases are widely used to model functional requirements in traditional as well as product line systems. A use case specification captures stakeholders concerns as interactions between a system and its actors. Various extensions to traditional use case modeling have been proposed for the expression of variability and commonality. For instance, Ecklund *et al.* [2] proposed *change case* to specify anticipated changes that may impact a software product line. Change cases provide an “*impact link*” that creates traceability to use cases whose imple-

mentations might be affected. In [3], Jacobson *et al.* proposed *variation points* and *abstract use cases* to model variabilities and commonalities with the UML “*extend*” and “*generalization*” relations. Whereas, Gomma [4], introduced UML stereotypes “*kernel*”, “*alternatives*” and “*optional*” to distinguish common and variable use case specifications in software product lines. Similarly, John and Muthig [5] proposed stereotype “*variant*” and the marking of sections of use case diagrams as optional to model variabilities. As for use case descriptions, they advocate using XML tags ‘*variant*’ ‘*/variant*’ to mark optional and alternative steps (and scenarios).

In our previous work [6], we proposed an approach to support use case based requirements engineering. This approach is supported by a tool called *Use Case Editor, (UCEd)* [7]. UCED is a use case modeling tool that takes a set of related use cases written in a restricted natural language and automatically generate executable State Charts that integrates the partial behaviours defined by these use cases. A domain model is used for syntactical analysis of use cases and as a basis for state model generation from the use cases. We then extended our approach to support modelling aspects in use case specifications [8]. We introduced an “*aspect*” relation for crosscutting requirements and derived a composition mechanism for the generation of a global behaviour

model integrating use cases with crosscutting concerns. In this paper, we apply this *aspect-oriented use case modeling* approach to product line systems specification. A number of recent works have demonstrated that applying *Aspect-Oriented Software Development (AOSD)* to SPL provides an improved mechanism to encapsulate and model variabilities and commonalities throughout the entire software lifecycle [9–11]. We model variabilities and crosscutting commonalities as use cases and link them with a “*variability*” relation. The “*variability*” relation is a *specialization* of the “*aspect*” relation. The approach allows variabilities and commonalities to be better encapsulated and modularized. An aspect composition mechanism enables building of executable behaviour models for each member of a product line family by integrating common elements with the applicable variable elements.

The remainder of this paper is organized as follows. Section 2 presents some background material on use case modeling and presents our approach on modeling the concerns with “*aspect*” relation. In section 3, we present our argument on modeling variability and commonalities in a product line by adapting the approach presented in section 2 and describe variability compositions in terms of Petri net formalism. We then position our work relatively to close works in Section 4 and finally, section 5 concludes the paper and discusses some future works.

## 2. Use Case Modeling

A use case is defined as “*the specification of a set of actions performed by a system (or subsystem), which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system.*” [12]. In this section, we briefly review the UML use case relationships and our “*aspect*” relationship introduced in [8] for crosscutting concerns.

### 2.1 UML Use Case Modeling

A UML use case model includes use cases, actors and relationships. There are three types of relationships between use cases – *include*, *extend* and *generalization*. An include relationship  $uc_{base} \times uc_{inc}$  represents the inclusion of use case  $uc_{inc}$  as a sub-process of use case  $uc_{base}$  (*base use case*). An extend relationship  $uc_{ext} \times econd \times epoints \times uc_{base}$  represents an extension of a base use case,  $uc_{base}$  by an extension use case,  $uc_{ext}$ . Behaviors described in the extension use case are included at specific places in the base use case called *extension points*, *epoints*. Each extension is realized under a specific condition known as, *econd*. Whereas, a generalization relationship defines an inheritance relation between an abstract use case and a

\*Operations are specified as active sentences and conditions are in the form of predicative sentences. Further reference to the semi-formal natural language can be found in [6, 13].

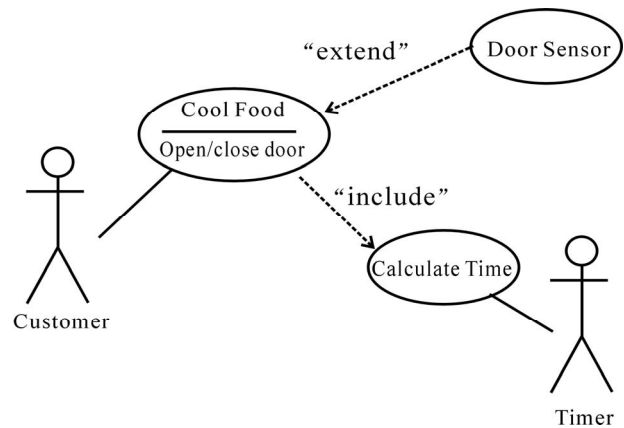


Figure 1. UML use case diagram for a microwave oven

more specific use cases. Figure 1 shows a UML use case diagram for a Microwave System (single system).

Use case diagrams represent abstract overview of a system. Each use case is specified in the form of description of interactions (as natural language text) between a user and the system. In order to support automated synthesis of state models from use cases, we formalized use case description by defining an abstract syntax, a concrete syntax based on natural language and by providing Petri nets based semantics to use cases [13].

Figure 2 shows an example of a use case. A semi-formal natural language\* is used for *operations* and *conditions* in use case steps. The UCed tool uses a domain model where domain entities including operations are defined to validate the use cases [6]. The domain model serves as a high-level class model that captures domain concepts and their relationships. Use case execution semantics are expressed using the Petri nets formalism [14] and an algorithm described in [13] generates Petri nets from use cases as an intermediate model for UML State Charts.

### 2.2 Aspect-Oriented Use Case Modeling

Aspects-oriented software development aims at providing software developers techniques and tools to better manage crosscutting concerns. A crosscutting concern is typically scattered among several other concerns. Crosscutting concerns needs to be identified and effectively handled from the beginning of the development lifecycle (i.e. requirements engineering). Jacobson and Ng [15] noticed a close relation between use cases and aspects as each use case typically crosscuts a set of components and usually involves crosscutting concerns such as synchronization, accuracy, access control and more. Other approaches that were proposed for aspect-oriented modeling of use cases include [16,17]. In our *aspect-oriented use case modeling* approach, crosscutting concerns defined as advice use cases are linked to affected concerns using an “*aspect*” relationship [8]. Differently to the UML relationships, the “*aspect*” relationship cardinality

is *one to many*. We defined *AspectJ* [18] constructs in use case terms and adopted a *symmetric* model where all concerns (including crosscutting concerns) may be extended as opposed to *AspectJ asymmetric* model.

Formally, an “*aspect*” relation defined as  $uc_{adv} \times acond \times apcuts \times baseUCs$  links the advice use case,  $uc_{adv}$  to the set of base use cases  $baseUCs$  according to pointcut expressions  $apcuts$  when condition  $acond$  is fulfilled. As mentioned previously, the cardinality of “*aspect*” relation is *one to many* since crosscutting concerns typically influences several use cases. The set of an “*aspect*” relationship target use cases is identified using parameterization based on use case description elements such as name, title, primary actors, goals, post-conditions etc. Advice use cases that capture crosscutting requirements are defined in the same form as normal use cases. However, we do not require that advice use cases strictly adhere to use cases *well-formedness* rules as stated in the UML specification [12]. An advice use cases may only have some of a use case sections. We also allow advice use cases to be initiated by the system and to describe incomplete interactions. The linking of advice use cases with affected base use cases is based on syntactical matching of *joinpoints* (potential occurrence of a crosscutting concern in a base use case) and *pointcut expressions*. Any use case description element i.e. *steps, operations, alternatives, extension points*, etc is a possible joinpoint. Pointcuts are parameterized pattern-based expressions that match joinpoints.

Additionally, a pointcut specifies *how* advice use cases are weaved at the joinpoints. Parameterization is essential in pointcut expressions since the number of target joinpoints can be large, complex and unpredictable. For instance a pointcut specified as “step 1, 2” refers to step one and two of the use case and “step \*” refers to all the steps in the use case. Similarly, pointcut “operations Microwave System \*” refers to all operations of entity “Microwave System” and operations “\*opens\*” matches all operations that contains the word “open” as part of the operation name. Three types of advice weaving are traditionally defined in AOP: *before, after* and *around*. We consider the same composition types with an additional type for concurrent composition (*concurrent*). Below is a brief description of each composition type:

- **before**: crosscutting requirements are applied before a joinpoint
- **after**: crosscutting requirements are applied after a joinpoint
- **around**: crosscutting requirements are applied instead of a particular joinpoint (*wrapping*)
- **concurrent**: crosscutting requirements are applied concurrently with a joinpoint (*in parallel*)

Figure 3 shows UCed representation of the Microwave Oven use case model with <<*aspect*>> relations.

The model includes one advice use case “Safety”. The target use cases of the <<*aspect*>> relation from use case “Safety” are specified as a wildcard “\*”. “Safety” use case is therefore applied to the all other use cases in the model but itself (an advice use case is always excluded from its set of related use cases to avoid infinite inclusion). The “Safety” advice use case is weaved based on operation pointcut “**after operations \* presses Start \***” (defined as a children of the relation).

Finally, we defined aspect composition at the Petri net level [8]. Behaviours defined in advice use cases are weaved with the affected base use cases according to pointcut expressions. A global behaviour model is obtained by integrating all crosscutting concerns and base use cases. Resulting Petri net models may be transformed

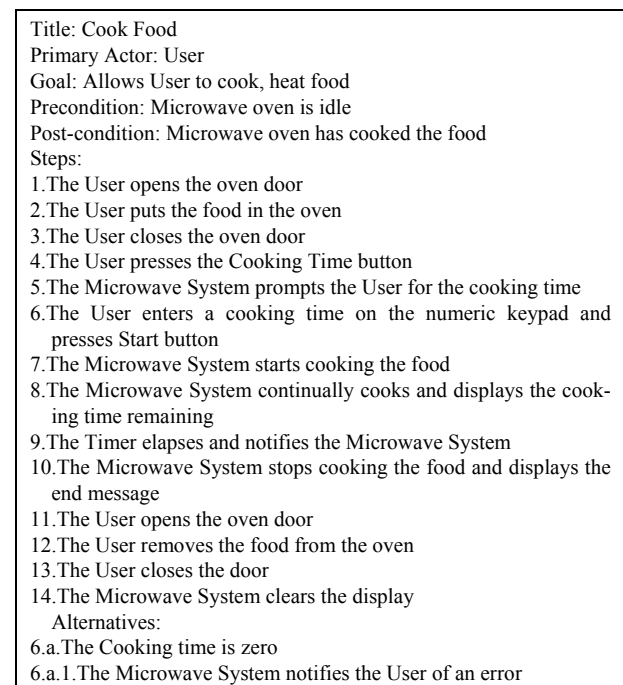


Figure 2. Details of use cases “Cook Food”

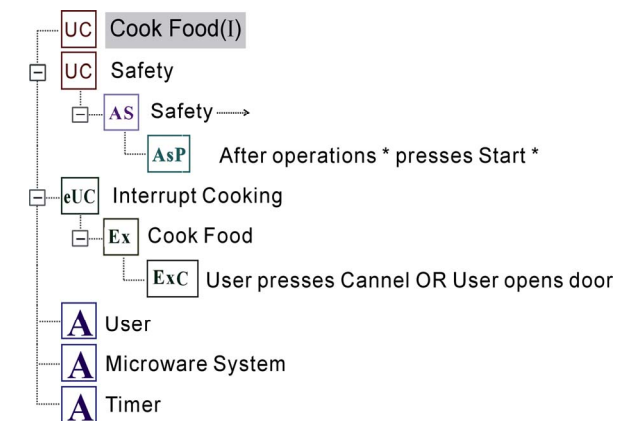


Figure 3. UCed representation use cases with “*aspect*” relations

to UML State Charts and used as prototypes for requirements validation by simulation in UCed [13].

### 3. Adapting Aspect-Oriented Use Case Modeling to Product Lines

A product line is a set of products that share a common set of characteristics and yet differ from each other based on a set of variabilities. *Product Line Engineering (PLE)* is about exploiting commonalities across product line systems while managing variabilities in order to improve reusability (of software artifacts such as requirements, models, components etc.), reduce time to market, cost and improve product quality. Commonalities are features that are common to a set of products and variabilities are features that products may optionally have in a product family. Variabilities influence software systems in the similar manner as crosscutting concerns [9–11,19]. For instance, variability “*x*” may be implemented several times and over different products across a product family. According to [19], aspects and variability are orthogonal concepts, which are independent of the core system and can be combined with it when needed.

In this section, we adapt our aspect-oriented use case modeling approach [8] to variabilities and commonalities. We introduce a “*variability*” relation to model variabilities and crosscutting commonalities in a product line system. The “*variability*” relation is a *specialization* of the “*aspect*” relation. It has similar characteristics as “*aspect*” relation but with some differences. Similar to crosscutting concerns, variabilities and crosscutting commonalities are specified as advice use cases and weaved into base use cases according to the different types of pointcut expressions. Our objectives in adapting aspect-oriented use case modeling to product lines specification include better encapsulation and modularity of variabilities and commonalities. Aspect-orientation allows variabilities and crosscutting commonalities to be modeled separately thus, improves readability as well as system evolution. By implementing the approach as part of the UCed toolset, we also aim at providing traceability between products and features and take advantage of UCed simulation capabilities for product design and validation.

We illustrate our approach with a product-line version of the microwave system [4]. The product line consists of microwave ovens that come with features ranging from basic to advanced features. A basic microwave oven system has input buttons for selecting Cooking Time, Start, and Cancel, as well as a numeric keypad. It also has a display to show remaining cooking time. Additionally, the oven has a microwave-heating element for cooking food, a weight sensor to detect if there is an object in the oven. Optional features for more-advanced microwaves include a beeper to indicate when cooking are done, a light that is switched on when the door is open and when

food is being cooked, and a turntable.

#### 3.1 “Variability” Relation

The “*variability*” relation is a “*specialization*” of “*aspect*” relation for modeling variabilities and crosscutting commonalities in a product line system. The “*variability*” relation links variabilities and crosscutting commonalities to affected base use cases and specifies pointcuts. More formally, a “*variability*” relationship  $uc_{var} \times plcond \times vpcuts \times baseUCs$  specifies that variabilities defined as an advice use case  $uc_{var}$  are weaved to the set of base use cases  $baseUCs$  according to pointcut expressions  $vpcuts$  when condition  $plcond$  is fulfilled. Condition,  $plcond$  specifies whether a given member of the product line provides the functionality described by the advice use case  $uc_{var}$ . Thus, the functionality is provided when  $plcond$  is *true*; whereas if the condition is *false*, then functionality is not provided. Similar to the “*aspect*” relation, the cardinality of the “*variability*” relation is *one to many*. This is a reflection of the fact that variabilities and crosscutting commonalities affect several use cases in a product line model. The *one-to-many* cardinality allows several base use case to be conveniently linked to a single variability. The set of target use cases is identified using parameterization in a similar way as the “*aspect*” relation. For instance, the targets of an “*variability*” relation specified as “\*cook\* are all use cases in the use case model which names contains the word “cook”, i.e. “Cook Food” and “Interrupt Cooking” in the use case model in Figure 1. Parameterization is not limited to use case names but may also be used for description elements such as “*primary actors*”, “*goals*”, “*pre-conditions*” and “*post-conditions*”. For instance, “Primary Actor User” matches all use cases with “User” as the primary actor. The parameterization allows changes to use case models and product evolution independently of variabilities, e.g. when a new variability option is added to a product family, changes to the existing model can be avoided. Furthermore, UCed “tree” representation (refer to Figure 7) helps model variabilities and commonalities within a software product family in a clearer manner.

#### 3.2 Composition Mechanism

Similar to crosscutting concerns, we define composition mechanism for product lines at the Petri net level, where advice use cases (variabilities) are weaved with affected base use cases based on pointcut expressions. Differently to composition mechanism for crosscutting concerns, where a global behavior model from use cases integrating all independently defined concerns is generated, we generate a Petri net for a particular product within a product line.

A Petri net is a triple  $[P, T, F]$  with:  $P$  a finite set of *places*,  $T$  a finite set of *transitions* and  $F \subseteq (P \times T) \cup (T \times P)$

a *flow relation*. Places are represented in the graphical description of a Petri net, as circles, transitions as boxes and the flow relation corresponds to arrows. Figure 4 shows how advice use cases (variabilities) are weaved according to the different types of pointcuts. Use cases are mapped to Petri nets such that each reference to use case description elements (*operations, step, etc*) corresponds to a *transition* in the Petri Net model [13]. To ease our discussion, we model the composition mechanism based on pointcut expressions formed with operation joinpoints (*op*). In Figure 4,  $a_1, a_2, a_3 \dots$  are sequence of events in the advice use cases (variabilities) and transition corresponding to the operation joinpoint, *op* in the Petri net derived from the base use case is colored black. Notice that differently to the general situation in [8], “*variability*” conditions (*plcond*) are not included in the composition for variabilities. Only relevant advice sequences are weaved at the corresponding operations joinpoints. Similar composition mechanisms are used for other types of joinpoints (i.e. *steps, alternatives, extension points*); with transition *op* replaced by the corresponding elements.

Several advice use cases (variabilities) can refer to the same joinpoint. In that case, the advice use cases are sequentially ordered based on the type of pointcuts. *Before* advices are applied first and then *after* advices. The behaviour described in an *around* advices substitutes the behaviour described by a specific operation or step in the base use case based on the “*variability*” relations condition. Lastly, *concurrent* advices are executed in parallel with joinpoints and *around* advices. Several pointcuts of the same type are applied in a *non-deterministic* manner. For instance, in a situation where several before pointcuts refer to a same joinpoint, the corresponding advice use cases would be randomly weaved one after the other and before the joinpoint. Although, this is not an optimal resolution technique, UCED allows the modeler to validate and simulate resulting Petri nets to uncover any

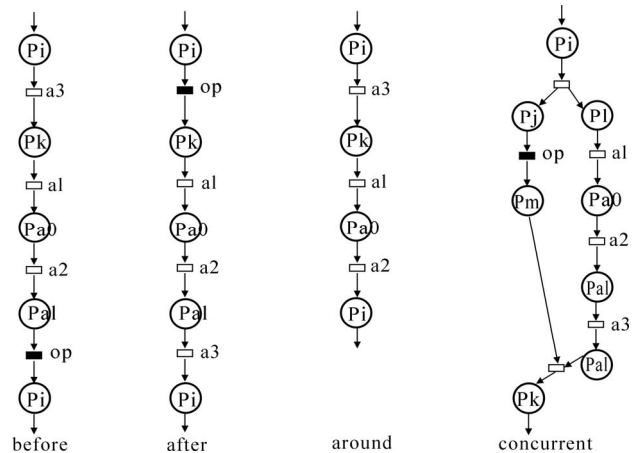


Figure 4. Weaving based on different types of operation, *op* inconsistencies that may occur and correct them accordingly.

### 3.3 Modeling Variability in UCED

Use case “Cook Food” in Figure 2 captures a common functionality of products in the microwave oven product line. We use feature modeling [20] to represent the high-level view of the Microwave Oven product family. *Optional* and *alternative* features describe variabilities in a product line and determine the characteristics of a given member of the product family. Optional features are variabilities that are required by some but not all members of a product family and alternative features are variabilities that are in different versions and are required by different members of the product line. These alternative variabilities are usually mutually exclusive. Figure 5 shows variabilities of the microwave oven product line as a feature model. Each microwave oven may include mandatory alternative and optional variabilities. The mandatory alternative variabilities are a *weight sensor* that is either Boolean or analog, a *display unit* that is

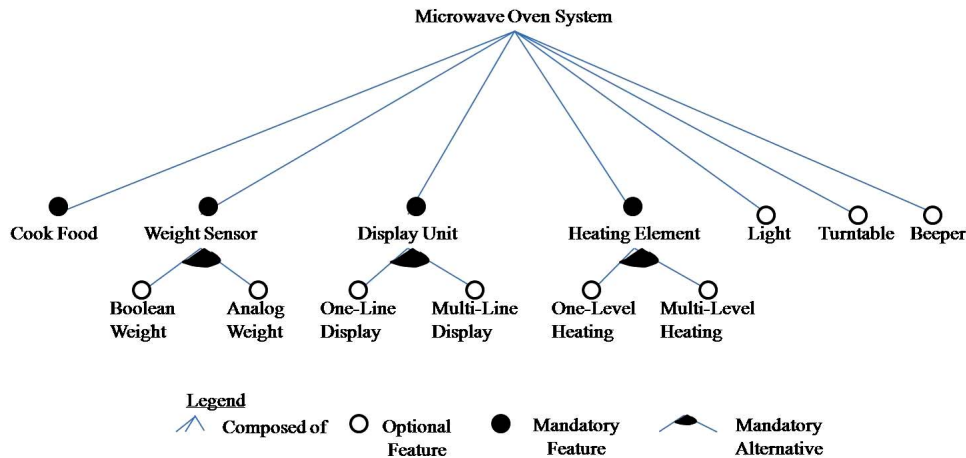


Figure 5. Feature model for microwave oven product line

one-line or multi-line, a one-level or multi-level *heating element*. The optional variabilities are a light, a turntable and a beeper.

We model variabilities as advice use cases and link them to commonalities with “*variability*” relations. As crosscutting concerns, variabilities are attached with conditions and pointcut expressions.

The attached condition specifies whether a given member of the product line provides the functionality described by the particular advice use case. Figure 7 shows UCED representation of the commonalities and variabilities of the Microwave product line system. Use case “Cook Food” represents commonalities while, variabilities are attached with the “*variability*” relation and are linked to all other use cases using wildcards ‘\*’. Each variable option is attached with a “*variability*” relation condition stating that the option is selected and a pointcut expression that specifies where the variable option is weaved. For instance, consider variability “One-Line Display” in Figure 7. The variability may affect all the other use cases in the model when condition “One Line Display option is selected” holds and it is weaved *before* step 1 of affected use cases. There can be multiple pointcut expressions attached to a single “*variability*” relation. For instance, in Figure 6 the “Light” option is weaved according to pointcuts “*concurrent operations User opens \**” and “*concurrent operations Microwave System starts \**”.

UCED uses a “tree” representation for use case models such that properties attached to a relation appears as children of that relation. This results in a representation where, variations in the product line are clearly distinguishable and identifiable. UCED allows all variabilities in a product family to be modeled at the same time. Specific variabilities can be selected to form distinct members of the product family during composition.



Figure 6. UCED use case integration tool

### 3.4 Product Generation

In order to generate a Petri net for a particular product within a product line, we need to specify which set of options apply. UCED includes an integration tool to enable the selection of relevant features. Figure 6, illustrates the UCED use case integration tool. Use cases listed in the integration tool are populated based on the “*variability*” relationship. Variable options that are selected are weaved with the common options and the corresponding Petri nets are generated. For instance, Table 1 shows the different feature sets for microwave oven models A and B.

Figure 6 shows the feature set selection for Model A in UCED. The model consists of a Boolean weight sensor, one-line display panel, one-level heating element and a beeper as an optional feature. Figure 10 shows the advice use cases (variabilities) for model A and B.

Figure 8 shows the Petri net obtained from the composition of the features in Model A while Figure 9 shows the Petri net obtained from the composition of the features in Model B.

Notice that <<*variability*>> conditions are not included in the generated Petri nets. Disabled variabilities are simply ignored from the resulting model. This allows the derivation of Petri nets specific to each member of a product family.

Table 2 shows how the different composition types are used for variabilities. The weaving of variability behaviors occurs *after*, *before*, *around* or *concurrent* to use case description elements such as operations, steps, extension points etc. *Optional* and *alternative* variabilities are weaved with *after*, *before*, or *concurrent* composition types. For instance, in Figure 7, the “Boolean Weight” variability is weaved *after operations User puts \** and the “One-Line Display” option is weaved *before step 1* of use case “Cook Food”. While, the *around* type can be used to substitute behaviors in commonalities with new behaviors. For instance, suppose that step 8 in use case “Cook Food” is expressed as follows: “*The Microwave System continually cooks and displays the cooking time remaining with a one-line display panel*”. An *around* composition type can be used to replace the one-line display panel to with multi-line display panel. We consider the concurrent composition as an essential composition type for use case modeling and to model variabilities and commonalities within a software product family (refer section 3.2) since it allows several operations (or use cases) to execute in parallel.

Table 1. Feature set for microwave ovens Model A and B

Model A	Model B
Boolean Weight	Boolean Weight
One-Line Display	Multi-Line Display
One-Level Heating	One-Level Heating
Beeper	Turntable
	Beeper

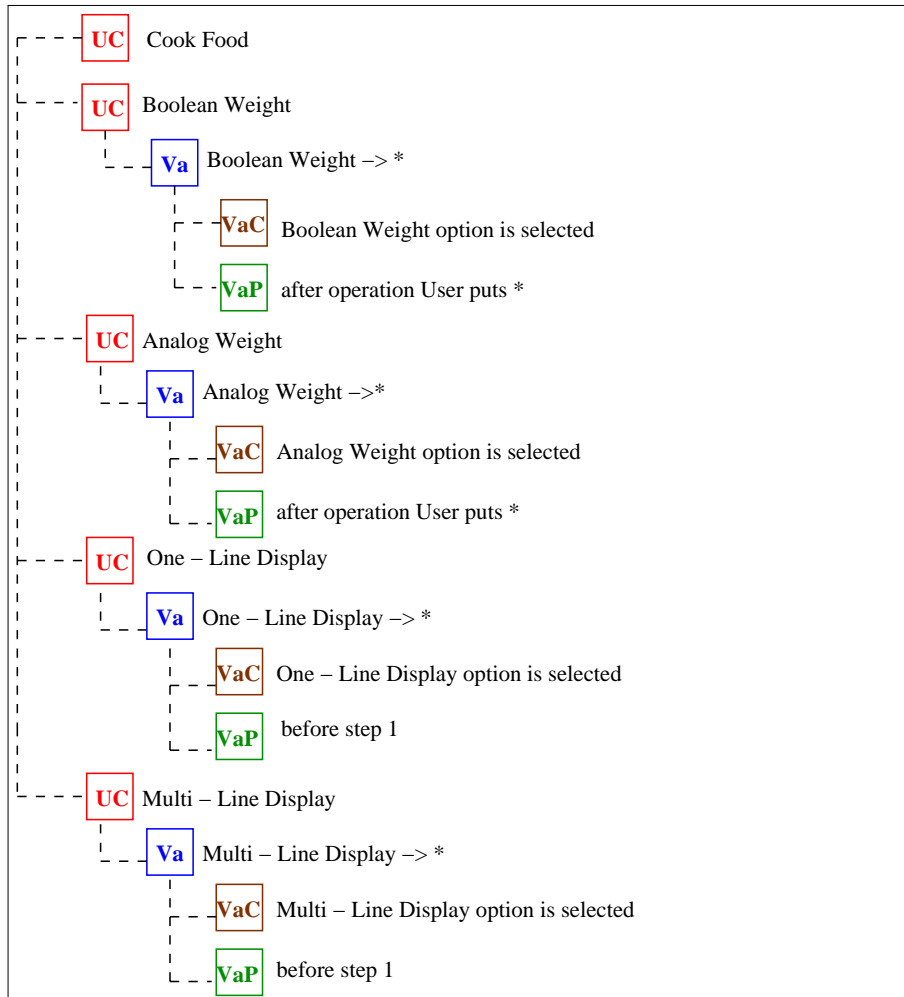


Figure 7. UCed representation of variabilities with "variability" relations

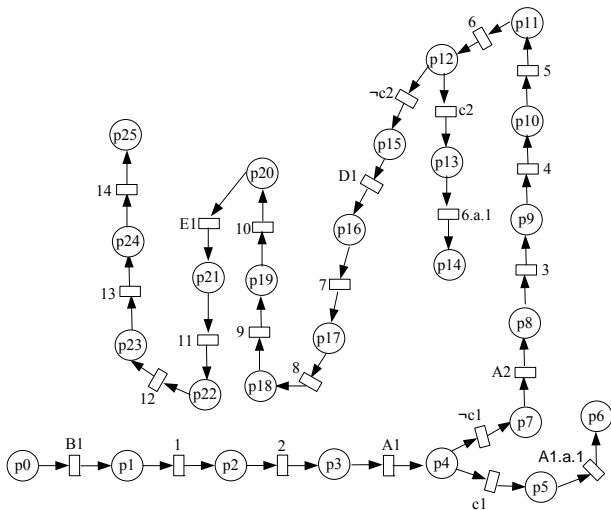


Figure 8. Petri net model for microwave oven Model A. We use corresponding step numbers as labels for transitions.  $c_1$  is condition "No item is present" and  $c_2$  is condition "Cooking time is zero"

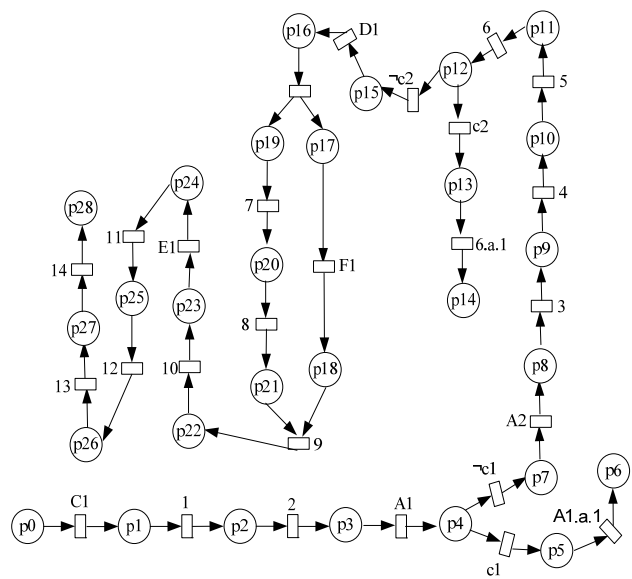


Figure 9. Petri net model for microwave oven Model B

<p>Title: Boolean Weight                  Goal: To check that item is present in the oven                  Steps:                  A1.The Microwave System checks for items in the oven                  A2.IF an item is present THEN the Weight Sensor indicates to the Microwave System that an item is present                  Alternatives:                  A1. a.No item is present                  A1. a.1.The Weight Sensor indicates to the Microwave System that no item is present</p>
<p>Title: One-Line Display                  Goal: To display messages and cooking time                  Steps:                  B1.The Microwave System enables one-line display panel</p>
<p>Title: Multi-Line Display                  Goal: To display messages and cooking time                  Steps:                  C1.The Microwave System enables multi-line display panel</p>
<p>Title: One-Level Heating                  Goal: To heat food                  Steps:                  D1.The Microwave System enables one-level heating capability</p>
<p>Title: Beeper                  Goal: To indicate user when cooking stops                  Steps:                  E1.The Microwave System activates the beeper when cooking stops</p>
<p>Title: Turntable                  Goal: To rotate food while cooking in progress                  Steps:                  F1.The Microwave System rotates for the duration of cooking</p>

**Figure 10. Advice use cases “Boolean Weight”, “One-Line Display”, “Multi-Line Display”, “One-Level Heating”, “Beeper” and “Turntable”**

**Table 2. Composition types**

Compositions	
<i>After</i>	Weave behavior after a use case description element
<i>Before</i>	Weave behavior before a use case description element
<i>Concurrent</i>	Weave variable behavior in parallel with a use case description element
<i>Around</i>	Substitute a use case description element with variable behavior

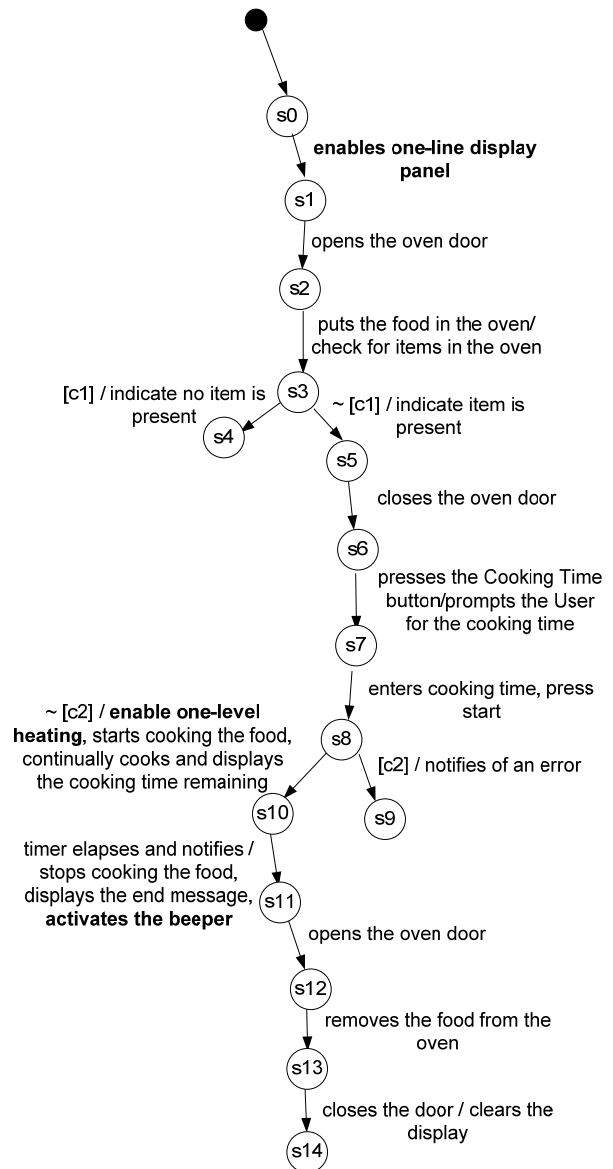
Some inconsistencies that may result from the combination of variabilities and commonalities can be identified using Petri nets analysis techniques integrated to UCed [8]. UCed also implements an algorithm for State Charts generation from Petri nets [13].

Figure 11 illustrates the UCed generated State Chart for Model A with the weaved variabilities in bold.

UCed allows simulation of the resulting State Charts as prototypes. It is thus possible to validate specific product member’s characteristics within a product family. Furthermore, test cases can be derived for early validations of specific member of a product family.

#### 4. Related Work

In [4], Gomma presents a UML-based software design



**Figure 11. State chart for microwave oven Model A**

method for software product lines called PLUS (Product Line UML-Based Software Engineering). The PLUS method extends the UML-based modeling approaches that are usually used for single systems to address software product lines. The objective of PLUS is to explicitly model commonalities and variabilities in a software product line. The method includes three stages: software product line requirements modeling, software product line analysis modeling and software product line design modeling. Firstly, during requirements modeling, kernel, optional and alternative use cases are developed to define the functional requirements of a system. The approach introduces UML stereotypes “kernel”, “alternatives” and “optional” to distinguish common and variable use case specifications in software product lines. Kernel use



cases describe functionalities that are common to all members of a software product line, whereas optional and alternative use cases describe variable functionalities which are specific to only certain members of a product line family. The approach also uses variation points to describe the location in a use case where a change can take place. Variation points are defined in a variation point section within a use case. Each identifies a line number where variability can be introduced or a conditional “*extend*” or “*include*” relationship. A feature model is then developed to capture the product line commonalities and variabilities and to describe how they relate to the use case models. During the analysis modeling, while static models are developed for defining kernel, optional and variant classes and their relationships, dynamic models which include state charts and interaction models are also developed. The state charts define the state dependent aspect of a product line and the interaction models describe the dynamic interaction between the objects that participate in each kernel, optional and alternative use case. Finally, during the design modeling stage, the component-based software architecture for the product line is developed. A similarity between our approach and the PLUS method is that use cases serve to capture requirements for product line systems. However, there are several differences between our work and the PLUS method. Our scope is limited to requirement specification while PLUS aims to address the whole development cycle. Another distinction between our work and PLUS concerns the way variation points are specified within variable use cases. Variation points are explicitly expressed in [4] and traditional use case relationships (“*extend*” and “*include*”) are used to link variabilities with commonalities. This creates a strong dependency between variable and kernel (base) use cases which limit flexibility and modularity. A similar approach, where variation points are explicitly specified in base use cases and “*extend*” or “*generalization*” relations are used to link variabilities, is used in [3]. By modeling variabilities as *aspects* and by using parameterized pattern based pointcuts, we avoid the dependency problem and allow the independent modeling of variabilities and commonalities. Unlike [3,4], we also propose a composition mechanism that allows the derivation of an executable behaviour model for each member of a product line family automatically integrating common elements with the applicable variable elements.

John and Muthig [5] describe how commonality and variabilities can be integrated and described in use case diagrams and textual use case descriptions. The approach allows modeling of variabilities by introducing a new type of use case stereotyped “*variant*”. Entire sections of use case diagrams may also be marked as variable and XML tags ‘*variant*’ ‘*/variant*’ may be used to mark optional and alternative steps (and scenarios) in use case

descriptions. The approach in [5] is very similar to Gomma's approach [4]; in that both extend the traditional use case modeling approach to accommodate modeling of commonality and variability in software product lines. The approach illustrates how a particular single-system use case can be extended to capture product line information, especially variabilities. However, while defining variabilities with a use case stereotyped as “*variant*” is an effective way to separate variable behaviours from the common behaviours in a product line, it forces requirements engineers to again adapt to a new type of use case. Furthermore, using XML tags ‘*variant*’ ‘*/variant*’ to mark optional and alternative steps (and scenarios) within use case specifications causes significant cluttering and thereby reduces readability. In our approach, variabilities and commonalities are modeled as normal use cases. We introduce a “*variability*” relation that links the selected variabilities to the base use cases based on pointcut expressions. The application of aspect-orientation allows variabilities and commonalities to be separately modeled and thus improves readability and maintainability. An additional advantage of our approach over [5] is that it allows the automated derivation of an executable specification for each product starting from textual use cases.

Our approach can also be seen as related to approaches that focus on the application of aspects to product lines modeling in general. There has been a significant interest in the AOSD community emphasizing the relations between variability and commonality in a software product line with *aspects* at the requirements engineering stage. *Aspect-Oriented Software Product Lines (AOSPL)* [21] is part of AOSD which focuses on *early aspects* in product lines. Various approaches suggest that variabilities in software product lines be modeled in the same manner as *crosscutting concerns*. For example Saleh and Goma [10] suggest grouping *optional* and *alternative* source code based on features in a variable source code file. This variable source file corresponds to an aspect file. Desired features can be selected and checked for consistency with a prototype tool which then are automatically integrated and compiled with the kernel source code to generate an executable member of a product line. Loughran *et al.* [9] use aspect-oriented techniques with natural language processing to facilitate requirements analysis and concern identification for the derivation of suitable feature-oriented models for implementation. This approach is implemented in a tool called NAPLES. The tool takes textual requirements, deduces concerns, aspects, feature commonalities and variabilities to ease implementation. Siy *et al.* [11] present an approach to separate functional requirements as *viewpoints* and non-functional requirements into aspects similar to ARCADE [22]. Their approach includes parameterization of requirements and a composition mechanism.

Our approach is similar to [9–11] by the application of AOSD to product lines. A major distinction is that we model crosscutting requirements and variabilities using use cases in textual form. Use cases enable the application of aspect-orientation early in the development life-cycle and thus, prevent crosscutting requirements from being overlooked.

## 5. Discussion & Conclusion

In this paper, we presented an approach for use case based modeling of software product line systems. We introduced a “*variability*” relation; a specialization of an “*aspect*” relation proposed for modeling crosscutting concerns [8]. Use case description elements are used as joinpoints and variable requirements are weaved with common requirements based on specified pointcut expressions. Selected variabilities and commonalities are composed and transformed into Petri net models as a step toward UML State Charts generation. The whole approach is automated and tool supported by adding extensions to UCed, an existing use case-modeling tool. UCed includes facilities for simulation of generated state charts and test generation. The simulation of a product model enables early validation of specific product members in the product family.

We noted some limitations to our proposed approach. For instance, some of the steps and operations required to be re-written in order to match them with appropriate pointcut expressions and ensure correct composition. For example, in our Microwave Oven example the “Analog Weight” advice use case is weaved *after operations User puts* \*. The modeler should verify the base use case to make sure that the “Analog Weight” advice is only weaved at the relevant places and there are no other operations that would match the specified pointcut. We also found that not all variabilities could be modeled easily. Examples of such variabilities are “One-Line Display Unit” and “Display Languages” [4]. These variabilities are more of initialization type. Therefore, multiple weaving at different locations is not appropriate. Thus, we cater a workaround in that; these variabilities are initialized before the execution of affected base use cases. For instance, “One-Line Display Unit” is weaved *before step 1* of use case “Cook Food”. Another potential problem is the scalability and complexity of resulting Petri net models when a great number of variability options are composed together for a specific product in a product family. This may make it harder to read and understand the resulting models and leave only simulation as validation approach.

In our future work, we plan to apply our approach to more case studies and investigate further ways of modeling variabilities and commonalities for software product lines in UCed. We also plan to explore other composition types (besides before, after, around and concurrent)

and define a more detailed joinpoint structure.

## REFERENCES

- [1] P. Clements and L. Northrop, “Software product lines: Practices and patterns,” Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
- [2] E. Ecklund, L. Delcambre, and M. Freiling, “Change cases-Use cases that identify future requirements,” Proceedings of OOPSLA 96, San Jose, Ca, pp. 342–358, October 1996.
- [3] I. Jacobson, M. Griss, and P. Jonsson, “Software reuse—Architecture,” Process and Organization for Business success, Addison-Wesley, 1997.
- [4] H. Gomaa, “Designing software product lines with UML—From use cases to pattern-based software architectures,” Addison-Wesley, 2004.
- [5] I. John, and D. Muthig, “Product line modeling with generic use cases,” SPLC-2 Workshop on Techniques for Exploiting Commonality Through Variability Management, Second Software Product Line Conference, San Diego, USA, August 2002.
- [6] S. Somé, “Supporting use cases based requirements engineering,” Information and Software Technology, Vol. 48(1), pp. 43–58, 2006.
- [7] Use Case Editor (UCed) toolset, <http://www.site.uott-awa.ca/~ssome/UseCaseEditorUCed.html>.
- [8] S. S. Somé and P. Anthonysamy, “An approach for aspect-oriented use case modeling,” Workshop on Early Aspects at ICSE 2008: Aspect-Oriented Requirements Engineering and Architecture Design, May 2008.
- [9] N. Loughran, A. Sampaio, and A. Rashid, “From requirements documents to feature models for aspect oriented product line implementation,” Workshop on MDD in Product Lines at MODELS, 2005.
- [10] M. Saleh and H. Gomaa, “Separation of concerns in software product line engineering,” In Proceedings of the 2005 Workshop on Modeling and Analysis of concerns in Software. St. Louis, Missouri, 2005.
- [11] H. Siy, P. Aryal, V. Winter, and M. Zand, “Aspectual support for specifying requirements in software product Lines,” Workshop on Early Aspects at ICSE. Minneapolis, USA, May 2007.
- [12] OMG: UML 2.1.2 Superstructure Object Management Group, 2007.
- [13] S. S. Somé, “Petri nets based formalization of textual use cases,” Technical Report TR-2007-11, SITE, University of Ottawa, 2007.
- [14] C. A. Petri, “Communication with Automata,” PhD thesis, Technische Universität Darmstadt, 1962.
- [15] I. Jacobson and P. W. Ng, “Aspect-oriented software development with use cases,” Addison Wesley, 2005.
- [16] J. Araújo and A. M. D. Moreira, “An aspectual use case driven approach,” In VIII Jornadas Ingeniería del Software y Bases de Datos (JISBD 2003), pp. 463–468, 2003.

- [17] R. Chitchyan, A. Rashid, P. Rayson, and R. Waters, "Semantics-based composition for aspect-oriented requirements engineering," In: AOSD '07: Proceedings of the 6<sup>th</sup> International conference on Aspect-oriented software development, ACM, pp. 36–48, 2007.
- [18] AspectJ Project, <http://www.eclipse.org/aspectj/>.
- [19] R. Stoiber, S. Meier, and M. Glinz, "Visualizing product line domain variability by aspect-oriented modeling," Proceedings of the 2<sup>nd</sup> International Workshop on Requirements Engineering Visualization (REV'07), in conjunction with RE'07. New Delhi, India.
- [20] M. Griss, J. Favaro, and M. d'Alessandro, "Integrating feature modeling with the RSEB," Proceedings of Fifth International Conference on Software Reuse, Victoria, B.C., 1998.
- [21] Aspects and Software Product Lines (EA-SPLC), <http://trese.cs.utwente.nl/workshops/early-aspects-SPLC2005/>.
- [22] A. Rashid, A. Moreira, and J. Araújo, "Modularization and composition of aspectual requirements," In Proc. of the 2<sup>nd</sup> International Conference on Aspect-Oriented Software Development (AOSD'03), Boston, MA, pp. 11–20, March 2003.