Innovations Syst Softw Eng (2012) 8:51–64
DOI 10.1007/s11334-011-0177-9

SI: MOMPES 2010

On the refinement of use case models with variability support

Sofia Azevedo · Ricardo J. Machado · Alexandre Bragança · Hugo Ribeiro

Received: 13 January 2011 / Accepted: 3 November 2011 / Published online: 11 December 2011

Abstract Modeling software product lines shall imply modeling from different perspectives with different modeling artifacts such as use case diagrams, component diagrams, class diagrams, activity diagrams, sequence diagrams and others. In this paper, we elaborate on use cases for modeling product lines and we explore them from the perspective of variability by working with the unified modeling language (UML) «*extend*» relationship. We also explore them from the perspective of detail by (functionally) refining use cases with «*extend*» relationships between them. This paper's intent is to provide for comprehension about use case modeling with functional refinement when variability is present.

Keywords Use case · Software product line · Refinement · Variability · «*extend*» · Alternative · Option · Specialization

1 Introduction

Use case diagrams are one of the modeling artifacts modelers have to deal with when developing product lines with model-driven approaches. A software product line can be faced as a family of software products that have been developed with explicit concern about variability (and consequently commonality) during the development process (see [1] for basic references on variability). This paper envisions use cases according to the perspectives of detail and variability. The «*extend*» relationship plays a vital role in variability modeling in the context of use cases and allows for the use case modeling activity to be applicable to the product line software development approach. This is possible by determining the locations in use case diagrams where variation will occur when instantiating the product line. This paper's contribution is on the formalization and understanding of the use case modeling activity with support for variability and functional refinement when variability is present. We will illustrate our approach with the Fraunhofer IESE's GoPhone case study [2], which presents a series of use cases for a part of a mobile phone product line particularly concerning the interaction between the user and the mobile phone software. We propose an extension to the unified modeling language (UML) metamodel [3] to formally provide for both the concrete and abstract syntaxes to represent different types of variability in use case diagrams. Throughout the paper, we refer to six different relationships. Some are from the UML, which explicitly uses the terminology «relationship »: the UML «*extend*» relationship, the UML «*include*» relationship and the UML generalization relationship. Some were introduced by us to the UML metamodel by extending it according to an extensive related work analysis: the (UML) «*refine*» relationship, the (UML) «*alternative*» relationship and the (UML) «*specialization*» relationship. We consider use cases in different abstraction levels to elaborate on the functional refinement of use cases with «*extend*»relationships between them. In this paper, we focus on the variability support as well as on the process point of view with regard to the use case modeling activity.

S. Azevedo (✉) · R. J. Machado
Universidade do Minho, Guimarães, Portugal
e-mail: sofia.azevedo@dsi.uminho.pt

R. J. Machado
e-mail: rmac@dsi.uminho.pt

A. Bragança
Instituto Superior de Engenharia do Porto, Porto, Portugal
e-mail: alex@dei.isep.ipp.pt

H. Ribeiro
Primavera Business Software Solutions, Braga, Portugal
e-mail: hugo.ribeiro@primaverabss.com

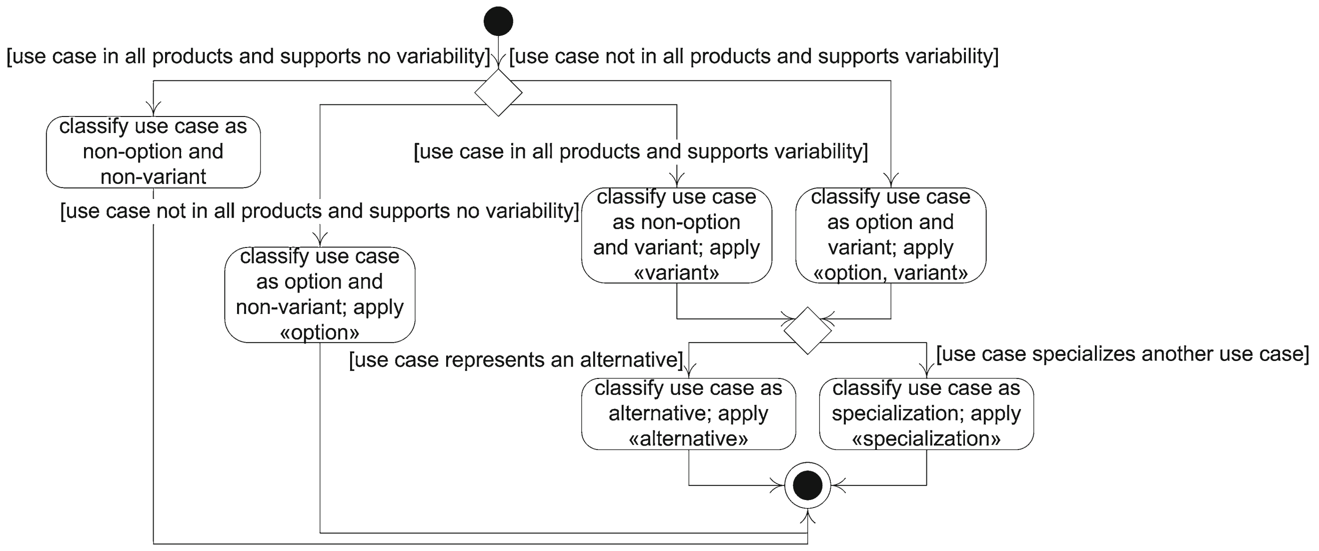

[use case in all products and supports no variability]
[use case not in all products and supports variability]
classify use case as non-option and non-variant
[use case in all products and supports variability]
[use case not in all products and supports no variability]
classify use case as non-option and variant; apply «variant»
classify use case as option and variant; apply «option, variant»
classify use case as option and non-variant; apply «option»
[use case represents an alternative]
[use case specializes another use case]
classify use case as alternative; apply «alternative»
classify use case as specialization; apply «specialization»

Fig. 1 The use case variability types

Modeling variability in use cases with the resources the UML makes available is a benefit of our approach because the UML is extensively used in the community (both academic and industrial) and is a widely recognized standard. We have systematized variability modeling for use cases according to a model with explicit decisions modelers may follow to apply our approach (Fig. 1). We have considered the refinement of use cases connected through «*extend*» relationships, which is pertinent in the context of large-scale product lines (see Sect. 6). Both the variability modeling in use cases and the functional decomposition of use cases are required at the requirements modeling stage to prepare the modeling artifacts for further handling in the product line development process (see Sect. 2). The complexity of the use case model in Fig. 10 may be considered as a limitation of the proposed approach but we present in Sect. 6 some ways of decreasing that complexity.

We define the functional decomposition of a use case as the decomposition of an initial use case diagram into smaller and more cohesive ones. Its goal is to decrease the complexity of the use case diagram and increase its cohesion. The advantage of functional decomposition of use cases from the process point of view when developing large software systems is allowing the delivery of less complex modeling artifacts to the teams implementing the software system. The advantage of functional decomposition of use cases from the process point of view when developing software systems with variability is the possibility of modeling later on an alternative to a part of the decomposed use case or modeling a part of the decomposed use case that is an optional part.

A lot of literature exists on variability modeling (some of it particularly related to requirements modeling). We analyzed an extensive and significant set of references on the subject and found the need to systematize the modeling of variability according to our position on the subject (according to our current and previous research). A set of stereotypes are the solution we concluded to be more adequate, efficient and effective for modeling variability in use cases. We also provide for comprehension about use case modeling with functional refinement (refer to the definition of functional decomposition in the paragraph above together with the notion of detailing the textual descriptions of use cases as the functional refinement of use cases) when variability is present, considering the systematization of the modeling of variability we undertook based on that set of references and our position on the subject.

The paper is structured as follows. Section 2 elaborates on the differences between others' approaches and this paper's approach. Section 3 elaborates on the different types of variability we propose to be used in the context of use case modeling. Section 4 provides for the analysis of the UML «*extend*» relationship in contexts of variability and also for the extension we propose to the UML metamodel to support the different variability types. Section 5 analyzes the process of handling variability in use case diagrams in the context of the functional refinement of use cases. Section 6 illustrates our approach with the GoPhone case study. Finally, Sect. 7 affords some concluding remarks.

2 Related work

Despite use cases being sometimes used as drafts during the process of developing software and not as modeling artifacts that actively contribute to the development of software, use cases shall have mechanisms to deal with variability for them to have the ability to actively contribute to the process of developing product lines. Consider that the variability types we propose in the context of use cases can be represented by option, alternative and specialization use cases. For instance, modeling variability in use case diagrams is important to later model variability in activity diagrams [4]: option use cases map to alternative insertions in activity diagrams (alternative insertion is a type of sequences of actions in the context of activity diagrams), and both alternative and specialization use cases map to alternative fragments in activity diagrams (alternative fragment is another type of sequences of actions in the context of activity diagrams). We do not elaborate further on this topic since it is out of the scope of this paper. Further in this paper we talk thoroughly about option, alternative and specialization use cases as the representation of the three different types of variability we consider in the context of we cases.

This paper's work is inspired on the approach of Bragança and Machado to variability modeling in use case diagrams [5]. Bragança and Machado represent variation points explicitly in use case diagrams through extension points. Their approach consists of commenting «*extend*» relationships with the name of the products from the product line on which the extension point shall be present. Their approach to product line modeling is bottom-up (rather than top-down), which means that all the product line's products are known *a priori*. A top-down approach would consider that the product line would support as many products as possible within the given domain. In [6,7], John and Muthig refer to required and anticipated variations as well as to a planned set of products for the product line, which indicates that their approach to product line modeling is bottom-up. The approach in this paper adopts the top-down approach for product line modeling, therefore discarding the comments to the «*extend*» relationships. In [8], Bayer et al. refer that all variants do not have to be anticipated when modeling the product line.

In [6], John and Muthig refer the benefits of representing variability in use cases, namely establishing a variability and product line mindset among all involved roles in a product line's engineering, supporting the derivation of models and instantiation in application engineering, and communicating the possibilities of the possible products to different stakeholders. Although we totally agree with the position of these authors towards the benefits of representing variability in use cases, we cannot agree when they state that information on whether certain use cases are optional or alternatives to other use cases shall only be in decision models as it would overload use case diagrams and make them less readable (decision models in this context are feature diagrams [9]). John and Muthig use one variability stereotype in use cases (the «*variant*» stereotype) applicable for variant use cases (use cases that are not supported by some products of the product line, whether optional or alternative). Our position is that features as well as use cases shall be suited for treating variability in its different types. If a use case is an alternative to another use case then both use cases shall be modeled in the use case diagram, otherwise the use case diagram will only show a part of the possibilities of the possible products as John and Muthig [6] mentioned. Bachmann et al. [10] mention that variability shall be introduced at different phases of the development of product families. Bühne et al. [11] propose a metamodel for representing variability in product lines based on the metamodel of Bachmann et al. [10] for representing variability in product lines.

Gomaa and Shin [12,13] analyze variability in different modeling views of product lines. They mention that the «*extend*» relationship models a variation of requirements through alternatives. They also model options in use case diagrams using the stereotype «*optional*» in use cases. We adopt these approaches to alternatives and options but we elaborate on another form of variability (specializations, which we consider to be a special kind of alternatives). Gomaa and Shin [12,13] refer specialization as a means to express variability. Besides alternative and optional use cases, Gomaa and Shin consider kernel use cases (use cases common to all product line members). Gomaa together with Olimpiew [14] talks again about kernel, optional and alternative use cases. Gomaa [15] models kernel and optional use cases both with the «*extend*» as well as with the «*include*» relationships (our approach is towards modeling kernel and optional use cases independently of their involvement in either «*extend*» or «*include*» relationships and with a stereotype in use cases). In [16], Webber and Gomaa propose the Variation Point Model to model variation points. In that context, the variation point shall be treated from four different views, one of which is the requirements variation point view. This view captures requirements together with variation points during the product line's domain analysis phase. Variation points are considered to be mandatory or optional (the difference between both is that mandatory variation points do not supply a default variant, whether optional ones do). In our approach to variability modeling, we consider more types of variability besides the optional one (alternative and specialization).

Bayer et al. [8] present the Consolidated Variability Metamodel. In that context, they systematize different kinds of variability recurrently present in product line models. The work of this paper is related to that systematization since it addresses some of those kinds of variability and realizes it in annotations to the UML applicable to some model elements to which the selected variability kinds apply. Ziadi et al. [17] expose a UML profile for software product lines, however they do not talk about stereotypes to be applied to requirements models.

Coplien et al. [18] defend the analysis of commonality and variability during the requirements analysis for the analysis decisions not to be taken during the implementation stage by the professionals who are not familiar with the implications and impact of decisions that shall be made much earlier during the development cycle. They refer that early decisions on commonality and variability contribute to large-scale reuse and the automated generation of family members.

Halmans and Pohl [19] propose use cases as the means to communicate variability relevant to the customer and they also propose extensions to use case diagrams to represent variability relevant to the customer. Halmans and Pohl consider that generalizations between use cases are adequate to represent use cases' variants. This is not our position. We recommend using the «*extend*» relationship instead of the generalization relationship. Although Halmans and Pohl consider that the «*extend*» relationship is suitable for modeling options to parts of the use cases to which those options refer, they do not recommend it because of not explicitly representing variation points (Halmans and Pohl consider that by not having the variation points explicitly represented in the use case diagrams, it is not documented if the customer can or must select one or more variants or if all of them are already present in the system, which violates the principle of communicating variability). They consider that modeling mandatory and optional use cases with stereotypes in use cases is not adequate because the same use case can be mandatory for one use case and optional for another. Again this is not our position. We also consider that a mandatory use case is not mandatory with regard to another use case, rather it is mandatory for all product line members. We also consider that an optional use case is optional with regard to one or more product line members. Halmans and Pohl end up by introducing additional graphical elements to use case diagrams to represent variation points and variability cardinality explicitly in use case diagrams. We do not agree with this approach since it introduces more complexity to use case diagrams than modeling variability with stereotypes and use case relationships as well as it introduces a reasoning about variability that should be present in decision models (the selection of the variants to be present in the system and the system/product to which that selection applies according to the features). Pohl [20] uses the graphical notation used by Halmans and himself [19] to represent variability in use case diagrams. Salicki and Farcet [21] talk about variation points and additionally in decision models.

Maßen and Lichter [22] talk about three types of variability: optional, alternative and optional alternative (as opposite to alternatives that represent a "1 from n choice", optional alternatives represent a "0 or 1 from n choice"). In this context, they propose to extend the UML metamodel to incorporate two new relationships for connecting use cases. Our approach considers options and alternatives as well but we introduce these concepts into the UML metamodel through stereotypes (we consider that the «*extend*» relationship is adequate for modeling alternatives and a stereotype applicable to use cases for modeling options).

According to Gomaa [15], and John and Muthig [6,7], use cases can be tagged with some stereotypes concerning variability. Table 1 shows the applicability of those stereotypes in our approach.

Table 1 Some use case stereotypes concerned with variability

Stereotype	Applicability
«kernel »	Use cases
«alternative»	«*extend*» relationships
«optional »	Use cases
«variant »	Use cases

Some examples of approaches to functional decomposition of software systems are the 4SRS (Four Step Rule Set) method [23], KobrA or RSEB (Reuse-Driven Software Engineering Business) [24,25]. However, neither KobrA nor RSEB clearly contemplates a technique for refining use cases like the 4SRS method does.

Greenfield and Short [26] refer to refinement as the inverse of abstraction or the process of turning a description more complex by adding information to it. They refer to the process of developing software through refinement as progressive refinement. The process starts with requirements and ends up with the more concrete description of the software (the executable). They consider refinement as a concatenation of interrelated transformations mapping a problem to a solution. The goal of refinement is to smoothly decrease the abstraction levels that separate the problem from the solution. In general terms Greenfield and Short talk about refinement as the stepwise decomposition of features' granularity. In the context of use cases, refinement is their detailing. However, we defend that use cases can themselves be refined to facilitate the transformation of a problem (which can be modeled with use cases) to a solution (which shall be modeled with design artifacts e.g. logical architectures).

Pons and Kutsche [27] present the refinement activity as a way to trace code back to system requirements and system requirements back to business goals, which allows verifying whether the code meets the business goals and the system requirements as expected in the specification of the system.

Eriksson et al. [28] treat refinement as a relation between features that are obtained from decomposing other features. Features at different levels of decomposition maintain relationships with use cases or parts of their textual descriptions. We consider that refining use cases includes their decomposition as well as features do. We also consider that use cases can be decomposed without being refined: in our approach refining use cases includes their detailing (adding detail to the description of use cases) besides their decomposition. For us to consider the relationship between features and use cases at different levels of abstraction, those different levels of abstraction shall be defined based on both decomposition and detailing.

Gomaa [15] explored refinement in the context of feature modeling, where a feature can be a refinement of another. But to get to the features, use cases have to be modeled and mapped to features. Our approach eliminates this mapping activity. To Gomaa the refinement is expressed through «*extend*» relationships in the context of use cases. To us, the refinement shall be expressed through the «*refine*» relationship we proposed in [29].

Refinement has been treated over the years. Paech and Rumpe [30] provide a formal approach to incrementally design static software system models through refinement. Our approach is not formal and relates to the refinement of external functionalities of software systems, which shall be taken into account before the static part of those systems. Quartel et al. [31] propose an approach for the refinement of actions. Again our approach relates to a perspective that shall be taken into account before behavior is. Darimont and van Lamsweerde [32] talk about goal refinement. This time our approach to refinement relates to a perspective that shall be taken into account after goals. Mikolajczak and Wang [33] present an approach to vertical conceptual modeling of concurrent systems through stepwise refinement using Petri net morphisms. Our approach to refinement is not formal. Batory created a model (the AHEAD model [34]) for expressing the refinement of system representations as equations. He worked at a code-oriented level. Our approach allows refining software models that shall be handled before code is handled during the software construction phase.

Cherfi et al. [35] (in their work on quality-based use case modeling with refinement) describe the refinement process as the application of a set of decomposition and restructuring rules to the initial use case diagram. Our approach is iterative and incremental. It consists of decomposing the initial use case diagram into smaller and more cohesive ones to decrease the complexity of the diagram and increase its cohesion. In the approach of Cherfi et al. to refinement, use cases are not actually detailed (like in ours), rather they are decomposed without detail being added to the description of those use cases.

Regarding use case semantics and notation, we analyzed the position of Simons [36] and of Heldal [37] on the topic. Simons argues that the insertion semantics of the «*extend*» relationship is inadequate to model alternatives. That is not our position (as we explain further on this paper) since the UML semantics supports our notion of variability (alternative is one type of variability that the «*extend*» relationship supports). Heldal worked on the extraction of system operations (calls into the system or communication between actors and the system) by structuring use cases through the grouping of action steps (e.g. sentences with the structure subject+verb+object) from use cases into action blocks. These action blocks allow writing contracts for the system operations (contracts are system operations with pre and post conditions). Heldal refers to event-driven systems where system operations make more sense than use cases. For every action block, a contract can be written. A use case has more than one action block. Input and output data shall be related to single action blocks and not to a single use case because a use case has more than one action block and a contract is written for a single action block. An incomplete use case contains only one action block. A complete use case has more than one action block and fulfills a goal for the actor(s). Heldal mentions that «*include*» and «*extend*» relationships do not refer to complete use cases on both ends of the relationship, therefore in his approach these use cases are contracts rather than a group of action blocks, which does not allow fulfilling (a) goal(s) for (an) actor(s). That is not our position. In our approach, use cases involved in «*include*» or «*extend*» relationships are still use cases that fulfill (a) goal(s) for (an) actor(s), representing external functionality of the system that can be performed by the actors (a use case still represents observable value to an actor, despite being more or less detailed, despite decomposing another use case or despite being an extension to another use case) [29].

Fowler made the following advice in his book "UML Distilled" [38]: "don't try to break down use cases into sub-use cases and subsub-use cases using functional decomposition. Such decomposition is a good way to waste a lot of time". We cannot agree with Fowler's opinion at a certain extent. The pertinence of functional decomposition lies on the scale of the software system under development. The development of large software systems benefits from decomposing the functionality of those systems to a level that allows delivering less complex modeling artifacts to the teams implementing the software system. All the more large software systems are frequently built from a series of components developed by different teams. A single team is not expected to develop the whole system, therefore it shall not be delivered the modeling artifacts concerning the whole system to guide the conception of the component that is required to be developed by that team [39]. Fowler made another suggestion in his book: "The UML includes other relationships between use cases beyond the simple includes, such as «extend »İ strongly suggest that you ignore them. I've seen too many situations in which teams can get terribly hung up on when to use different use case relationships, and such energy is wasted. Instead, concentrate on the textual description of a use case; that's where the real value of the technique lies". We completely agree with Fowler when he says that the value of use case modeling lies on the textual descriptions of use cases. Our approach to use case refinement is based on those descriptions. But we cannot agree when Fowler says that the relationships besides the «*include*» relationship shall be ignored when modeling use cases. The «*refine*» relationship cannot be ignored. It is needed to formalize at an early stage (the use case modeling) where functional decomposition shall happen to decrease the complexity of the modeling artifacts delivered to the different development teams. Also the «*extend*» relationship is needed to formalize at an early stage (the use case modeling) where variation will occur when instantiating the product line (Bosch et al. [40] mention the need for describing variability within different modeling levels such as the requirements one).

3 Handling variability in use case modeling

Figure 1 illustrates the variability types we consider and propose to be applicable in the context of use cases [1]. Use cases can be non-option or option. Non-option use cases are present in all product line members. Option use cases can be present in one product of the product line and not in another. It is not mandatory that option use cases are present in all products of the product line. Non-variant use cases are use cases that do not support variability. Variant use cases are use cases that support variability. This means that different products will support different alternatives for performing the same functionality or that different products will support different specializations of the same functionality. Later on during the modeling activity, variant use cases are realized into alternatives or specializations, respectively. Alternative use cases represent alternatives for performing the same system's use in different products or sets of products from the product line. A specialization use case is a use case that represents the specialization of another use case. Specialization use cases that specialize the same use case indeed represent alternatives to each other but they specialize a use case, which is not the case of alternative use cases. Option, alternative and specialization use cases are the representation of the three variability types that will be translated into stereotypes to be applicable to use cases. The use cases that do not represent options and are not variant (not later alternatives or specializations) are non-option and non-variant, and shall not be marked with any stereotype. Non-option and option use cases are different as well as non-variant and variant use cases. Figure 1 represents the activity of classifying use cases with variability types: either *non-option and non-variant* or *option and non-variant* or *non-option and variant* or *option and variant*. These last two variability types can be realized into the *alternative* or the *specialization* variability types (as already explained). The activity of classifying use cases with the variability types is important for applying the corresponding stereotypes to the use cases (except for the *non-option and non-variant* use cases, which shall not be marked with any stereotype). The conditions of the decision nodes express the semantics of each one of the variability types. We would like to give emphasis to a particular variability type: the *option and variant* variability type. This variability type is applicable to a use case that is not present in all product line members but the different members in which it is present support different alternatives for performing that use case's functionality or different specializations of that use case's functionality. *Option and non-variant* use cases shall be marked as option use cases; *non-option and variant* as variant use cases; and *option and variant* use cases as both option and variant use cases.

4 The «*extend*» relationship

For use cases to be appropriate for product line modeling, they have to be equipped with variability mechanisms. These variability mechanisms must allow determining the locations in diagrams (in this case use case diagrams) where variation will occur when instantiating the product line.

The «*extend*» relationship allows modeling alternative and specialization use cases in use case diagrams. Consider that an extending use case is a use case that extends another use case and that an extended use case is a use case that is extended by other use cases. As any other use case, an extending use case represents a given use of the system by a given actor or actors.

In the UML metamodel, the extending use cases are considered to represent supplementary behavioral increments that have to be inserted into the appropriate insertion points between the extended use case's fragments. These fragments refer to parts of the textual descriptions of use cases. Our position is that both extending and extended use cases represent supplementary behavioral increments since in the context of product lines they represent functionality that is only essential for developing product lines. In principle the functionality represented by the extended use cases will be available for less advanced products in terms of functionality.

Alternatives [1] represent supplementary functionality (or supplementary behavioral increments) since they are not essential for a product without variability to function. It shall be noted that alternatives are no longer supplementary when product line members are instantiated from the product line. Alternatives can be modeled with the generalization relationship in use case diagrams, but we recommend to model alternatives with the «*extend*» relationship to evidence their supplementary character according to the UML semantics (when we mention supplementary in the paper, we refer to supplementary behavioral increments from the UML semantics associated with the «extend » relationship). Therefore, the concept of alternative is semantically supported by the «*extend*» relationship. The «*extend*» relationship implies that alternatives are represented as binary and unidirectional dependencies. The alternative relationship is binary and unidirectional because the extending use case (just one or one from many) is an alternative to the extended use case. The extended use case is modeled as the extended one to evidence that it shall be present in products less robust in terms of functionality as opposite to all the others. A situation in which we have more than one alternative to a specific use case shall be represented with that specific use case as the extended use case and the other use cases as the extending ones relatively to that specific use case (the «*extend*» relationships shall be marked with the stereotype «*alternative*»). Situations with a high number of alternatives shall be modeled with different diagrams that shall have the extended use case in common.

If the intention is to use differential specification, specializations [1] shall be modeled with the «*extend*» relationship to evidence their supplementary character according to the UML semantics, otherwise they shall be modeled with the generalization relationship. Differential specification of specializations means that specialization use cases represent supplementary functionality regarding the use case they specialize, therefore, a product without variability does not require the specialization use cases to function. Not requiring the specialization use cases implies that the respective use case that has been specialized is not required for a product without variability to function as well. Besides that, a specialization use case is an extending use case and the respective use case that has been specialized is an extended use case, which according to our position means that both represent supplementary functionality as previously explained. It can be concluded that differential specification is related to supplementary functionality from the UML «*extend*» relationship's semantics. In our approach, we use differential specification, therefore a specialization is represented as a relationship through a stereotype applicable to the «*extend*» relationship. A «*specialization*» relationship is an «*extend*» relationship marked with the stereotype «*specialization*».

Options [1] represent functionality that is only essential for a product with variability to function (when developing product lines), therefore options represent supplementary functionality. However, we do not recommend modeling options with the «*extend*» relationship because if the stereotype was on the relationship, the relationship itself would be optional and that is not the case (the use case is not optional with regard to any other use case, rather it is optional by itself).

Options shall be modeled with a stereotype in use cases. The involvement of an option use case in either «*extend*» or «*include*» relationships, or even in none of those does not imply the presence of that use case in all product line members (which makes of it optional).

In principle, an extending use case is a use case that extends another use case both in the case of alternatives and specializations. In the case of specializations, we consider that there is no multiple inheritance, therefore it is impossible for an extending use case to extend more than one use case. If we have more than one alternative use case for the same functionality, one of those use cases shall be the alternative to all the others and extended by them. That use case is the one to be present in the products less robust in terms of functionality. The extended use case is not aware of the

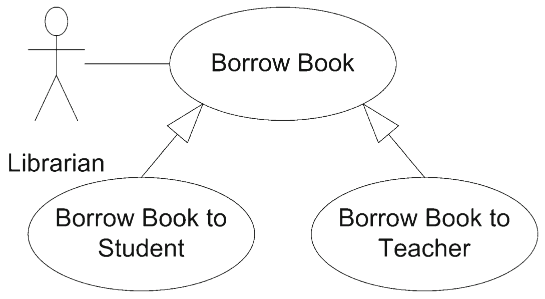


Fig. 2 The specialization of the variant use case *Borrow Book* with a single actor

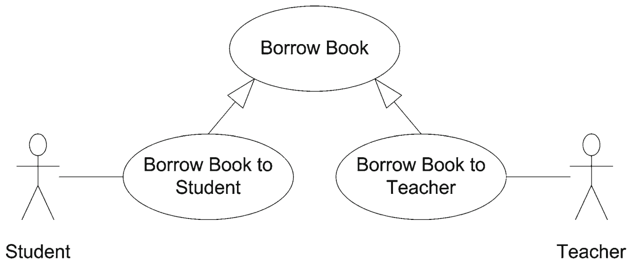


Fig. 3 The specialization of the use case *Borrow Book* with two different actors

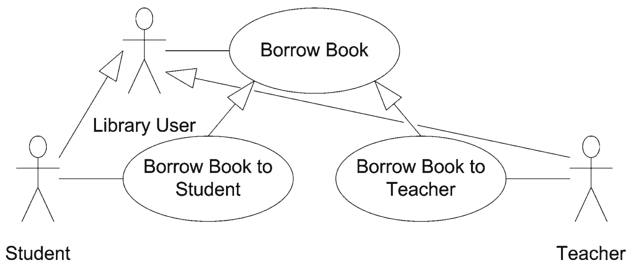


Fig. 4 The specialization of the variant use case *Borrow Book* with two different actors

functionality described in the extending use case. We impose the restriction of multiple inheritance due to the technology we use to implement use cases in our projects. It is not an argument to avoid the discussion of the topic in this paper.

As previously mentioned, if the intention is not to use differential specification, generalization relationships shall be used. However, we may argue in a different way that the generalization relationship shall not be used to represent specializations in contexts of variability. Consider the examples depicted in Figs. 2, 3, 4 and 5. The example is an exception in terms of the (GoPhone) case study we will use further on this paper. The figure shows that the use case *Borrow Book* can be specialized into *Borrow Book to Student* and *Borrow Book to Teacher*. If the actor is the same (the *Librarian*, who registers the borrowing), then the use cases that specialize the *Borrow Book* use case are alternatives to borrowing a book as both can be performed by the same actor. If the actor is not the same (the *Student* in the case of the *Borrow Book to Student* and the *Teacher* in the case of the *Borrow Book to Teacher*),

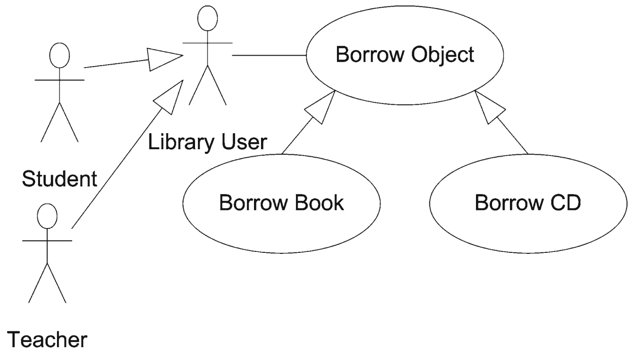


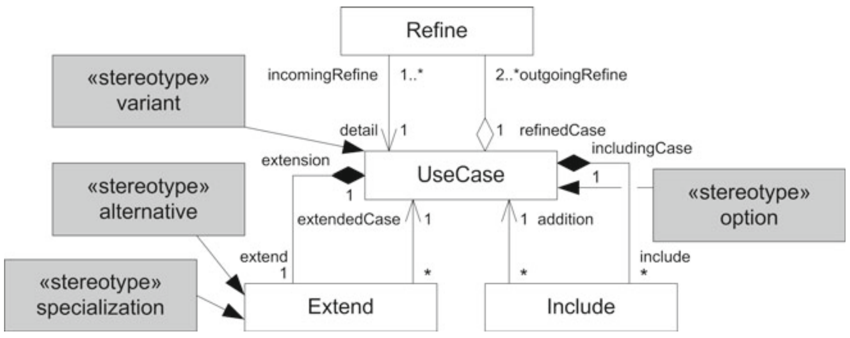
Fig. 5 The specialization of the variant use case *Borrow Object*

then the use cases that specialize the *Borrow Book* use case are not alternatives to borrowing a book as both cannot be performed by the same actor (the same actor does not have an alternative way of borrowing a book). Although the use case Borrow Book is connected to no actor, it is a use case that is in fact connected to the actors of the use cases that specialize it (Student and Teacher). If both of these actors were connected to the use case Borrow Book, it would not be explicit which part of the use case they would perform (either the one related to the book borrowing to student or the other related to the book borrowing to teacher). In this case, for the generalization to be considered as variability, the actor of *Borrow Book* has to be the *Library User* (connected to *Borrow Book*) specialized into the *Student* (connected to *Borrow Book to Student*) and into the *Teacher* (connected to *Borrow Book to Teacher*). Following the semantics of generalization in what actors in use case diagrams are concerned, being Student and Teacher subtypes of Library User, they both interact with the specific subuse case they are associated with, as well as with the superuse case they are associated with via the superactor. Contrarily to the case of Fig. 3, in Fig. 4 we know explicitly which part of the superuse case Borrow Book they perform (the Student performs the one related to the book borrowing to student and the Teacher performs the other related to the book borrowing to teacher). Another example: the use case *Borrow Object* can be specialized into *Borrow Book* and *Borrow CD*. In this case the actor can be the same for all the use cases (the *Student* OR the *Teacher*). To support all the actors at the same time (the *Student* AND the *Teacher*), the *Library User* has to be specialized into them (the *Student* and the *Teacher*) and connected to the *Borrow Object* use case. In this way, the same actor (the *Library User*) can borrow an object (a *Book*) or alternatively another (a *CD*).

4.1 Extending the UML for modeling variability in use cases

Figure 6 depicts the extension we propose to the UML metamodel concerning the «*extend*» relationship and use cases. We have added the stereotypes «*alternative*», «*specialization*» and «*option*» to the standard UML stereotypes to distinguish the three variability types that were to be translated into stereotypes to be applicable to use cases. We have also added the stereotype «*variant*» to the standard UML stereotypes to mark use cases at higher levels of abstraction before they are realized into alternatives or specializations. A use case can include some use cases that are not marked with «*variant*» since they are alternatives (involved in «*alternative*» relationships), they are involved in «*specialization*» relationships or they are non-option and non-variant (if not marked with any stereotype and not involved in «*alternative*» or «*specialization*» relationships). For instance, *Send Message* (Fig. 10) is at the highest level of detail and it is marked with «*variant*». Some of the use cases it includes are not marked with «*variant*» since they have been realized as alternatives (involved in «*alternative*» relationships), or they are non-option and non-variant (if not marked with any stereotype and not involved in «*alternative*» relationships). We could have modeled use cases marked with «*variant*» in our approach as being related to variation points. Usually, a variation point is associated with one or more variants (from [19]). We have not adopted variation points to avoid additional graphical elements to use case diagrams, to avoid more complexity to use case diagrams and to avoid reasoning about variability that shall be present in decision models. We propose the stereotype «*option*» to be applicable to use cases that represent options. «*option*» is for marking use cases that

Fig. 6 The proposed extension to the UML metamodel (figure 16.2 from [3]) for modeling variability in use case diagrams



are not mandatory for all product line members. We also propose the stereotypes «*alternative*» and «*specialization*» to be applicable to the «*extend*» relationship for modeling alternatives and specializations, respectively. Extending use cases involved in «*alternative*» relationships do not need to be marked with the stereotype «*alternative*» to evidence them as alternatives since they do not make sense without being involved in that kind of relationships (an alternative use case is always alternative to another use case). The same happens with the stereotype «*specialization*» (a use case involved in a specialization relationship always specializes another use case). Regarding Fig. 6 and the *Extend* metamodel element, as far as the unidirectional association is concerned, the end named *extendedCase* references the use case that is being extended (the extended use case) and the association means that many (zero or more) «*extend*» relationships refer to one extended use case. Regarding the aggregation, the end named *extend* references the «*extend*» relationships owned by the use case, and the end named *extension* references the use case that represents the extension (the extending use case) and owns the «*extend*» relationship. The metamodel means that one «*extend*» relationship is owned by one extending use case. In summary, a use case can be extended by many use cases and a use case can extend another use case. There can be zero or more alternatives («*alternative*» relationships) to a use case. There can also be zero or more specializations («*specialization*» relationships) for a use case. Although it can be argued that specializations are only worth the effort when there are two or more specialization use cases, we do not want to take freedom away from the modeler.

From now on, we either use the «*extend*» relationship without stereotypes or with one of the two stereotypes applicable to this relationship from the proposed extension to the UML metamodel (depending on whether we are modeling alternatives or specializations). Using no stereotypes on the «*extend*» relationship means that no variability is being modeled, otherwise the stereotypes applicable to the «*extend*» relationship from the proposed extension to the UML metamodel shall be used.

It is important to distinguish alternatives from generalizations in contexts of variability. In the case of alternatives, the extending use case is an alternative to the extended use case. In the case of specializations, the extending use cases are alternatives to each other. Figure 7 shows the specialization of two alternative use cases from the GoPhone case study: *Insert Picture* and *Insert Picture or Draft Text*. It is possible to transform alternatives into specializations and the other way around. Again we are not restrictive on this since we do not want to take freedom away from the modeler. *Insert Picture or Draft Text* is an alternative to *Insert Picture* because it extends the functionality represented by *Insert Picture* (which means that in this case and in the context of product lines it is an alternative to *Insert Picture*).

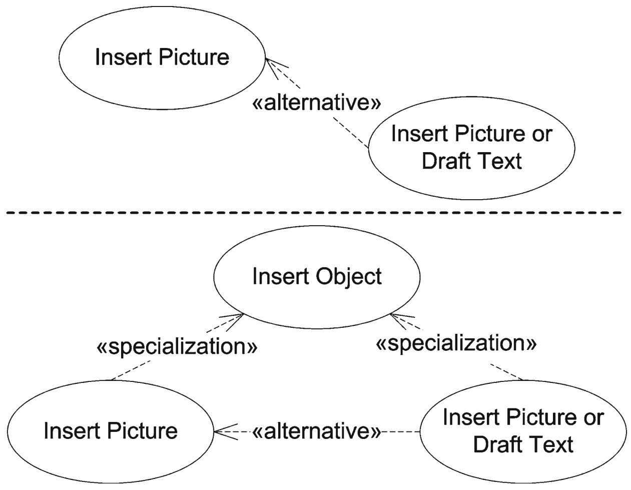


Fig. 7 The specialization of *Insert Picture* and *Insert Picture or Draft Text*

5 Handling variability in use case modeling with refinement

Use cases can be decomposed with or without detailing their non-stepwise textual descriptions. Without detailing those descriptions, we propose to represent the decomposition of use cases in use case diagrams with the «*include*» relationship. This decomposition suits the purpose of e.g. (1) modeling an alternative to a part of the decomposed use case or (2) modeling a part of the decomposed use case that is an optional part.

We consider that refining means decomposing and simultaneously detailing use cases. By refining use cases, the artifacts resulting from the refinement process (the refining use cases) are situated in lower abstraction levels comparatively to the refined use cases (the use cases that were submitted to the refinement process). To represent in the use case diagram this decrease in the abstraction level when refining use cases, we proposed in [29] to use the «*refine*» relationship (as a sort of traceability between use cases at different levels of detail).

In this section of the paper, we depict in Fig. 8 use cases according to the perspectives of detail*variability to illustrate in abstract terms our approach to use case modeling with support for variability. The detail perspective is intimately related to the activity of use case refinement. In this sense, use cases can be more detailed if they are refined. The variability perspective is associated with the modeling of variability for product line support. The two perspectives (detail and variability) have been converted into axes of the illustrated space: y = detail and z = variability. Each level of the z axis corresponds to a (parallel) plan, which means that we position use cases in variability plans. Thus, the variability plans are the plans that contain use cases representing variability in the three different types that have been translated

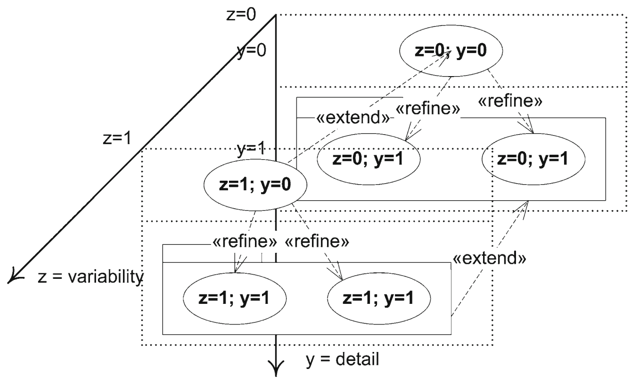


Fig. 8 Use cases positioned according to the perspectives of detail*variability

into stereotypes to be applicable to use cases. The plan $z = 0$ contains none of these use cases that represent variability.

The figure clarifies that the «*refine*» relationships imply increasing the detail level, whereas the «*extend*» relationships do not imply increasing the detail level but rather changing from one variability plan (z plan) to another. Extending use cases represent alternative or specialization use cases, therefore they must be situated at the same level of detail but in different variability plans (z plans). Variabilities do not imply adding detail to the non-stepwise textual descriptions of the use cases like refinements do.

The figure shows the general case of the refinement of two use cases connected through an«*extend*» relationship. The refinement of a use case stereotyped as «*option*» is not relevant here since it is not the case of an «*extend*» relationship connecting two use cases. The figure evidences that the refinement of two use cases connected through an «*extend*» relationship originates more detailed use cases organized in two packages that have also an «*extend*» relationship connecting them. That is not always the case. It is possible to have two use cases connected through a «*specialization*» relationship, which produces «*specialization*» relationships connecting more detailed individual use cases (and not packages) in different variability plans (an example of such case is in the next section of this paper).

6 The variability in the GoPhone case study

The non-stepwise textual descriptions in Fig. 9 were elaborated based on the functional requirements for the GoPhone. We rely on non-stepwise textual descriptions of use cases (the opposite of stepwise textual descriptions of use cases) to model variability in use case diagrams. Stepwise textual descriptions are structured textual descriptions in natural language that provide for a stepwise view of the use case as a sequence of steps, alert for the decisions that have to be made by the user and evidence the notion of use case actions temporarily dependent on each other. Stepwise descriptions shall be treated after modeling the use cases. (Cockburn presents [41] different forms of writing textual descriptions for use cases.)

Figure 10 shows some examples of variability modeled in use cases. The use cases in grey are those that do not represent variability. No borders containing use cases in the variability plans with $z > 0$ were drawn because those borders are going to be needed during product derivation (or the generation of product models from the product line model, which is out of the scope of this particular paper). All the use cases in the diagrams in this section have the values they take for both the perspectives of variability (z) and detail (y). These are not tagged values, rather just a help for the reader to visualize the use cases in the right place. Figure 10 seems complex but it ought to be noticed that the figure contains two diagrams and that the extensions to the use cases in the diagrams could have been modeled in different artifacts. By stating this we say that the diagram in Fig. 10 could have been separated in some diagrams, which we did not do because of space restrictions. Nevertheless, we cannot escape to variability in its different types for the reasons already explained.

The «*include*» relationship involves two types of use cases: the including use case (the use case that includes other use cases) and the included use case (the use case that is included by other use cases). In the context of the «*include*» relationship, the UML Superstructure states that the including use case depends on the addition of the included use cases to be complete. Nevertheless in our opinion, the functionality of the included use cases shall be described in the including use case. Since we rely on non-stepwise textual descriptions of use cases to determine the «*include*» relationships, the including use case has to contain the description of the included use cases so that the modeler is able to define the parts that compose the including use case to decompose that use case (e.g. as can be seen from Fig. 9 the functionality of the *Compose Message* use case is described in the *Send Message* use case).

In the context of the «*extend*» relationship, the UML Superstructure states that an extending use case consists of one or more behavior fragment descriptions to be inserted into the appropriate spots of the extended use case. This means that the functionality of the extending use case is not described in the extended use case. The extended use case is not aware of the functionality described in the extending use case (e.g. as can be seen from Fig. 9 the functionality of the *Automatically Archive Message* use case is not described in the *Archive Message by Request* use case). As Fig. 10 depicts, the use case *Automatically Archive Message* is an alternative to the use case *Archive Message by Request* (they are connected through a kind of «*extend*» relationship, tagged with the stereotype «*alternative*» to evidence that the use case *Automatically Archive Message* is an alternative to

Fig. 9 Non-stepwise textual descriptions from the GoPhone use case *Send Message* and some of its related use cases

Use case name: Send Message
Use case description: The mobile user writes the message in a text editor. The GoPhone connects to the network to send the message. In order for the GoPhone to show an acknowledgement to the mobile user (stating that the message was successfully sent), it receives an acknowledgement from the network. Upon request from the GoPhone, the mobile user chooses to save the message into the sent messages folder.
Alternatives:
The mobile user sends some different kinds of messages through the GoPhone. [Use cases' name: *Select Basic or Extended Kind of Message / Select Basic, Extended or E-mail Kind of Message*]
The mobile user inserts objects into a message. [Use cases' name: *Insert Picture / Insert Picture or Draft Text*]
The mobile user attaches objects to a message. [Use cases' name: *Attach Business Card or Calendar Entry / Attach File, Business Card, Calendar Entry or Sound*]
The mobile user chooses the recipient's contact. [Use cases' name: *Choose Recipient's Phone Number / Choose Recipient's Phone Number or E-mail Address*]
The message is saved into the sent messages folder. [Use cases' name: *Archive Message by Request / Automatically Archive Message*]
Specializations: -
Options: When writing the message, the mobile user activates letter combination (T9).

Use case name: Compose Message
Use case description: The mobile user writes the message in a text editor.
Alternatives:
The mobile user sends some different kinds of messages through the GoPhone. [Use cases' name: *Select Basic or Extended Kind of Message / Select Basic, Extended or E-mail Kind of Message*]
The mobile user inserts objects into a message. [Use cases' name: *Insert Picture / Insert Picture or Draft Text*]
The mobile user attaches objects to a message. [Use cases' name: *Attach Business Card or Calendar Entry / Attach File, Business Card, Calendar Entry or Sound*]
Specializations: -
Options: When writing the message, the mobile user activates letter combination (T9).

Use case name: Archive Message by Request
Use case description: Upon request from the GoPhone, the mobile user chooses to save the message into the sent messages folder.
Alternatives: The GoPhone automatically archives the message [Use cases' name: *Automatically Archive Message*]
Specializations: -
Options: -

Use case name: Automatically Archive Message
Use case description: The GoPhone saves the message into the sent messages folder and notifies the mobile user on the successful message saving into that folder.
Alternatives: -
Specializations: -
Options: -

the use case *Archive Message by Request*). It must be noticed that *Archive Message by Request* is an (included) use case included by the including use case *Send Message*, which means that the functionality of the use case *Archive Message by Request* is described in the *Send Message* use case. For this reason, we could have extended the *Send Message* use case with the use case *Automatically Archive Message*, but then we would not be evidencing to which part of the functionality of the *Send Message* use case the use case *Automatically Archive Message* is an alternative to. Figure 10 also depicts that the *Browse Directory of Pictures* use case is a specialization of the use case *Browse Repository* (they are connected through another kind of «*extend*» relationship, tagged with the stereotype «*specialization*» to evidence that the use case *Browse Directory of Pictures* is a specialization of the use case *Browse Repository*). Option use cases shall be marked with the stereotype «*option*» (e.g. as Fig. 10 evidences for the *Activate Letter Combination* use case).

6.1 Refinement of specializations and alternatives

Figure 11 shows the refinement of the specialization type of variability. The figure shows that both the use case that has been specialized (the *Browse Repository* use case) and the specialization use cases (the *Browse Directory* and *Browse List* use cases) were refined. Some use cases that refine the specialization use cases are specializations of the use cases that refine the use case that has been specialized (e.g. the *View Picture* use case is a specialization of the *View Object* use case). The use case *Open Folder* represents

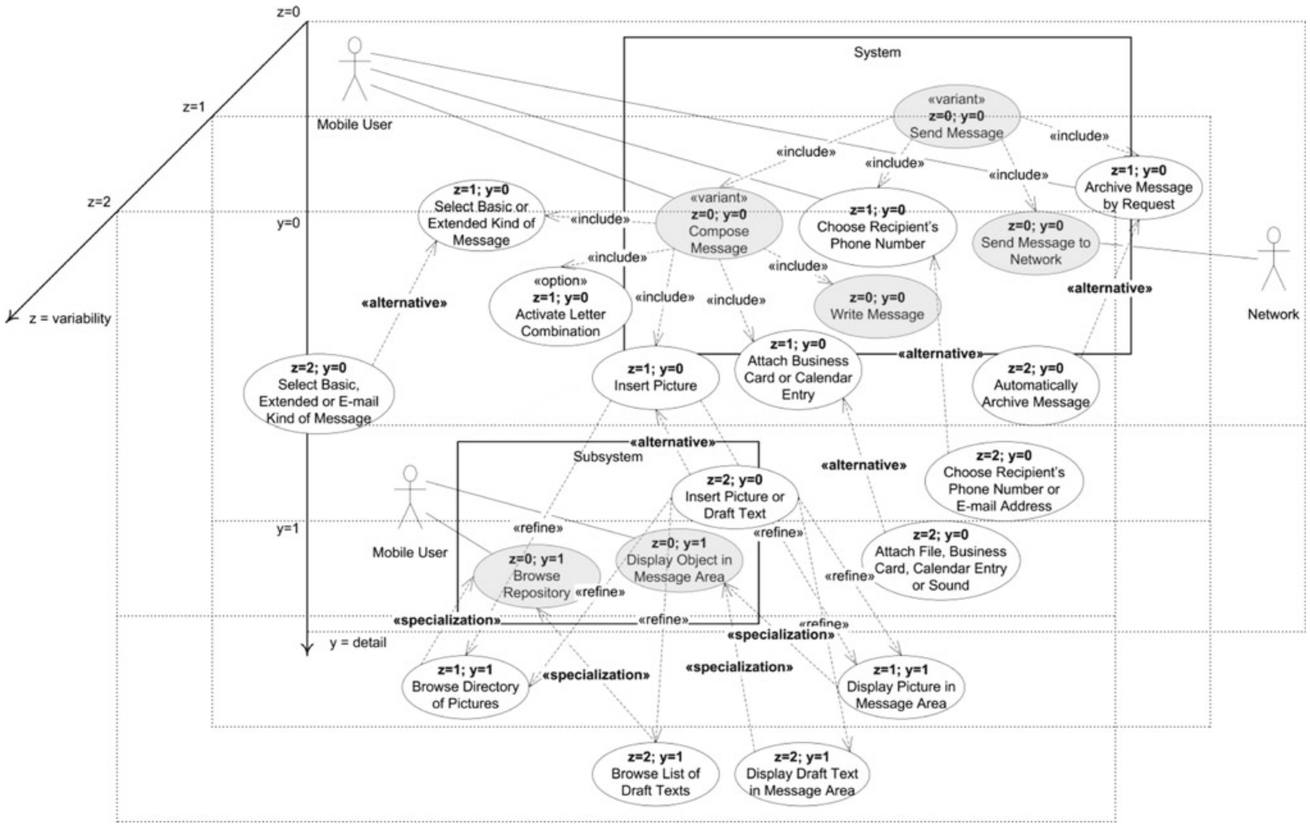

z=0
z=1
z=2
z = variability
y=0
y=1
y = detail
Mobile User
System
«variant» z=0; y=0 Send Message
«include»
z=1; y=0 Archive Message by Request
z=1; y=0 Select Basic or Extended Kind of Message
«variant» z=0; y=0 Compose Message
z=1; y=0 Choose Recipient's Phone Number
z=0; y=0 Send Message to Network
Network
«option» z=1; y=0 Activate Letter Combination
z=0; y=0 Write Message
«alternative»
z=2; y=0 Select Basic, Extended or E-mail Kind of Message
z=1; y=0 Insert Picture
z=1; y=0 Attach Business Card or Calendar Entry
z=2; y=0 Automatically Archive Message
Subsystem
z=2; y=0 Insert Picture or Draft Text
z=2; y=0 Choose Recipient's Phone Number or E-mail Address
z=2; y=0 Attach File, Business Card, Calendar Entry or Sound
z=0; y=1 Browse Repository
z=0; y=1 Display Object in Message Area
«refine»
«specialization»
z=1; y=1 Browse Directory of Pictures
z=1; y=1 Display Picture in Message Area
z=2; y=1 Browse List of Draft Texts
z=2; y=1 Display Draft Text in Message Area

Fig. 10 Variability modeled for the *Send Message* use case from the GoPhone

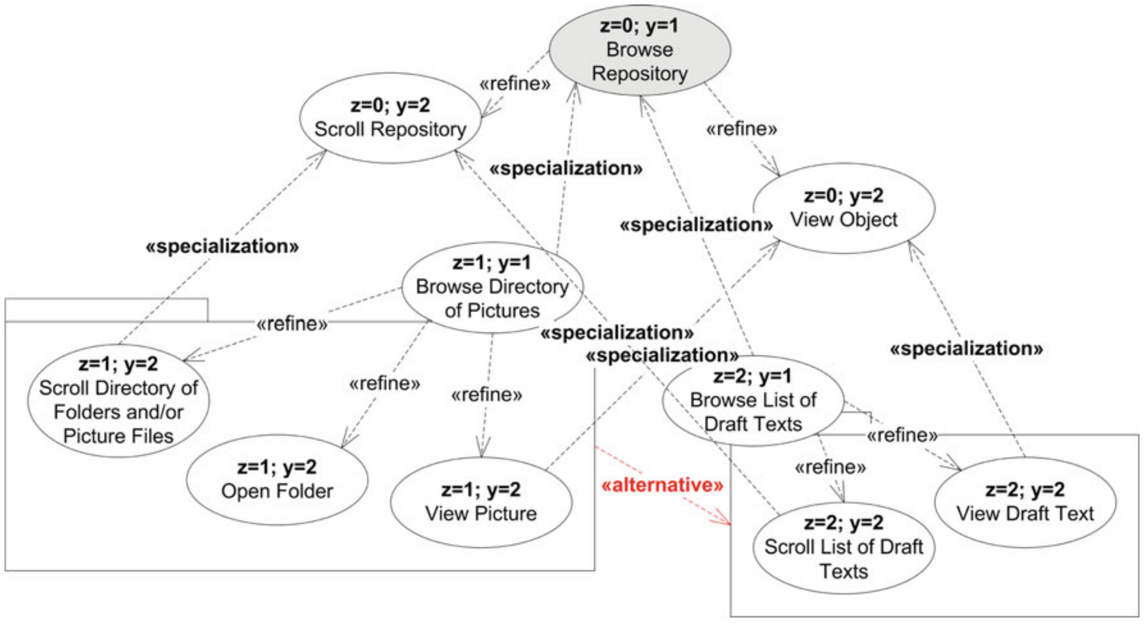

z=0; y=1 Browse Repository
«refine»
z=0; y=2 Scroll Repository
«specialization»
z=0; y=2 View Object
z=1; y=1 Browse Directory of Pictures
z=1; y=2 Scroll Directory of Folders and/or Picture Files
z=1; y=2 Open Folder
z=1; y=2 View Picture
z=2; y=1 Browse List of Draft Texts
«alternative»
z=2; y=2 Scroll List of Draft Texts
z=2; y=2 View Draft Text

Fig. 11 An example of refinement of the specialization type of variability from the GoPhone

functionality that is not common to both specialization use cases since it is only applicable to one of the objects the specialization use cases refer to (the *Directory of Pictures*). Having in mind that specializations are a special kind of alternatives, specialization use cases are alternatives to each other. Figure 11 illustrates that the use cases that refine the

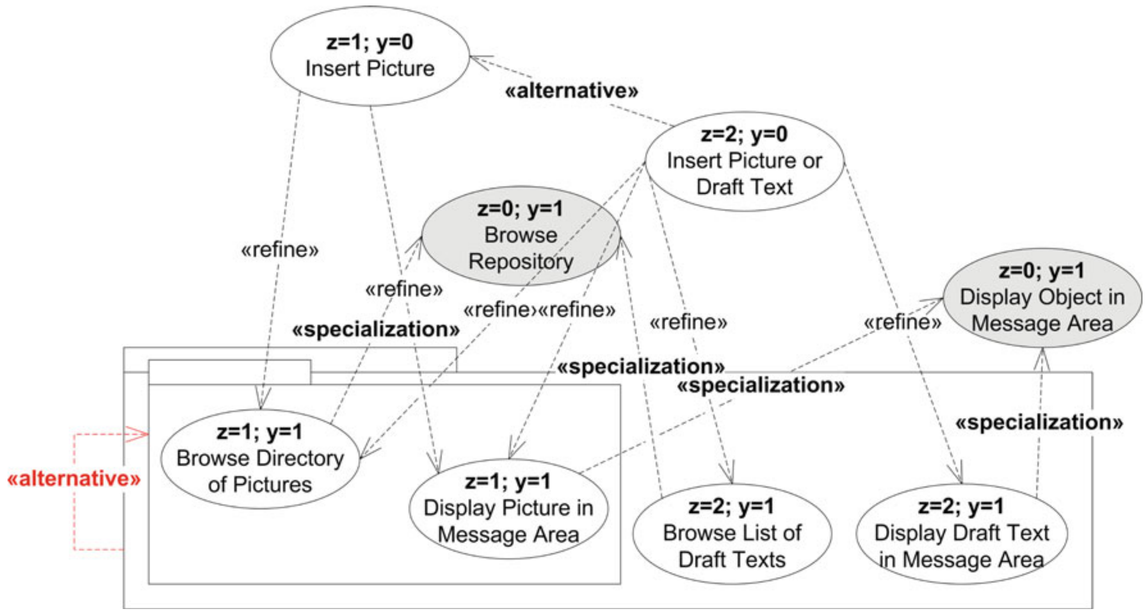


Fig. 12 An example of refinement of alternative variability from the GoPhone

specialization use cases are alternatives to each other as packages.

Figure 12 depicts that the use cases that refine two use cases connected through an «*alternative*» relationship are alternatives to each other as packages.

7 Conclusions

This paper has elaborated on the representation of variability in use case diagrams and the implications of functionally refining use cases when variability is represented in this kind of diagrams. It began by providing an in depth analysis of the state-of-the-art concerned with both if these topics. Based on our position towards the related work, we proposed an extension to the UML metamodel to represent the three types of variability we have synthesized: alternatives, specializations and options. We concluded that alternatives and specializations shall be adequately modeled with the «*extend*» relationship, and that options shall be adequately modeled with a stereotype in use cases. This conclusion was based on the UML metamodel's semantics associated with the relationships for connecting use cases in use case diagrams: alternatives, specializations and options represent supplementary functionality. We have also introduced the functional refinement of use cases connected through «*extend*» relationships due to its pertinence in large-scale product line contexts.

References

1. Azevedo S, Machado RJ, Bragança A, Ribeiro H (2010) The UML «extend» relationship as support for software variability. In: 14th international software product line conference (SPLC 2010), Jeju Island, South Korea. Springer, Berlin
2. Muthig D, John I, Anastasopoulos M, Forster T, Dörr J, Schmid K (2004) GoPhone—a software product line in the mobile phone domain. Fraunhofer IESE, IESE-Report No. 025.04/EMarch 5
3. OMG (2009) Unified modeling language: superstructure—version 2.2. Object Management Group, p 740
4. Bragança A, Machado RJ (2006) Extending UML 2.0 metamodel for complementary usages of the «extend» relationship within use case variability specification. In: 10th international software product line conference (SPLC 2006), Baltimore, MD, USA. IEEE Computer Society, California
5. Bragança A, Machado RJ (2005) Deriving software product line's architectural requirements from use cases: an experimental approach. In: 2nd international workshop on model-based methodologies for pervasive and embedded software (MOMPES 2005), Rennes, France. TUCS General Publications, Turku
6. John I, Muthig D (2002) Product line modeling with generic use cases. In: Workshop on techniques for exploiting commonality through variability management, San Diego, CA, USA. Springer, Berlin
7. John I, Muthig D (2002) Tailoring use cases for product line modeling. In: International workshop on requirements engineering for product lines (REPL 2002), Essen, Germany. Avaya Labs, New Jersey
8. Bayer J, Gerard S, Haugen Ø, Mansell J, Møller-Pedersen B, Oldevik J, Tessier P, Thibault J-P, Widen T (2006) Consolidated Product Line Variability Modeling. In: Käköla T, Duenas JC (eds) Software product lines—research issues in engineering and management. Springer, Berlin, pp 195–241
9. Kang K, Cohen S, Hess J, Novak W, Peterson AS (1990) Feature-oriented domain analysis (FODA) feasibility study. Software Engineering Institute, Carnegie Mellon University, Technical Report
10. Bachmann F, Goedicke M, Leite J, Nord R, Pohl K, Ramesh B, Vilbig A (2004) A Meta-model for representing variability in product family development. In: 5th international workshop on product-family engineering (PFE-5), Siena, Italy. Springer, Berlin
11. Bühne S, Lauenroth K, Pohl K (2005) Modelling Requirements Variability across Product Lines. In: 13th IEEE international

conference on requirements engineering (RE 2005), Paris, France: IEEE Computer Society
12. Gomaa H, Shin ME (2008) Multiple-view modelling and meta-modelling of software product lines. Inst Eng Technol Softw 2:94–122
13. Gomaa H, Shin ME (2004) A multiple-view meta-modeling approach for variability management in software product lines. In: 8th international conference on software reuse (ICSR-8), Madrid, Spain. Springer, Berlin
14. Gomaa H, Olimpiew EM (2008) Managing variability in reusable requirement models for software product lines. In: 10th international conference on software reuse (ICSR-10), Beijing, China. Springer, Berlin
15. Gomaa H (2004) Designing software product lines with uml: from use cases to pattern-based software architectures. Addison-Wesley, Upper Saddle River
16. Webber DL, Gomaa H (2004) Modeling Variability in Software Product Lines with the Variation Point Model. Sci Comput Program 53:305–331
17. Ziadi T, Hélouët L, Jézéquel J-M (2004) Towards a UML profile for software product lines. In: 5th international workshop on product-family engineering (PFE-5), Siena, Italy. Springer, Berlin
18. Coplien J, Hoffman D, Weiss D (1998) Commonality and variability in software engineering. IEEE Softw 15:37–45
19. Halmans G, Pohl K (2003) Communicating the variability of a software-product family to customers. Softw Syst Model 2:15–36
20. Pohl K, Böckle G, Linden Fvd (2005) Software product line engineering: foundations, principles, and techniques. Springer, Heidelberg
21. Salicki S, Farcet N (2002) Expression and usage of the variability in the software product lines. In: 4th international workshop on product family engineering (PFE-4), Bilbao, Spain. Springer, Berlin
22. Maßen Tvd, Lichter H (2002) Modeling variability by UML use case diagrams. In: International workshop on requirements engineering for product lines (REPL 2002), Essen, Germany. Avaya Labs, New Jersey
23. Machado RJ, Fernandes JM, Monteiro P, Rodrigues H (2005) Transformation of UML models for service-oriented software architectures. In: 12th IEEE international conference and workshops on the engineering of computer-based systems (ECBS 2005), Greenbelt, MD, USA. IEEE Computer Society, California
24. Atkinson C, Bayer J, Muthig D (2000) Component-based product line development: the KobrA approach. In: 1st software product line conference (SPLC 2000), Denver, CO, USA. Kluwer Academic Publishers, Dordrecht
25. Jacobson I, Griss M, Jonsson P (1997) Software reuse: architecture, process and organization for business success. Addison-Wesley, Upper Saddle River
26. Greenfield J, Short K (2004) Software factories: assembling applications with patterns, models, frameworks, and tools. Wiley, Hoboken
27. Pons C, Kutsche R-D (2004) Traceability across refinement steps in uml modeling. In: 3rd UML workshop in software model engineering (WiSME 2004), Lisbon, Portugal. Springer, Berlin
28. Eriksson M, Börstler J, Borg K (2006) Software product line modeling made practical. Commun. ACM 49:49–53
29. Azevedo S, Machado RJ, Bragança A, Ribeiro H (2010) The UML «include» relationship and the functional refinement of use cases. In: 36th euromicro conference on software engineering and advanced applications (SEAA 2010), Lille, France. IEEE Computer Society, California
30. Paech B, Rumpe B (1994) A new concept of refinement used for behaviour modelling with automata. In: 2nd international symposium of formal methods Europe (FME 1994), Barcelona, Spain. Springer, Berlin
31. Quartel DAC, Pires LF, Franken HM, Vissers CA (1995) An engineering approach towards action refinement. In: 5th IEEE workshop on future trends of distributed computing systems (FTDCS 1995), Chenju, Korea. IEEE Computer Society, California
32. Darimont R, Lamsweerde Av (1996) Formal refinement patterns for goal-driven requirements elaboration. In: 4th symposium on the foundations of software engineering (FSE-4) San Francisco, CA, USA. ACM, New York
33. Mikolajczak B, Wang Z (2003) Conceptual modeling of concurrent systems through stepwise abstraction and refinement using petri net morphisms. In: 22nd international conference on conceptual modeling (ER 2003), Chicago, IL, USA. Springer, Berlin
34. Batory D, Sarvela JN, Rauschmayer A (2004) Scaling step-wise refinement. IEEE Trans Softw Eng 30:355–371
35. Cherfi SS-s, Akoka J, Comyn-Wattiau I (2006) Use case modeling and refinement: a quality-based approach. In: 25th international conference on conceptual modeling (ER 2006), Tucson, AZ, USA. Springer, Berlin
36. Simons AJH (1999) Use cases considered harmful. In: 29th conference on technology of object-oriented languages and systems (TOOLS Europe 1999), Nancy, France: IEEE Computer Society, California
37. Heldal R (2005) Use cases are more than system operations. In: 2nd international workshop on use case modeling (WUsCaM 2005), Montego Bay, Jamaica. Chalmers Publication Library, Sweden
38. Fowler M (2004) UML distilled: a brief guide to the standard object modeling language. Addison-Wesley, Upper Saddle River
39. Machado RJ, Fernandes JM, Monteiro P, Rodrigues H (2006) Refinement of software architectures by recursive model transformations. In: 7th international conference on product focused software process improvement (PROFES 2006), Amsterdam, The Netherlands. Springer, Berlin
40. Bosch J, Florijn G, Greefhorst D, Kuusela J, Obbink JH, Pohl K (2002) Variability issues in software product lines. In: 4th international workshop on product family engineering (PFE-4), Bilbao, Spain. Springer, Berlin
41. Cockburn A (2000) Writing effective use cases. Addison-Wesley, Upper Saddle River