Modeling behaviors in Product Lines*

Tewfik Ziadi, Loïc Hélouët, Jean-Marc Jézéquel
IRISA, Campus de Beaulieu
35042 Rennes Cedex, France
{tziadi,lhelouet,jezequel}@irisa.fr

abstract: This paper proposes a model for the definition of behavioral requirements in Product lines. These requirements are expressed by High-level Message Sequence Charts (HMSC) extended with constructs for handling variability. Then some clues for defining coherence between static and dynamic parts of the architecture with OCL are given.

1 Introduction

Recent proposals [3, 8, 2] for the development of software product lines (PL) define methodologies leveraging the UML for modeling commonalities and variability across the PL, using UML built-in extension mechanisms such as stereotypes. However, these proposals usually concentrate on static structural variability, and often neglect the behavioral aspects, that are only expressed by means of simple textual fragments. However, the analysis of the requirements domain could benefit from a formalized representation of interactions between constituents of a system.

This paper proposes the integration of behavioral aspects in a UML-based product line architecture. These interactions are described by means of High-level Message Sequence Charts (HMSC for short), a scenario language first standardized by ITU [5], that is bound to be integrated in UML 2.0. Hence, a product derived from a product line will be described by a UML model including all its features, but also by a set of scenarios describing the main interactions among parts of the system.

One of the main challenges of this work is to define behavioral variability on a scenario language. At first sight, HMSC do not seem to contain such notions. Hence, we propose to define variability through existing features such as inheritance, and to extend the notation with new constructs: variation points and optional behaviors. Another challenging task is to define a notion of coherent product lines. In order to be able to generate products from a product line, one has to ensure compatibility of features in the static model, between scenarios of the dynamic model, but also a global coherence between static and dynamic aspects of the product line.

This paper is organized as follows. Section 2 describes the static and dynamic aspects of our product line model, section 3 proposes some definitions for the coherence of a model, and section 4 concludes this work.

2 Models

This section presents a software Product Line Architecture model integrating dynamic aspects. HMSC are used to build dynamic assets for product line. In the literature many solutions (see for example [3]) are proposed for extending UML class diagram to support variability. The static model proposed hereafter uses templates, optionality, and inheritance to define variability. For illustrating the constructs introduced hereafter, we propose a small ad-hoc example: a product line

*This work has been partially supported by the ITEA project ip00004, CAFE in the Eureka Σ! 2023 Programme

for a digital camera. A digital camera comports an interface, a memory, a sensor, a display, and may propose a compression feature. The main variation points in this example concerns the presence of compression and the format of images stored, which can be parameterized.

2.1 Static Model

UML class diagrams describe sets of classes and their dependencies, but do not allow the definition of variability. Variation points can be expressed on this static model using mechanisms such as templates, stereotypes, and inheritance. A **template**[4] is the descriptor for a class with unbound formal parameters. It defines a family of classes that can be derived by binding the parameters to actual values. The **stereotype** concept provides a way of classifying model elements. A stereotype can be associated to any UML element. It allows the definition of new UML extensions without modification of the core of the language (See extension mechanisms p2-79 of the UML standard [4]).

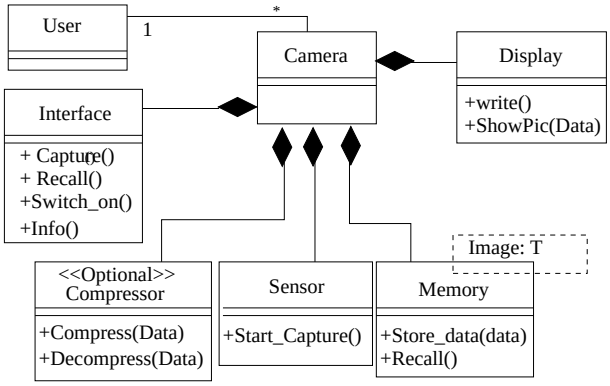

Figure 1: Class diagram for the digital camera

Variability may appear in products in different ways. Products may differ only on specific parameters (the size or color of a window, the memory available, ...). This kind of variability is called **Parameterization**. To implement parameterization, product lines must allow for the definition of generic assets with a set of parameters. Each product will then bind these parameters in a specific way. UML template classes can specify classes parameterized by a type.

Another kind of variability called **optionality** appears when a part of a product (feature, functionality,...) can be implemented or ignored by products from a family. Product line models gather information about a set of products. They must therefore be more generic than product models, and include elements from all the products of the family they represent. Some elements will be common to all products, but for each product, some **optional** elements will be omitted. To show optionality information we use a specific stereotype «Optional», that can be associated to any part of the class diagram.

In addition to parameterization and optionality, standard constructs such as inheritance can be used to define variants. In [6], for example, variants of a product are built from a model leveraging Creational Design Patterns. Example of Figure 1 shows the static model of a digital camera Product line, in which class *Compressor* is optional and class *Memory* is parameterized with image type. Other parameters, such as the size of a buffer, can be instantiated using model transformations.

2.2 Dynamic Model

In addition to the static aspects of products, a user of a product line may also want to express **behavioral variations**. A family of network softwares, for example, may use different commu-

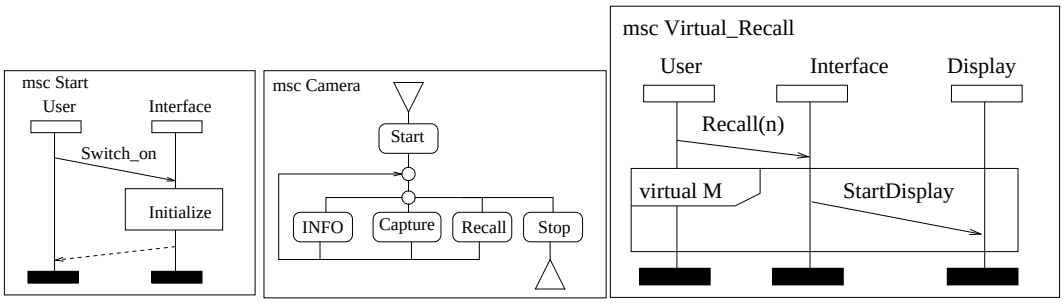

Figure 2: a- bMSC b- HMSC c- Virtual MSC

nication protocols, and components of a system may cooperate differently in different products. Furthermore, a certain coherence between static and dynamic aspects of a product model should be ensured by the product line. We propose to define dynamic aspects of products with Message Sequence Charts, and to use the OCL [11] for the expression of coherence properties.

Message Sequence Charts is a scenario language standardized by ITU [5]. The latest evolution of the language is very close from the future sequence diagrams of UML 2.0. MSC is composed of two kinds of diagrams, basic MSC (bMSC) and High-level MSC (HMSC). They define a set of basic charts composed by means of alternative, parallel composition, and iteration operators. A bMSC describes communications between entities of a system called instances. Diagram of Figure 2-a represents a bMSC defining communications between instances $User$ and $Interface$. HMSC $Camera$ of figure 2-b describes a set of scenarios starting with the interactions defined in bMSC $Start$, in which the communication patterns defined by $INFO$, $Capture$, and $Recall$ can be repeated an unlimited number of times before the occurrence of $Stop$. Interested readers can consult [10] for a complete presentation of the language.

Message Sequence Charts are equiped with a semantics based on process algebra [7], and their properties have been well studied during the last decade. Furthermore, it is interesting to describe behaviors at a high level of abstraction with a visual and intuitive formalism that allows formal manipulations. The main idea for integrating MSC into a PL approach is to associate a set of typical behaviors to a product model derived from the PL. Behaviors from a product to another differ only by small details. Hence managing different and disjoint sets of scenarios for each product would be a very heavy construction, and would not capture the notion of commonality and variability inherent to a product line approach.

Unfortunately, in its current form, MSC is not equiped with features allowing the expression of variability and commonality. We propose to use the inheritance mechanisms of [9], and to extend the language with some new constructs for expressing variability. Since HMSC are supposed to be a part of the new UML standard, variation constructs can be easily defined by specializing alternative and MSC references with ad hoc stereotypes. The main idea for integrating scenarios into a product line is to maintain a set of generic scenarios with variation points, and scenario assets. From this behavioral asset base, a set of "terminal" scenarios can be derived for each product by instanciation of variation points, and used for simulation, documentation, test purposes, etc. Note that the semantics of variable scenarios is defined as the semantics of derived scenarios. A terminal scenario derivation is obtained by syntactic replacement of a variable part, or by abstraction of an optional part. Hence, the semantics of a scenario can be very different from a product to another. If semantics preservation for some parts of a MSC is required in all variants, then additionnal constraints can be attached to variation points. One may for example require that all possible choice for variation point $Comp$ in MSC $Capture$ of Figure 4 impose an order from an event performed by the $Sensor$ to an event performed by the $memory$.

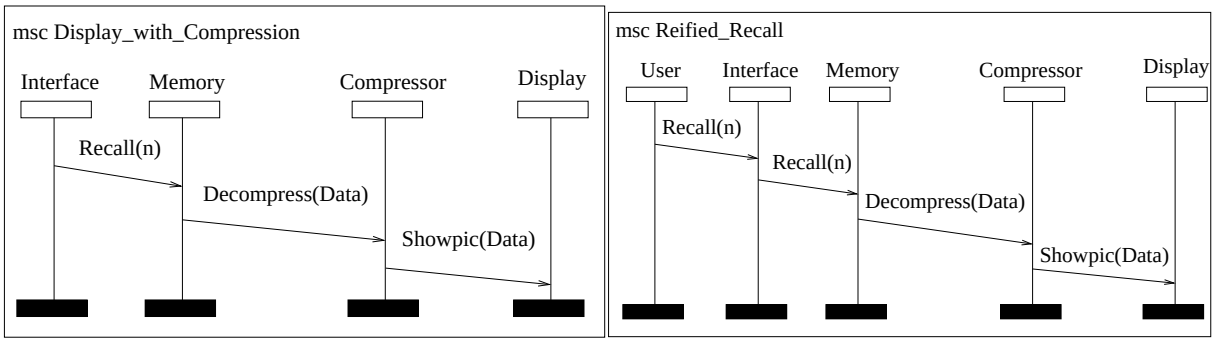

Figure 3: a) bmsc DisplayWithCompression b) bMSC Reified_Recall

A **virtual part** of a bMSC can be refined to obtain another bMSC. The example of Figure 2-c shows a MSC *Virtual_Recall* with a virtual part M. The derived MSC of Figure 3-b is obtained by applying the redefinition proposed by [9]: "`msc Redefined_Recall = Virtual_Recall with M redefined by Display_With_Compression`". In a product, a virtual part can be replaced by another MSC. When a virtual part is not redefined, the behavior contained in the virtual frame is supposed to be the default behavior.

A **variation point** in a MSC has the common meaning in PL approaches: for a given product, only one alternative defined by the variation point will be present in the scenario. Note that this variation can not be expressed by the choice operator of MSC. For example, the choice of Figure 2-b allows the conformance to the behavior described by *Capture* at the first occurrence of the choice, and then conform to *Recall* at the next occurrence of the same choice. Replacing this choice by an alternative between *Capture* and *Recall* would allow for the derivation of two products: one allowing image capture, and another allowing image displaying. A variation point is depicted by means of a rectangular frame, labeled by a variation name. Variable behaviors are separated by a dashed line. The example of Figure 4-b contains a variation point *Comp*, and two possible behaviors describing how data is stored depending on the presence of a compression device.

[3] defines **optionality** for instances and messages of sequence diagrams. We propose to introduce optional instances in MSC and a new construct called optional behaviors. Optional instances are represented by a dashed life line, and their name is tagged with the «optional» keyword. When an optional instance is not present, any incoming or outgoing message should be removed from the resulting behavior. An optional part can be included or removed from a bMSC the same way. Figure 4-b contains an optional instance *Compressor*. An optional behavior in defined by a rectangular frame labeled by an option name. Figure 4-a contains an optional part M that makes the memory information display optional.

3 Constraints on PL models

Bass et al [1] define product line architecture as a set of components, connectors, and additional constraints. Obviously, a product line is more than a collection of models, and has to ensure a certain coherence between it components. The PL architecture model we propose is composed of a static part (a stereotyped class diagram), of a set of behavioral requirements (HMSC) and of a set of constraints expressed in OCL. The model associated to a product can be considered as a sub-model of the PL class diagram, and as a set of HMSC, obtained by choosing a specific variant for each variation point, and satisfying some constraints. These constraints can be applied to both static and dynamic parts, and can be of different kind:

Generic constraints are constraints that apply to any product line, and define structural proper-

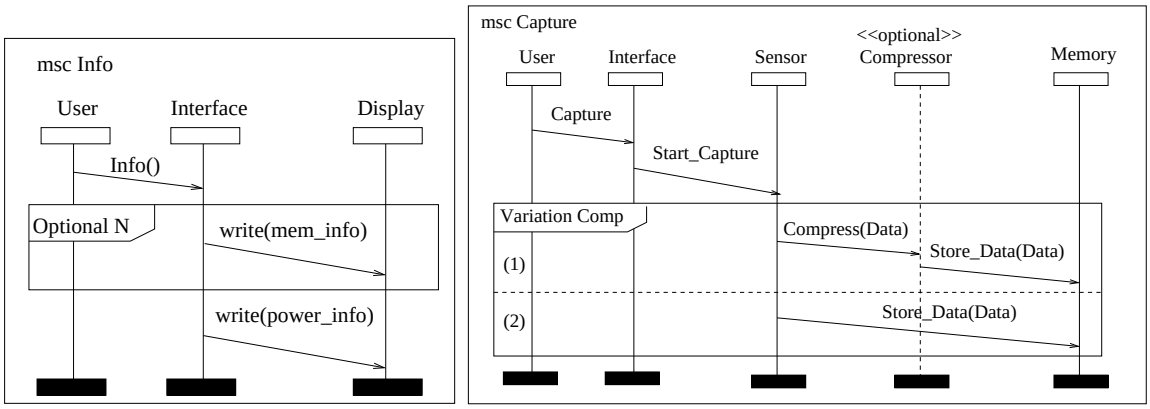

Figure 4: a) optional behavior b)optional instances and variation point

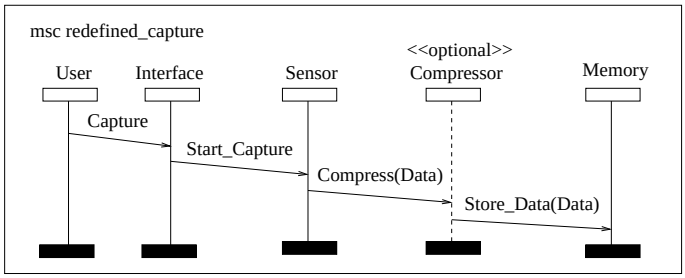

Figure 5: MSC with options and variation points fixed

ties. An example of such constraint is the dependency constraint, that forces non optional elements to depend only on non optional elements. This property can be expressed by the following OCL meta-level constraint, where `isStereotyped(S)` is an auxiliary OCL operation indicating if an element is stereotyped by a string S:

context Dependency
–Warning if a mandatory element depend on the optional one
inv:self.supplier → **exists**(S:ModelElement | S.isStereotyped("optional") **implies** self.client → **forAll**(C: ModelElement | C.isStereotyped("optional"))

Generic constraints can also concern the requirement part of the PL model. For example, instances that appear as optional in bMSCs should be instances of classes that are described as optional in the static model. When the UML 2.0 meta-model contains a definitive version of sequence diagrams it will be possible to express such a constraint as an OCL meta-level constraint. **PL specific constraints** specify properties and dependency relationships between architecture elements. Such a constraint can be for example a presence constraint between two optional classes $C1$ and $C2$. This can be expressed by the following OCL meta-level constraint:

context Namespace
– Presence of class C1 in a such Namespace requires the presence of class C2 in the same Namespace
inv : presenceClass(self, C1) **implies** presenceClass(self,C2)
presenceClass(N:Namespace, class:Class):boolean
post: result = N.ownedElement → **exists**(C:ModelEl ement|C.oclIsTypeOf(Class) **and** C.name=class.name)

PL specific constraint can also involve the dynamic part of the model. For example, on the digital camera Product Line, if class *compressor* is not instantiated, the virtual part of MSC

Virtual_Recall must not be redefined by MSC *Display_With_Compression*. Note that the coherence notion between static and dynamic part does not only concern optional instances, as the presence of components or functionalities of a system can influence the way communications are designed for the rest of the system.

4 Conclusions

We have introduced a formal dynamic representation of behaviors in UML-based product line architectures. Dynamic behaviors are represented by HMSC, where variability is introduced by constructs such as optional instances, optional parts, inheritance, and variation points. Constraint for the coherence of a model can be expressed with the OCL language. One of the main advantages of this approach is that static parts, dynamic parts, and constraints on a product line are expressed with UML elements, and therefore can be easily supported by a UML CASE tool. From this work, one of the challenging point is to derive automatically and efficiently a product model.

References

[1] L. Bass, P. Clements, and R. Kazman. *Software architecture in Practices*. The SEI Series in Software Engineering. Addison-Wesley, 1998.

[2] F. Boisbourdin, B. Pronk, J. Savolaien, and S. Salicki. System family requirements classification and formalisms. Technical Report WP2-0011-01, ESAPS-ITEA, november 2000. ESAPS deliverable.

[3] J.C Duenas, W. El Kaim, and C. Gacek. Style, structure and views for handling commonalities and variabilities - esaps deliverable (wg 2.2.3). Technical report, ESAPS Project, 2001.

[4] Object Management Group. Omg unified modeling language specification, version 1.4, September 2001.

[5] ITU-T. Z.120 : Message sequence charts (MSC), november 1999.

[6] J.M Jézéquel. Reifying variants in configuration management. *ACM Transaction on Software Engineering and Methodology*, 8(3):284–295, July 1999.

[7] M. Reniers and S. Mauw. High-level message sequence charts. In A. Cavalli and A. Sarma, editors, *SDL97: Time for Testing - SDL, MSC and Trends*, Proceedings of the Eighth SDL Forum, pages 291–306, Evry, France, Septembre 1997.

[8] Andreas Reuys, Klaus Pohl, Cristina Gacek, et al. System family process frameworks. Technical Report ESI-WP2-0002-04, ESI, december 2000. ESAPS technical report.

[9] E. Rudolph, P. Graubmann, and J. Grabowski. Message Sequence Chart: composition techniques versus OO-techniques - 'tema con variazioni'. In R. Bræk and A. Sarma, editors, *SDL'95 with MSC in CASE*, Proceedings of the Seventh SDL Forum, pages 77–88, Amsterdam, 1995.

[10] E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.

[11] J. Warmer and A. Kleppe. *The Object Constraint Language-Precise Modeling with UML*. Object Technology Series. Addison-Wesley, 1998.