EPiC Series in Computing

Volume 79, 2021, Pages 99–112

Proceedings of ISCA 34th International Conference on Computer Applications in Industry and Engineering

A SysML-based Holistic Variability Modelling of Software and Systems Product Lines

Olfa Ferchichi[1], Raoudha Beltaifa[1], Raûl Mazo Peña[2], Lamia Labed Jilani[3]

[1] Laboratoire de Recherche en Génie Logiciel, Applications distribuées, Systèmes décisionnels et Imagerie intelligentes (RIADI), Université de Manouba

[2] ENSTA Bretagne, France

[3] Université de Tunis

olfaferchi@yahoo.fr, raoudha.beltaifa@ensi.rnu.tn, raulmazo@gmail.com, lamia.labed@isg.rnu.tn

Abstract

Software and Systems Product Line (SSPL) engineering has shown capabilities to reduce costs and time to market. This is thanks to the creation and management of a common platform dedicated to develop a family of products. These latters can be a family of mobile phones, a family of different brake systems variants for automative needs, etc. Recently, more and more large-scale companies start to implement SSPL engineering in their domains by adopting Model-Based Systems Engineering (MBSE). Systems Product Line (PL) engineering is much broader than software PL engineering. Therefore, various aspects of variability (e.g. functional and quality attributes variabilities) have to be considered in MBSE. However, variability integrated in MBSE is still limited to functional variability. This paper contributes to enhance the SSPL modelling based on SysML by extending the SysML language. The principle aim is to include various aspects of variability. In fact, a holistic variability model is proposed to define the SysML extensions by means of the UML profiling mechanism. This permits to express variability constructs in different SysML modelling artifacts. We also present an application example namely the brake systems family extracted from Splot repository. We in fact, show how our SysML extensions are concretely used.

1 Introduction

Software and Systems Product Lines (SSPL) offers common platform to develop a family of products (Bosch, 2013). by highlighting commonalities and variability between systems. If we take the example of a mobile phone family, all the products have a screen, a phone line, a camera, a drum

kit but they can differ in the type of operating systems (Android or Windowsphone, etc.), in having or not a storage element, etc. SSPL frameworks use models as useful ways to manage the underline complexity of the product lines variability (Pohl, 2005), (Mikko, 2019). These models provide communication capabilities between different system stakeholders. Recently, Model-Based Systems Engineering (MBSE) techniques are applied to support SSPL engineering. Some works have extended UML to manage variability of software product lines such as (Dumitrescu, 2014). Others extended SysML to manage variability of SSPLs such as (Gaeta, 2015) and (Trujillo, 2010). However, modelling variability by extending UML and SysML remains limited. Major reasons is a lack of representing various aspects of variability (e.g. quality attribute variability like variabilities in terms of security or capacity storage, etc.). Motivated by these lacks, this paper proposes: 1) a holistic variability model to represent variability in product lines according to different views and 2) a UML profile, named SysML4SSPL which is an extension of SysML UML profile, to enhance SSPL modelling with SysML. The remainder of this paper is structured as follows. Section 2 provides a brief overview of the necessary background on variability in product lines and SysML language. Section 3 presents the Holistic Variability Model we propose to model various dimensions of PL variability. Section 4 presents the UML profile we propose, called SysML4SSPL. Section 5 presents related research. Finally, Section 6 presents the conclusions and suggests future research directions.

2 Background

This section describes background on SysML and variability modelling in PLs.

2.1 SysML

SysML (OMG, 2019) is an extension of UML in the sense that it reuses some elements of the UML meta-model and integrates new elements defined in a profile called SysML. SysML offers system engineers several improvements over UML; for instance, it reduces the software-centric restrictions of UML and adds more diagram types such as block definition diagrams, internal block diagrams, parametric and requirements diagrams. Due to the above additions, SysML is able to model a wide range of systems such as hardware, software, information and processes.

The SysML diagrams can be used to specify system requirements, behavior, structure and parametric relationships. These are known as the four pillars of SysML.

The system structure is represented by Block Definition Diagrams and Internal Block Diagrams. A Block Definition Diagram describes the system hierarchy and system/component classifications. The Internal Block Diagram describes the internal structure of a system in terms of its Parts, Ports, Interfaces and Connectors. Parts are the constituent components that make up the system defined by the Block. Interfaces define the access points by which Parts and external systems access the Block. Connectors are the links or associations between the Parts of the Block. Often these are connected via the Ports.

The behavior diagrams include the Use Case Diagram, Activity Diagram, Sequence Diagram and State Machine Diagram. The Requirement Diagram captures requirements hierarchies and the derivation, satisfaction, verification and refinement relationships. The relationships provide the capability to relate requirements to one another and to relate requirements to system design model elements and test cases. The parametric diagram represents constraints on system parameter values such as performance, reliability and mass properties to support engineering analysis.

SysML includes an allocation relationship to represent various types of allocation including allocation of functions to components, logical to physical components and software to hardware. Finally, SysML includes traceability relationships such as refine and verify.

2.2 Variability in product lines

One of the key aspects of product lines is variability. Variability is defined as the ability of a software system or artifact to be efficiently extended, changed, customized, or configured for use in a particular context (Svahnberg, 2005). Variability is explicated in a software variability model, which we use broadly to refer to any kind of artifact abstracting or documenting variability.

Variability models are divided into two classes (Dumitrescu,2014): 1) Dedicated variability models that are independent from existing models and propose constructs specific to variability such as Feature Model (FM), see for example figure 1, and Orthogonal Variability Model (OVM) and 2) Integrated variability models that rely on the existing models of software which are adapted or extended with variability information. For example, variability stereotypes associated to variability constructs used in OVM and FM are specified for UML and SysML extensions. There are only a few modelling constructs used in an orthogonal variability model. A *variation point* documents a variable item and thus defines "what can vary" (without saying how it can vary). A *variant* documents a concrete variation and is related to a variation point. A variant thus defines "how something can vary". In addition, *variability constraints* can be defined to specify restrictions about the variability; e.g., to define permissible combinations of variants in an application or to define that the selection of one variant requires or excludes the selection of another variant.

A feature model (Schobbens, 2006) is a tree or a directed acyclic graph of features. A feature model is organized hierarchically. A *mandatory* feature has to be selected if its parent feature is mandatory or if its parent feature is *optional* and has been selected. Mandatory features define commonalities. *Optional*, *alternative*, and '*or*' features define variability in feature models. As a result, a feature model is a compact representation of all mandatory and optional features of a software product line. Each valid combination of features represents a potential product line application. Since the introduction of Feature-Oriented Domain Analysis (FODA) by (Kang, 1990), over 40 different feature model dialects have been proposed (Kang,1990). Based on the expressiveness of those extensions, they can be grouped into three categories: basic feature models (offering mandatory, alternative and 'or' features, as well as 'requires' and 'excludes' cross-tree constraints), cardinality-based feature models (offering, in addition, UML-like multiplicities for feature selection [m..n]) and extended feature models (adding arbitrary feature attributes; e.g., to express variation in quality requirements).

Mainly, variability modelling considers functional variability. However, quality variability in software product lines has not received so much attention by researchers. In a product line different members of the line may require different levels of a quality requirement, for instance they could differ in terms of their availability, security, reliability, etc. Due to this variability, quality evaluation in software product lines is much more complicated than in single-systems.

The definition of variability dimensions facilitates the discovery of variation points and allows understanding of its meaning, origin and rationale. This, in turn, facilitates the process of selecting the appropriate variant. For this to be possible, apart from the identification of the dimensions, a modelling framework for both accommodating variability and allowing the selection of variants needs to be introduced for each dimension.

3 A Holistic Variability Model

In this section, we present a holistic variability model. In our perspective, a holistic variability model is the product of the conjunction of different dimensions of variability views, i.e. functional and non-functional views of variability. In other words, each dimension can be seen as a view on a

particular aspect of the variability at a product line. At moment, we have defined four dimensions (see table 1): Functional, Quality Attribute, Architectural and Contextual. To describe these dimensions and their application on SysML modelling of Product Lines, we consider the Electric Parking Brake (EPB) System (see figure 1). This example is well known and it is extracted from Splot repository (Splot, 2021).

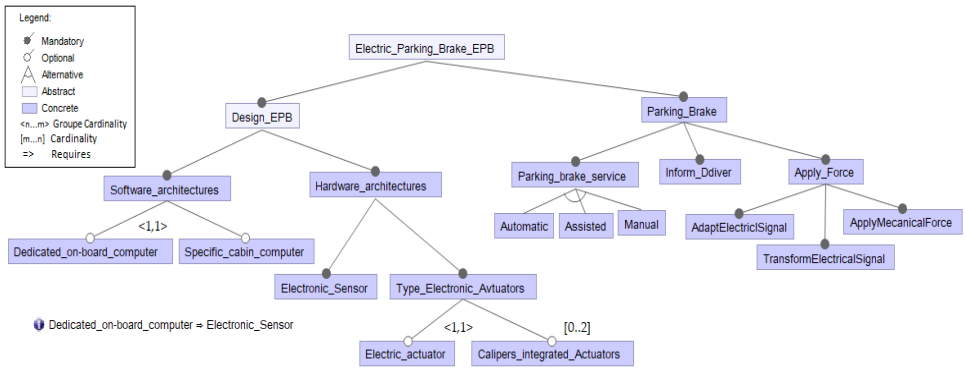


Figure.1. Electric Parking Brake (EPB) System Feature Model (Splot, 2021)

The "Functional" dimension aims at modelling all aspects of variability inside functional elements models of product lines, such as features of FMs and use cases of use case diagrams. In fact, modelling variability inside a SysML use case diagram or a Feature Model of a Product Line should consider the following properties: Optionality, Variation, Priority, Constraint, Dependency, Relationship, Risk, Binding time and Visibility. If we consider a use case, which is defined inside the use case diagram of a PL then the use case may have the following properties: 1) It can be optional or mandatory. 2) It can represent a variation point or a variant. 3) It can have the priority level: S (Small), M(Medium), L(Large) or XL(eXtra Large). A use case with a XL priority has an important business value (BV) for the Product Line. Therefore, its realization takes precedence over all use cases with lower priority (S, M or L). 4) It has a dependency relation with other use cases. To this end, various types of dependencies defined as constraints are identified, which are *include*, *extend*, *after*, *before*, *during*, *inside* and *outside* constraints. *Include* and *exclude* constraints indicate that two use cases are fully dependent or one is exclusive of the other, respectively. Constraints *after*, *before* and *during* indicate evolution in time of use cases, which are specializations of the same parent use case (UC). For the constraint *after*, 'UC1 after UC2' indicates that UC1 and UC2 are inserted into two different versions of a PL use case diagram. Besides, the version of the UC diagram in which UC1 is inserted is older than the UC diagram version in which UC2 is inserted. For instance, in the running example, the 'parking brake' function has two specialization functions: manual and automatic. With SysML, these functions are defined as a parent use case and its specializations, respectively. Considering that UC1 is assigned to 'manual parking brake' and UC2 is assigned to 'automatic parking brake'. In the PL use case diagram versions, the manual parking brake exists earlier than the automatic one, the constraint 'UC2 *After* UC1' is defined. The constraint *Before* indicates the opposite of *After*. Two use cases (functions) added to the same version of a PL use case diagram requires a *During* constraint. Constraints *inside* and *outside* indicate the use case's evolution in space. The constraint *inside* (*outside*) indicates that one block exists inside (outside) another one (in the block definition diagram). For example, in a parking brake system, an "Electronic_Sensor" may exist inside or outside "Dedicated_On_Board_Computer". A constraint *Inside* or *Outside* is useful to add this precision. 5) The risk value may be high, medium or low if a function is removed from the PL

functional requirements. This property (risk) is useful for PLs evolution. In fact, if a function with a high risk is removed from the PL functional requirements then the PL's quality is affected. However, if a function with a medium risk value is removed then the quality of the PL may not be affected. Besides, if a function with a low risk is removed from the PL then the PL's quality is not affected. Consequently, we can take the risk of removing it. 6) The binding time property of a PL functional requirement represents the point in time when the function is practically included inside the PL system. Binding time values may be design time, implementation time, link time, load time or run time. 7) Visibility property indicates that PL functional requirements can be visible or hidden for the application engineer or the customer who has the role to derive products. If a function, such as a use case, is visible then the application engineer can decide to select it or not for the product to derive. However, if a function is hidden then it can be seen only at the domain level of the PL. For example, functional requirements, which are technical, are usually hidden for non-expert users.

Dimension	**Properties**
Functional	– Optionality (optional, mandatory) – Variation (variation point and variant) – Dependency (OR, XOR, AND) – Priority (S, M, L, XL) – Relationship (is-a, composition, sharable aggregation) – Constraint (include, exclude, inside, outside, before, after, during) – Risk (high, medium, low) – Binding time (Design time, Implementation time, Link time, Load time, Run time) – Visibility (hidden, visible).
Quality Attribute	– Optionality (optional, mandatory) – Level (low, medium, high) – Binding time (Design time, Implementation time, Link time, Load time, Run time) – Risk (high, medium, low) – Relationships (influence, require)
Architectural	– Optionality (optional, mandatory) – Relationship (composition, sharable aggregation, specialization, association) – Constraints (use, ordered, frozen, etc.) – Visibility (hidden, visible) – Risk (high, medium, low)
Contextual	– Optionality (optional, mandatory), – Variation (variation point and variant) – Sensor (hard, soft) – Risk (high, medium, low) – Visibility (hidden, visible)

Table 1: Variability dimensions

The "Quality attribute" dimension represents variability of quality for requirements and architecture. We use the term quality attribute to express software non-functional properties like availability, security, dependability, performance, among others. A quality attribute is defined by the properties: optionality, level, binding time, risk and relationships. The properties optionality, binding time and risk have the same definitions as those presented for functional dimension. A quality

attribute (QA) can have a *high*, *medium* or *low* level. For example, a parking brake system can have a high security but its performance may be medium. The property relationships defines what relates the quality attributes of a PL. In fact, a quality attribute can influence or require other ones. For example, the security of the parking brake system includes its integrity. But, its maintainability influences its safety.

The "Architectural variability" dimension is defined by the properties: Optionality, Relationship, Dependency, Constraints and Visibility. The Properties optionality, visibility and risk are described in the above sections. Relationship property indicates that components of the PL architecture can be related by composition, sharable aggregation, specialization relationships or an association. The property constraints indicates the conditions that the architecture components should satisfy. The constraint use is defined for two components to indicate that a component uses temporarily an instance of another component at the execution time. Frozen constraint applied on a component or a relationship between two components indicates that the instance of the component or the relationship is not changeable. Ordered constraint applied on a component of the architecture indicates that the component instances should be organized according to their instantiation order.

Contextual variability dimension defines the properties Optionality, Variation, Sensor and Risk. Sensor property can have the values hardware or software. In fact, for a derived product, the application engineer can choose a soft or a hard sensor. Optionality indicates that a context element can be mandatory or optional, such as an electric sensor. Variation, risk and visibility properties have the same semantic as described in above sections. However, they are applied on the context elements such as the sensors.

The holistic variability model we propose can be used to extend dedicated or orthogonal variability models to provide more rigorous variability modelling by considering various key-facets. In the following section, we present a SysML profile extending SysML to model SSPL by considering the holistic variability model.

4 SysML4SSPL profile

SysML4SSPL extends the SysML UML profile to represent Software and Systems Product Lines (SSPLs) according to the holistic variability model. SysML4SSPL is a profile including stereotypes, tagged values and constraints associated to variability dimensions, properties and their values defined in the holistic variability model. SysML4SSPL profile (see figure 2) is defined by five packages. Each package is a SysML view package. The variability view contains the properties defined inside the holistic variability model, which represent the intersection of all its dimensions. The four other views are associated to the holistic variability model dimensions. The profile views are described in the following sections. For each view, we describe its stereotypes and then we apply it on the running example.

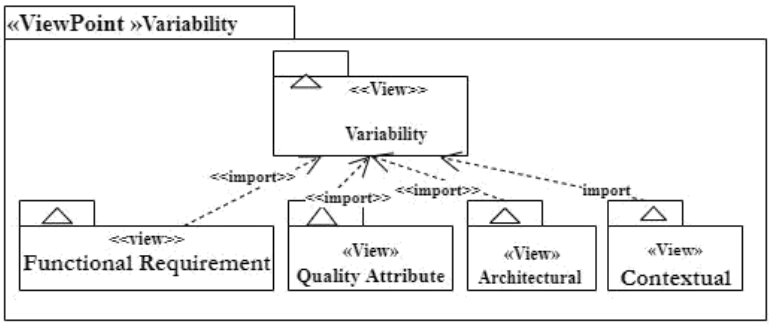


Figure 2 : SysML4SSPL profile

4.1 Variability view

Variability view package (see figure 3) encompasses stereotypes associated to properties included inside the intersection of all the variability dimensions. It includes stereotypes associated to

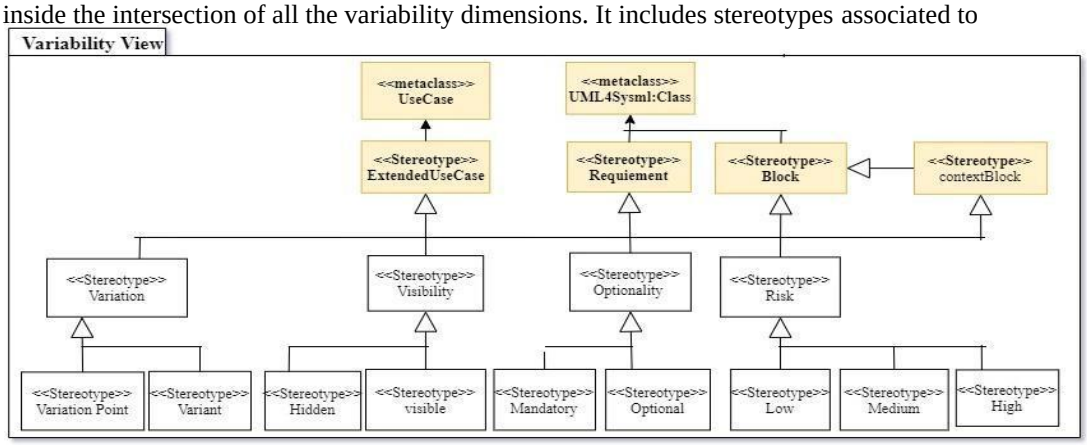


Figure 3: Variability view

optionality, variation, risk and visibility properties values. These stereotypes are «Mandatory», «Optional», «Variation Point», «Variant», «high risk», «medium risk», «low risk», «hidden» and «visible» (see Figure 3). They specialize the SysML stereotypes ExtendedUseCase, Block, Requirement, contextBlock. ExtendedUseCase is an extension of meta-class *UseCase*. Block and Requirement are extensions of meta-class C*lass* and *contextBlock* stereotype is a specialization of *block* stereotype.

4.2 Functional view

With SysML, functional modelling of systems can be defined by using use case diagrams. The functional view (see Figure 4) includes stereotypes extending 'Use Case' and 'DirectedRelationship' meta-classes. For a use case, stereotypes associated to its priority are «S», «L», «M» and «XL». Besides, a use case can have a biding time, a stereotype is defined for it. Values of the property 'binding time' are defined as tagged values. An extended relationship between two use cases is specialized to define a dependency, a constraint and a relationship. Dependency stereotypes are «And», «Or» and «Xor». Relationships stereotypes are «composition» and «Aggregation». Constraint stereotypes are «Inside», «Outside», «Include», « Exclude», «After», «Before» and «During».

For use cases, stereotypes associated to their Optionality, Risk, Visibility and Variation properties are already defined inside the variability view. The value 'is-a' associated to the Relationship property, which is defined inside the functional view in table 1, is already defined by SysML generalization relationship between use cases. To this end, we do not define a stereotype for it.

Extending the SysML use case diagram to model SSPLs involves using profiles of Variability and Functional views. In fact, inside these two views model elements of a use case diagram are extended to handle variability. Figure 5, shows a fragment of the extended use case diagram associated to the running example of PL, which is 'Electric Parking Brake System'.

Stereotypes defined inside the fragment of the extended use case diagram consider variability modelling of use cases and dependencies. In fact, use cases 'Manage Hardware' and 'Manage Software' are stereotyped with three stereotypes: 1) «XL», to indicate their high priority, 2) «Mandatory», to indicate that they are compulsory in any use case diagram of a derived EPB product and 3) «VariantPoint», to represent a parent use case. The use case 'Parking Brake Service' is mandatory, but its realization time is arbitrary.

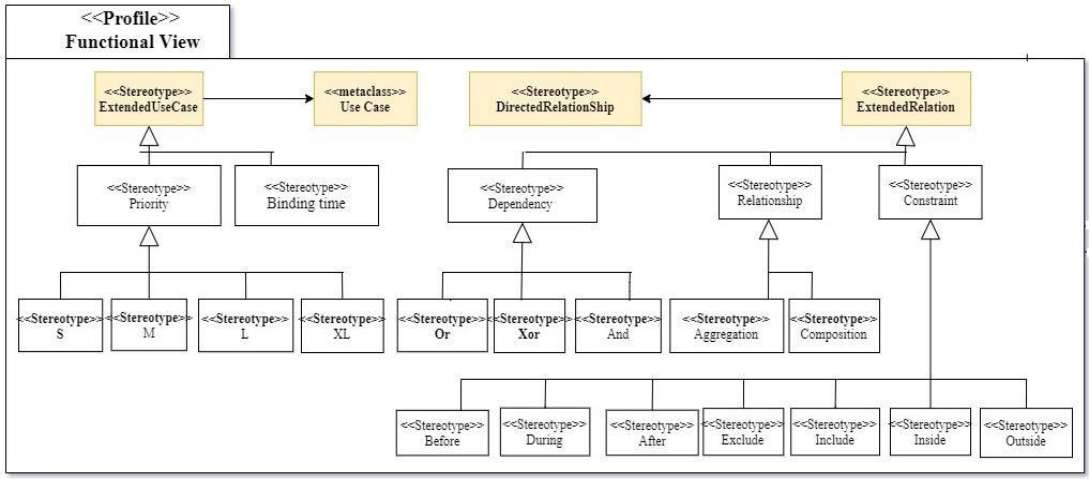


Figure.4. Functional view

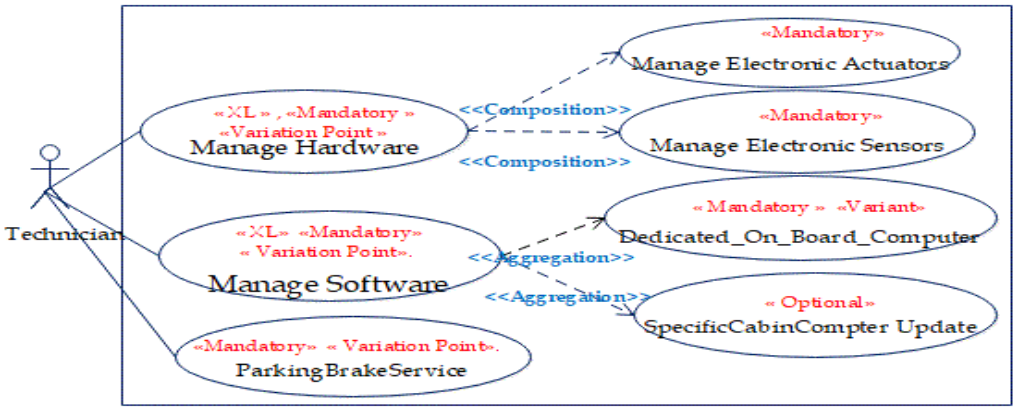


Figure 5: Extended Use Case Diagram for EPB product line

The use case ‘Manage Hardware' is composed of («composition» stereotype) two use cases, which are ‘ManageElectronicActuators' and 'Manage Electronic Sensors’. However, use case 'Manage Software' has a sharable aggregation with its two variants 'Dedicated_On_Board_Computer' and 'SpecificCabinCompter Update’.

4.3 Quality View

Inside SysML models, Quality variability properties can be assigned to Block and Requirement model elements. Then, we defined the two stereotypes «QualityBlock» and «QualityRequirement» to specialize these SysML model elements. For each quality element (requirement or block) a level is assigned and the level value is defined by «high», «medium» and «low» stereotypes (see figure 6). The property ‘Binding time’ is defined as a stereotype and its values are defined as tagged values.

Extensions defined inside the variability and quality views can be applied on the SysML Block Definition Diagram (BDD) or the Requirement diagram. In Figure 7, we present an extension of the SysML BDD associated to the PL 'Electric Parking Brake System'. This extension of BDD is restricted because it considers just stereotypes related to quality blocks.

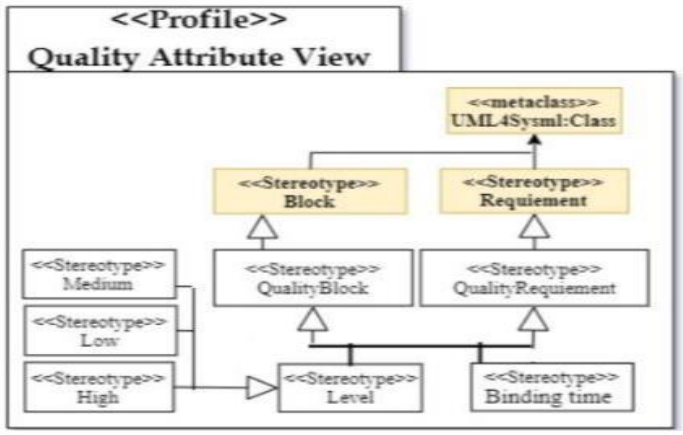


Figure 6: Quality Attribute view

Figure 7, shows an example extending a quality block associated to the running example. The Electronic Parking Brake block has a high level of security, which is defined by a security QualityBlock.

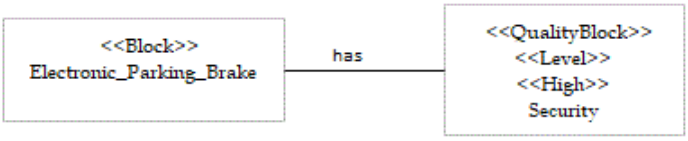


Figure 7: Extended Quality Block

4.4 Architectural View

With SysML, architectural modelling of systems can be defined by using Block Definition Diagrams (BDD). Therefore, the architectural view (see Figure 8) includes stereotypes extending the block diagram elements. Stereotypes extending the Block model element are already defined inside the variability view package. In the package of figure 8, we define the stereotype «constraint» to extend Dependency meta-class. Besides, we define stereotypes «After», «Before», «During», «Inside», «Outside», «Include» and «Exclude» as specializations of «constraint».

The property relationship defined inside the architectural dimension (shown in table 1) does not need SysML stereotypes. Because, the proposed relationships are already defined by SysML.

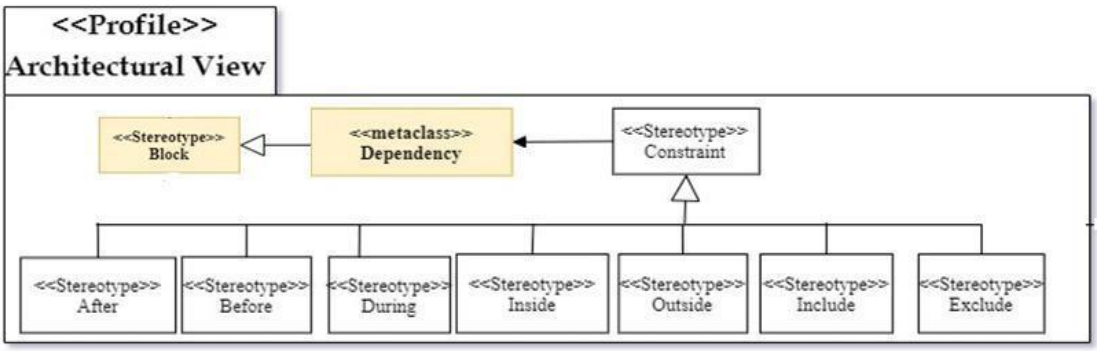
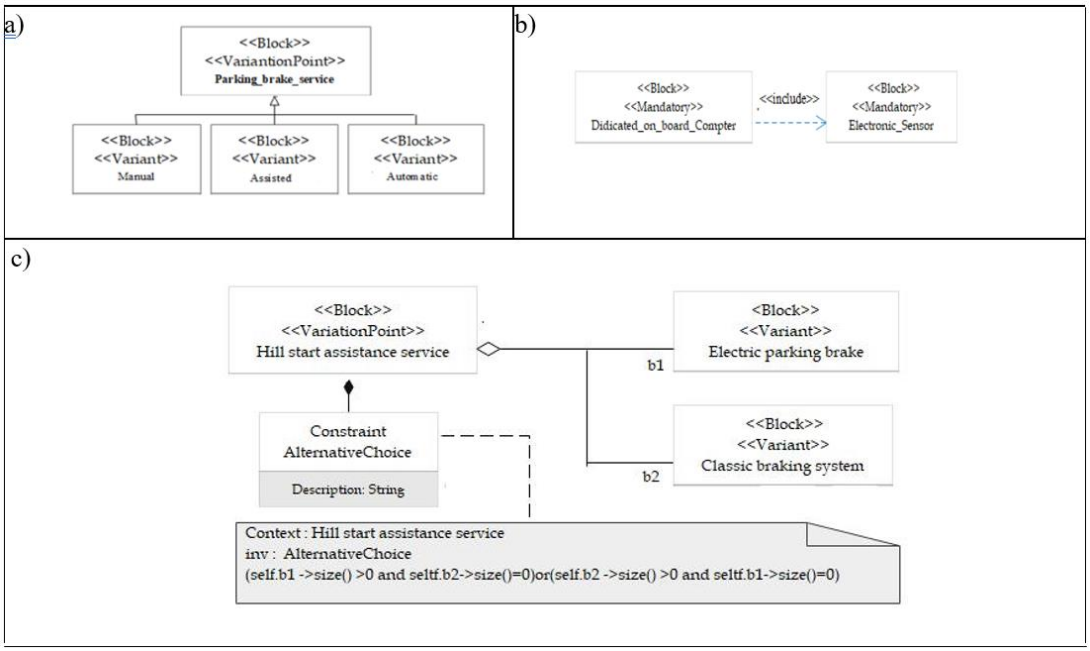


Figure 8: Architectural view

In Figure 9, we present an extension of the BDD associated to the Product Line (PL) of the running example. This extension is based on variability and architecture profiles (views) and it is applied on the EPB product line. Fragment a) indicates an extension of a Block model element with variation values. In fact, the parent Block is stereotyped "VariationPoint" and its specializations are stereotyped "variant". Fragment b) indicates a dependency between two blocks, namely «include». Fragment c) represents a sharable aggregation between a variation point block and its variants. To indicate that for a derived product only one variant block should be selected, we defined the constraint *AlternativeChoice* with OCL language. Fragment d) represents a sharable aggregation between a variation point block and its variants. To indicate that for a derived product a multiple choice of variants can occur, we defined the constraint *ChoiceMultipleCard* with OCL language.



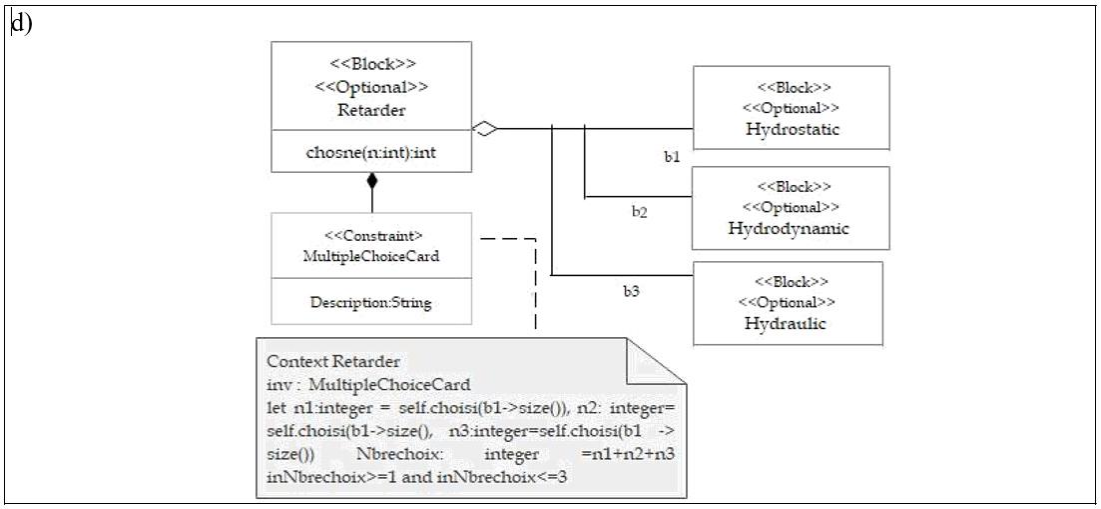


Figure 9: Fragments of an extended BDD

4.5 Contextual View

In this view, we extend the SysML Block model element. This means that the SysML BDD can be extended to consider contextual variability. In this view (see figure 10), we defined a specialization of a Block, which is the *sensor* stereotype. Two stereotypes are also defined to specialize a *sensor*, they are «Hardware» and «Software». Optionality, variation, risk and visibility stereotypes related to a contextual block are already defined by variability view.

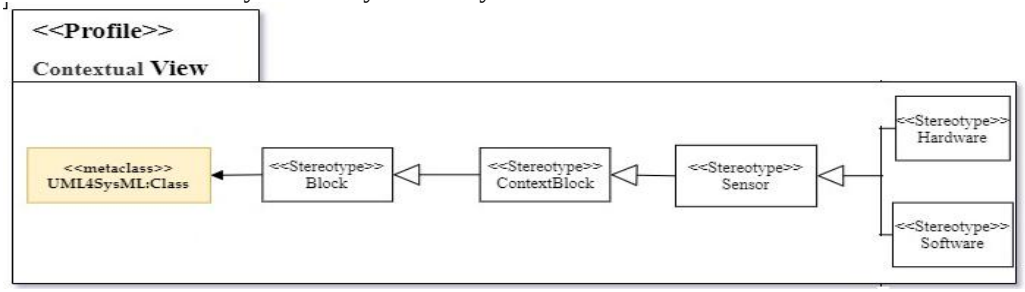


Figure 10: Contextual view

Figure 11, shows an example extending a context block associated to the running example. An architecture of an EPB (Architecture_EPB) is considered as a sensor and it is composed of two mandatory context blocks. Gearbox is a hardware and Regulatory is a software contextual block.

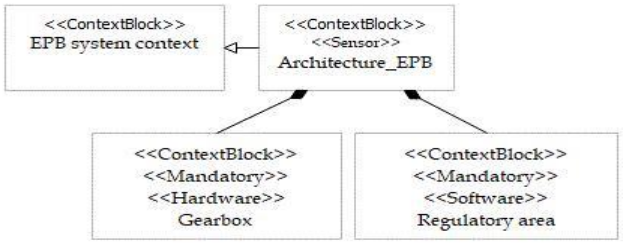


Figure 11: Extended BlockContext

The running example, which is associated to the PL ‘electronic parking brake system’ does not consider all the stereotypes defined inside the SysML packages. However, it is possible to apply them for other PLs.

5 Related work

UML and SysML have spurred multitude of works on their extension for software and systems product line modelling. Notable extensions include profiles for modelling variability of product lines.

Trujillo et al. (Trujillo, 2010) proposed extensions for SysML Block Definition, activity and state/transition diagrams. They defined a set of stereotypes to extend some SysML meta-classes such as «BlockVariationPoint», «AttributVariationPoint» and «StateVariationPoint». Stereotypes proposed by Trujillo et al. are limited to variability properties ‘optionality’ and ‘variation’, which we defined inside the variability view of our proposed profile. More stereotypes should be defined to model various aspects of variability in SysML.

Clauss (Clauss, 2001) defined extensions of UML class diagram to model variability of PL architecture. The proposed stereotypes are «optional», «variationPoint» and «variant». This work is similar to the one of Trujillo et al., but it is limited to UML.

Helouetet et al. (Helouetet, 2003) proposed extensions of use case, class and sequence diagrams to model software product lines by UML. The proposed stereotypes are «VariationPoint», «Variant», «Optional», «Virtual» and «Variation». The authors have also defined two types of OCL constraints: generic constraints and specific constraints. Generic constraints considers any software PL. However, specific constraints are specific by SPL. These constraints are used to verify consistency of derived products. The Helouetet et al. proposition is limited as well. It considers only one dimension of variability modelling, which contains basic stereotypes of optionality and variation properties.

Gaeta et al (Gaeta, 2015) proposed a methodology and a set of SysML construct extensions to model avionics product lines. Extensions are represented by two types of stereotypes: (i) optionality stereotypes for modelling variability inside some SysML model elements. Examples of the proposed stereotypes are « Mandatory_Requirement» and «Optional_ Requirement», (ii) stereotypes for including variation points in the requirements and sequence diagrams of SysML. Examples of these stereotypes are "SPL_Requirement" and "SPL_Fragment". The proposed methodology is defined to guide the modelling of Avionics product lines based on the proposed stereotypes and a set of constraints. The proposed constraints are used to maintain the consistency of the Avionics models, but they are not enough to obtain consistent products. For example, a derived product may have a software component (block) without the hardware block to which it must be assigned. In this work, as well, variability modelling inside SysML model elements are limited to ‘optionality’ and ‘variation’ properties.

Flege et al (Flege, 2000) defined optionality as the only mechanism of variability. The optionality is introduced using two stereotypes: «Optional» and «HasVariability». Stereotype «Optional» is defined for UML model elements which are optional. However, stereotype «HasVariability» is associated to non-optional UML model elements that may contain one or more optional elements (e.g., a UML package that contain one or more optional classes). This work is limited to optionality property.

In our work, we have used a different approach even if there was a myriad of works extending UML for supporting variability. First, we defined a holistic profile for Software and Systems Product Lines’s variability modelling. We extend a large variety of SysML diagrams in order to support variability. We have established 4 facets of product line variability, namely functional variability, quality attribute variability, architectural variability, and contextual variability. Variability properties and their values are represented inside SysML UML profile as stereotypes, tagged values and constraints.

6 Conclusion

In this paper, we have presented a holistic variability modelling approach for Software and Systems Product Line by extending SysML which is itself a UML profile. Four dimensions describing different facets of product line variability modelling were proposed. These are functional, architectural, contextual and quality dimensions. In fact, we have defined a set of stereotypes, tagged values and constraints to model various views of variability inside SysML. More specifically, we add profiles to : 1) use case diagrams for the functional variability, 2) block and requirement diagrams for quality variability, 3) bloc diagram for architectural variability and 4) contextual dimension. As a proof of concept, we applied the proposed SysML extensions to the 'Electric Parking Brake system' Product Line. Variability plays a very important role in managing family of products and letting more flexibility and agility for systems configurations. In the near future, we are going to apply our proposal to other real industrial product lines. We in fact, have to validate our proposal and investigate if operationally it actually improves variability handling in systems. We began also to create a framework that supports the Software and Systems Product Line construction. It also permits the tracing of different artefacts. We will also use machine learning technics in order to manage the Software and System Families evolutions. In fact, for a given system product, if we have a set of a system family product versions, a learning algorithm will be able to predict its evolution.

References

Flege, O. (December 2000). "System Family Architecture Description Using the UML" *Technical Report IESE-Report No. 092.00/E,* IESE.

Clauss, M. (2001). Generic modelling using UML extensions forvariability. *InWorkshop on Domain Specific Visual Languages at OOPSLA*.

Dumitrescu, C., Tessier, P., Salinesi, C. , Gerard, S., Dauron, A., and Mazo, R. (2014). Capturing variability in model based systems engineering. *In Complex Systems Design & Management*, pages 125–139. Springer.

https://www.omg.org/spec/SysML/1.6/.(2019)

Gaeta, J.P., Czarnecki, K. (2015). Modelling aerospace systems product lines in SysML. *In Proceedings of the 19th international conference on software product line, pages 293–302.* ACM.

http://www.splot-research.org/ (2021)

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (FODA) feasibility study. *Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.*

Helouetet, L., Ziadi, T. and Jezequel, J. (2003). Towards a UML profile for software product lines. *Proceedings of the Fifth International Workshop on Product Familly Engineering (PFE 5), Rennes/France*.

Mazo, R. , Salinesi, C. , Diaz, D. Djebbi, O. and LoraMichiels, A. . (2012). Constraints: The heart of domain and application engineering in the product lines engineering strategy. *International Journal of Information System Modelling and Design (IJISMD),* 3(2):33–68.

Pohl, K., Bockle, G. and Der Linden, F. J. (2005). Software product line engineering: foundations, principles and techniques. Springer Science & Business Media.

Mikko, R., Tiihonen, J., Tomi, M. . (2019). Software product lines and variability modelling.

Schobbens, P-Y., Heymans, P. and Trigaux, J-C. (2006). Feature diagrams: A survey and a formal semantics.

Svahnberg, M., Van Gurp, J. and Bosch, J. (2005). Taxonomy of variability realization techniques. *Software: Practice and experience,* 35(8):705–754.

Trujillo, S., Garate, J-M. , Lopez-Herrejon, R. E ., Mendialdua, X. , Rosado, A. , Egyed, A., Krueger, C-W. , and De Sosa,J. (2010). .Coping with variability in model-based systems engineering: An experience in green energy. *In European Conference on Modelling Foundations and Applications, pages 293–304*. Springer.

Bosch, J. (2013). Software product line engineering. *In Systems and Software Variability Management, pages 3–24.* Springer