Software Product Lines for the development of Families of DSLs for Robotics

Tewfik Ziadi, Tegane Saher, Selma Kchir, Serge Stinckwich

Software Product Lines for the development of Families of DSLs for Robotics

Tewfik Ziadi* — **Saher Tegane**** — **Selma Kchir***** — **Serge Stinckwich******

* *LIP6-UPMC, Tewfik.Ziadi@lip6.fr.* ** *ESI, Algeria, s_tegane@esi.dz.* *** *LIP6-UPMC, Selma.Kchir@lip6.fr.* **** *UMMISCO-IRD-UPMC, Serge.Stinckwich@ird.fr*

RÉSUMÉ.

ABSTRACT. *The development of robotics software must deal with a large amount of variability. This position paper proposes a framework based on Software Product Line Engineering to allow defining families of DSL in the context of robotics. This framework allows: 1) Managing variability when defining the domain model for a family of DSL. 2) Deriving specific DSL according the user choices.*

MOTS-CLÉS : *Lignes de produits, DSL, Robotique*

KEYWORDS: *software product lines, DSL, Robotics*

1. Introduction

Domain Specific Languages (DSLs) have emerged as a powerful mechanism for managing complexity in software development. Designing and implementing DSL is built on two main steps[MER 05] :

– Analysing the domain to capture the key concepts and their relationships. In the context of Model Driven Engineering (MDE), these concepts can be defined in what is called a *domain model* or a *meta-model*.

– Implementing related *tools*. This includes editors, compilers, and code generators.

As mentioned in [WHI 09], DSL are focused on very specific domain concerns and this narrow scope makes it hard to reuse a DSL for a new set of requirements. This is specially true in the context of robotics where the development of robotics software must deal with a large amount of variability from at least two perspectives :

– Variability in sensors and actuators. There is a large number of families of sensors and actuators. The choice of each type of sensors or actuators depending on the type of task target and/or the type of the used robots.

– For the same task we can find a family of existing algorithms, e.g. the bug family for the navigation task [LUM 90].

Defining a single DSL that considers these two kinds of variability can make it complex to be used. Indeed, including all types of sensors, actuators and existing algorithms overloads its domain model and makes it useless for end users that only use a subset of these sensors, actuators and algorithms. In addition, any introduction of new sensors and/or actuators needs an evolution of the domain model and its related tools. Therefore, we believe that to consider this challenge of variability management in robotics applications, we need not only to implement a single DSL but a family of DSLs[SVE 10].

Implementing a family of software is known in the software engineering community by the notion of *software product lines*. Software Product Line (SPL) focuses on capturing commonality and variability between several software products belonging to the same domain [CLE 01]. It introduces the notion of *feature models* to explicitly specify variability and proposes to automated what is called *product derivation*, which consists of building product members based on the selection of features from the feature model.

In this paper we propose to reuse the SPL approach to design and implement a family of DSLs for robotics application. The objective is to propose a framework that allows : 1) Specifying variability using feature models where designing the DSL family. 2) Deriving specific DSL according to the user choices and/or requirements.

The reminder of this paper is organized as follows. Section 2 presents the motivations of our work. Section 3 present our approach ans illustrates it on a simple example. Section 4 concludes this work and presents some perspectives.

2. Motivations

To illustrate the motivations of our approach, let's consider a simple example of DSL, called `SearchRescueRobotics-DSL`, in the context of search and rescue robotics. This DSL should allow specifying two types of missions :

– **Exploration**. This type of task involves robot maneuvers within a disaster area. The robot can sense the obstacles within its sensing area. To specify exploration mission, the DSL should allows defining distance sensors (e.g. ; `Lidar`, `Camera`) and motion actuators (e.g. ; `Differential Wheels`).

– **Human Detection**. The objective of the robot in this kind of missions is to detect the presence of human in a specific area. To specify human detection, the DSL should specify specific sensors related to human detection such as `Sound` and `CO2` sensors. For human detection, we also need additional actuators such as `ManipulatorArm` in order to manipulate the environement (opening doors, providing medical care for victims, ...).

These two kinds of missions may reuse a set navigation algorithms. For instance, we consider three navigation algorithms of the Bug family[LUM 90].

Figure 1 illustrates a part of domain model (meta-model) of `SearchResscueRobotics-DSL`[1]. This includes four main concepts : `Mission`, `Sensor`, `Actuator` and `NavigationAlgo`. To model the different variants of sensors and actuators that can be used to define missions, we reused inheritance. For instance, `Lidar`, `SoundSensor` that inherit from `Sensor` represent two examples of concrete existing sensors. In the same, we modeled the different navigation algorithms using inheritance. We introduced the NavigationAlgo meta class and all algorithm variants are modeled using inheritance. For instance, Bug1, Bug2, and Alg1 represent meta-classes that implement Bug family members.

In the second step, when we implement the code generator that is related to the domain model of Figure 1, we need to identify each type of the concrete sensors and actuators and generate the corresponding code. The following template [2] presents the general structure of a code generator which generates the code for sensors.

```
//generating code from sensors
for (Sensor s: sensors)
~~if (s instanceof Lidar)
~~~~print " specific code for Lidar
~~if (s instanceof SoundSensor)
~~~~print " specific code for SoundSensor
........
~~if (s instanceof CO2Sensor)
~~~~print " specific code for CO2Sensor
```

1. We have only interested on Sensors, Actuators and Navigation algorithms.
2. We follow a EMF/Java like syntax.

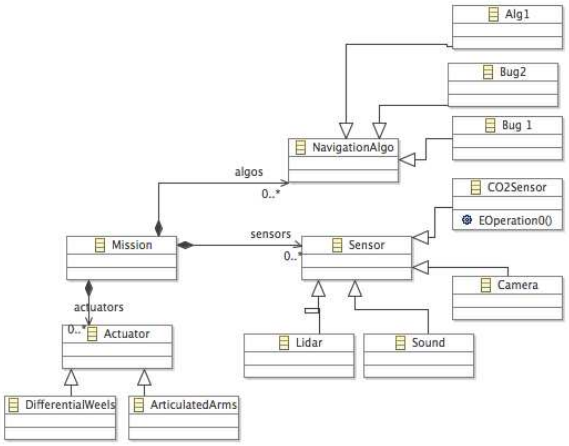

Figure 1. *An example of a domain model*

The design and the implementation of a single DSL for these two kinds of missions allows gathering the common concepts in the same domain model. However, it can give rise to some problems concerning the usability and the evolution of DSL. By including all concepts related to the both types of missions makes the domain model overladen and useless for users that only use one kind of missions. Indeed, for the users that only specify *Exploration* missions, why we include in the `SearchRescueRobotics-DSL` DSL concepts such as `CO2Sensors` or `SoundSensor` that are only specific for *Human Detection* missions ? In addition, any introduction of new concrete sensors and/or actuators needs the modification of the domain model and its related code generators. For instance, we need to modify the template below to consider new sensors and/or actuators.

In this position paper, we advocate the idea that for robotics applications, we need an approach that allows designing not a single DSL but a family of DSLs. To implement such approach, we propose to use the concepts of Software Product Line Engineering[CLE 01]. In particular, we reuse the Common Variability Language (CVL)[HAU 12] tool to implement our approach.

Next section present in detail our approach.

3. Our approach

Our aim is to propose a framework that allows designing a family of DSL rather than a single DSL in a specific domain. For robotics applications, this enables to derive specific DSL according to different variability factors. For instance, it is possible to

obtain a specific DSL for a specific kind of missions and/or for a specific types of sensors or actuators. We identified two main requirements for such framework :

– Mechanisms to specify the variability in the domain model and in its related tools.

– Mechanisms that allow the specialization of the domain model ans its related tools to a specific user choices and/or needs.

To achieve these two requirements is to reuse Software Product Line Engineering approaches. We particularly reuse a CVL (Common Variability Language)[HAU 12] approach. Next subsections presents our framework.

3.1. *Variability Management in the domain model*

The Common Variability Language[HAU 12] is a generic and separate language for modeling variability in models. The main idea is to allows defining families of models. CVL is based on three based principals :

– **The base model**. It represent the model that gathers the basic elements among the family.

– **The variability model**. It represents the core model that defines variability on the base model.

– **The resolution model**. It is a model that defines how to resolve the variability model to create a specific model in the base model.

– **The CVL transformations**. CVL proposes an engine to transform the base model according to a specific resolution model.

Our idea to manage variability in the domain model is to apply CVL on the meta-models. The base model is thus defined as the meta-model part that gathers all basic concepts in the DSL family where the variability model specifies the variability between the DSL family members. Next subsections illustrates our use of CVL on the example of `SearchRescueRobotics-DSL`.

3.1.1. *The Base Domain Model*

The base domain model is defined as the meta-model representing the basic concepts in the DSL family. Figure 2 illustrates the base domain model for the family of DSL concerning the `MobileRobot-DSL`. It only contains concepts of Mission, `Sensor`, `Actuator` and `NavigationAlg`. We have omitted all variable concepts such as concrete sensors and actuators because these concepts are variable and depend on user choices and/or needs.

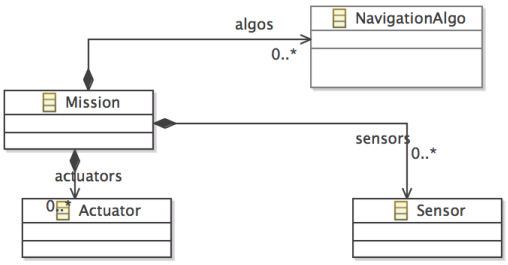

Figure 2. *The base model*

3.1.2. *The Variability Model*

The Variability Model (VM) in CVL specifies the variability among the family. To define the VM in CVL, two parts are specified : Feature Model and Fragment Substitution.

3.1.2.1. Feature Model.

The feature model (FM) in CVL is inspired from feature modeling in SPLE[KAN]. It allows specifying the variability in the User-Centric Layer. Figure 3 shows the feature model concerning the example of `SearchRescueRobotics-DSL`. In addition to root feature, we introduced two sub-features `Mission` and `NavigationAlgo` to specify that we have two variation points. For `Mission` feature, we used the âĂIJXORâĂİ operator to specify that the user have a choice between two kinds of missions : Exploration and Human Detection. The `NavigationAlgo` features specify an "OR" choice between the different algorithm variants.

3.1.2.2. Fragment Substitution

. Where the feature model is defined, the next task is to specify the modification that we want apply on the base model when a specific feature is chosen. CVL proposes the use of what is called "Fragment Substitution" to define library models. This is realized by defining two types of fragments :

– *Placement Fragment (PF)*. It consists to select the element in the base model that will be replaced when the feature is selected.

– *Replacement Fragment (RF)*. It specifies the model fragment that will replace the selected element in CFP.

Figure 4 shows the fragment substitutions for the `SeachRescueRobotics-DSL` example. For each feature we define the two types of fragments. To illustrate this, let"s consider the `Exploration` feature :

PFs definition. We defined two PFs, that are highlight in red in Figure 4, for the `Exploration` feature. The PF `ExplorationPFSensor` specifies that where the

Exploration feature is selected we need to replace the element *Sensor* in the base model of Figure2. Indeed, to allow specifying exploration mission we need to replace the `Sensor` meta-class in the base domain model to include specific distance sensors. In the same, the `ExplorationPFActuator` PF specifies that we also need to replace the `Actuator` element is the base model.

RFs definition We define for each PF an RF that specifies the model fragment that will replace the selected element. As illustrated in the Figure 4, we define `ExplorationRFSensor` as a RF associated with the `ExplorationPFSensor`. We add to the `ExplorationRFSensor`, a fragment of model that will be replace the `Sensor` meta-class where the Exploration feature is selected. This fragment model contains an inheritance hierarchy with the meta-classes `Lidar` and `Camera` that are necessary to specify exploration missions.

Figure 4 also illustrates the PFs and RFs for the `Humain Detection` feature. Here also, we replace the `Sensor` and `Actuator` meta-classes.

3.1.3. *DSL derivation*

Until now, the family of DSLs is specified, thanks to CVL, as a base domain model, a feature model and a set of fragment substitutions. From this specification, we can reuse the CVL transformation engine to derive a specific DSL from the user choices. Indeed, CVL includes a set of transformations that allows creating a new model from the base model by applying substitutions defined in CF and RPs. This derivation is defined in two steps :

– **Create a Resolution Model (RM)**. The RM can be seen as an instantiation of the feature model. It specifies the user choice and/or needs. It can be defined as a set of the selected. For instance, if we want a DSL that allows specifying only exploration mission and that supports only the `Bug1` navigation algorithm, the RM will be defined with the set : `Rm1= Exploration, Bug1`.

– **Apply CVL transformations**. From the DSL family specification and a specific RM, CVL creates a new meta-model by applying substitution rules on the base meta-model. The idea is to replace each element in the PF that concerns a selected feature in the RM with the fragment model defined in its RF.

Figure 5 illustrates the domain model automatically derived using CVL transformations for the `Rm1` resolution model.

In the same way, can derive the domain model of different DSLs according to the resolution model.

3.2. *Variability Management in code generator*

Above, we presented the use of CVL to manage varibiality in the metamodel. The second challenge is to manage variability in the code generator. In this section, we propose an approach, based on aspect oriented programming.

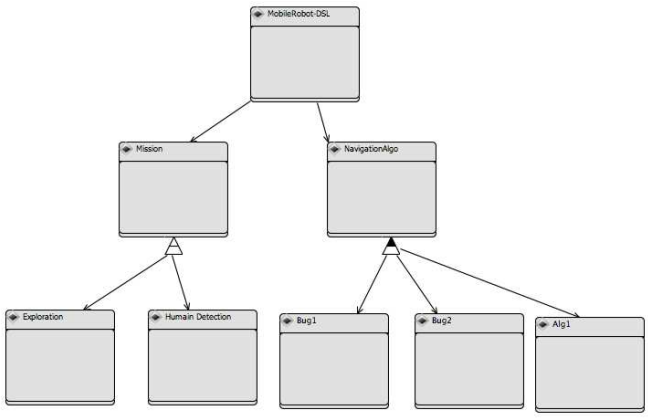

Figure 3. *The feature model*

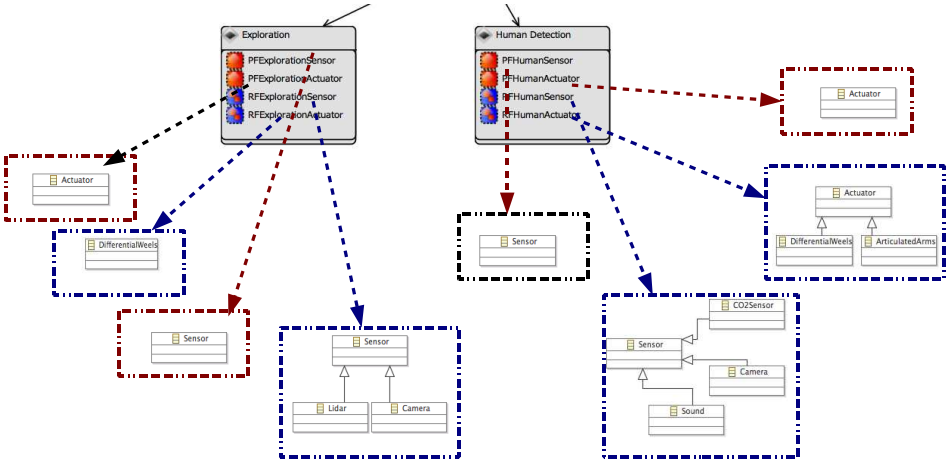

Figure 4. *A part of the variability model*

3.2.1. *Code generator derivation*

4. Conclusion and Perspectives

To handle variability in robotics application, this position paper proposes to define a family of DSLs rather than a single DSL. The idea is to allow users to derive specific DSLs according to their requirements and needs. We show that existing software product line approach can be easily reused. In this context we present a framework based on the CVL approach that allows :

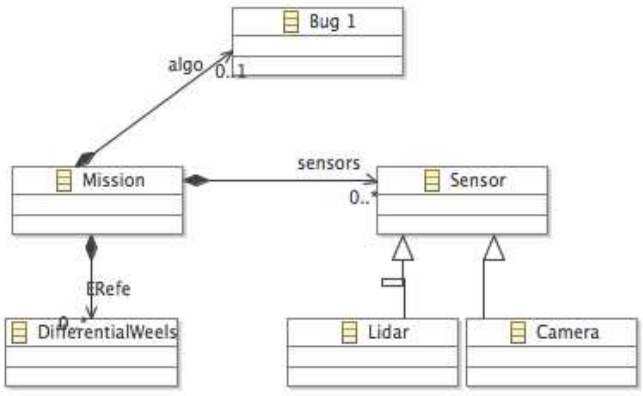

Figure 5. *The domain model of the DSL corresponding to the* `Rm1` *resolution model*

– Defining a family of DSLs. We only considered at this stage the domain model of the DSL.

– Deriving specific DSL according to user requirements.

The framework is fully implemented, thanks to the CVL tool[3]. We used a simple example to illustrate our framework. Our future work spans in managing variability in the tools related to the domain model. In particular, we aim to investigate variability management in code generators.

5. Bibliographie

[CLE 01] CLEMENTS P., NORTHROP L., *Software product lines : practices and patterns*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[HAU 12] HAUGEN Ø., WASOWSKI A., CZARNECKI K., ń CVL : common variability language ż, DE ALMEIDA E. S., SCHWANNINGER C., BENAVIDES D., Eds., *SPLC (2)*, ACM, 2012, p. 266-267.

[KAN] KANG K., COHEN S., HESS J., NOVAK W., PETERSON S., ń Feature-Oriented Domain Analysis (FODA) Feasibility Study ż, rapport.

[LUM 90] LUMELSKY V. J., SKEWIS T., ń Incorporating range sensing in the robot navigation function ż, *IEEE Transactions on Systems, Man, and Cybernetics*, , 1990, p. 1058-1069.

[MER 05] MERNIK M., HEERING J., SLOANE A. M., ń When and how to develop domain-specific languages ż, *ACM Comput. Surv.*, vol. 37, n° 4, 2005, p. 316-344.

3. `http://www.omgwiki.org/variability/doku.php?id=cvl_tool_from_sintef`

[SVE 10] SVENDSEN A., ZHANG X., LIND-TVIBERG R., FLEUREY F., HAUGEN Ø., MØLLER-PEDERSEN B., OLSEN G. K., ń Developing a Software Product Line for Train Control : A Case Study of CVL ż, BOSCH J., LEE J., Eds., *SPLC*, vol. 6287 de *Lecture Notes in Computer Science*, Springer, 2010, p. 106-120.

[WHI 09] WHITE J., HILL J. H., GRAY J., TAMBE S., GOKHALE A. S., SCHMIDT D. C., ń Improving Domain-Specific Language Reuse with Software Product Line Techniques ż, *IEEE Software*, vol. 26, n° 4, 2009, p. 47-53.