F2MoC: A Preliminary Product Line DSL for Mobile Robots

Rafael S. Durelli[1], **Vinicius H. S. Durelli**[2]

[1]Computing Departament
Federal University of São Carlos (UFSCAR)
São Carlos, SP, Brazil.

[2]Computer Systems Departament
University of São Paulo
São Carlos, SP, Brazil.

rafael_durelli@dc.ufscar.br[1], durelli@icmc.usp.br[2]

Abstract. Background: *SPL and MDD have been drawing increased attention from the software community. In the literature it is possible to find a set of articles that apply MDD techniques to assist the development of a SPL.* ***Objectives:*** *To show how to create a DSL based on a feature model.* ***Methods:*** *A SPL and MDD techniques were used to create a DSL.* ***Results:*** *A DSL was developed in order to assist the development of the mobile robots.* ***Conclusions:*** *Advantages can be gained from using the DSL: (i) an easier instantiation of SPL members; (ii) it is possible for the engineers to focus on an high level model (i.e., features model), obviating the need of dealing with platform-specific issues; (iii) source code is generated automatically from this high level model.*

1. Introduction

Software product line (SPL) and model-driven development (MDD) are two trends that have been drawing increased attention from the software development community[Czarnecki et al. 2005].

A software product line (SPL) is a set of software systems that share a common and managed set of characteristics that satisfy the needs of a particular market segment or mission, and that are developed from a common set of core assets following a planned process[P. Clements 2001]. Traditional SPL development processes identify two main technical stages: Domain Engineering, where reusable assets are developed and maintained, and the scope and production plan are defined, and Application Engineering, where particular member requirements are gathered, and the product is built by arranging the reusable assets according to the production plan.

Model-driven development (MDD) aims at capturing every important aspect of a software system through appropriate models. Compared to implementation code, models capture the intentions of the stakeholders more directly, are freer from accidental implementation details, and are more amenable to analysis. In MDD, models are not just auxiliary documentation artifacts; rather, they are source artifacts and can be used for automated code generation (model-to-code) and model transformation (model-to-model). Modeling languages play a central role in MDD. They range from the more generic modeling languages like UML (Unified Modeling Language) to the so called Domain-Specific Languages (DSLs) that is, formal languages whose constructs represent concepts from a specific problem domain.

The relationship between SPL and MDD is not new in the literature. In this way, it is possible to find articles which use MDD in the context of SPL in order to complement and assist the different aspects of a given SPL [Trujillo et al. 2007] [Avila-García et al. 2007] [Freeman et al. 2008] [Iris et al. 2007] [Polzer et al. 2009]. For instance, MDD can be used in the context of SPL to assist the application engineer in the activity of instantiation SPL's members.

This paper presents in detail the creation of a DSL, called FeatureToModelOrCode (F2MoC) to support instantiation of members of a given SPL. Our DSL comprises a metamodel[1] that represents all features of SPL feature models[2].

We have argued that several advantages can be gained from using the F2MoC: *(i)* an easier instantiation of SPL members, given that the application engineer has at his disposal all possible features belonging to the SPL; *(ii)* it also makes it possible for the application engineers to focus on an high level model (i.e., features model), obviating the need of dealing with platform-specific issues; *(iii)* source code is generated from this high level model, e.i., the underlying application source-code.

The remainder of this paper is organized as follows: Section 2 presents the foundation related to MDD and DSL, Section 3 describes the guidelines that we have developed to assist the development of a DSL based on a feature model and shows how to create a DSL named F2MoC, Section 4 gives an overview of the F2MoC, and finally in Section 5 we conclude the paper with some remarks and future directions.

2. Background

In what follows we present the main concepts about MDD and DSL.

2.1. Model-Driven Development

In Model-Driven Development (MDD) software engineer does not need to manually interact with the source code, focusing into models of higher level of abstraction. Mechanism that perform transformations are employed to generate other artifacts (source code, and/or more specific models) from the input models. These models guide the development tasks, maintenance and are also considered fundamental parts of the software as well as source code, serving as input to tools that perform automatic transformations of code, thereby reducing the efforts of developers [Stahl et al. 2006].

Domain specific Languages (DSLs) play a cornerstone role in MDD for representing models and metamodels. DSLs are small languages that present limited expressiveness focused on a particular domain[van Deursen et al. 2000]. Usually, DSLs are not turing complete and have no imperative control structures, e.g., conditions and loops. Rather, most of them are declarative, consequently, they can be regarded as specification

[1]Metamodel is a model containing elements that describe other models. Metamodels play a supportive role in MDD: they *(i)* define the abstract syntax of model languages, *(ii)* facilitate the understanding of concepts involved in the transformations, and *(iii)* are employed by the mechanism responsible for performing the transformations that map an entry-model to another model or code.

[2]A feature model is a tree based structure in which each node represents a variability point or unit of functionality in the product. The root of the tree represents the most generalized concept in the product, and successively deeper levels indicate software refinement. The parent-child relationships indicate configuration constraints that you must satisfy when choosing values for variability points.

languages. These small, declarative, special-purpose languages have a simplified suite of notations that is tailored toward their domain abstractions, features, semantics, and jargon. Hence, by using DSLs, developers perceive themselves as dealing directly with domain concepts.

DSLs are differentiated into their appearance (textual vs. graphical) and their origin (internal vs. external)[Fowler 2009]. External DSLs have their own custom-built syntax. As a consequence, developing an external DSL implies in writing a full-fledged parser in order to process it. Internal DSLs use existing general-purpose languages structures and, in most cases, the underlying execution environment, as a hosting base. An advantage of this approach is that the compiler or interpreter of the base language is reused. The main limitation is related to the limited expressiveness that can be achieved by using the base language syntactic mechanisms[van Deursen et al. 2000].

3. Methodology for the Development of F2MoC

This section briefly describes the guidelines to assist the development of a DSL called FeatureToModelorCode (F2MoC) in which aims to increase the level of reuse and accelerate the instantiation of members belonging to a given SPL. In Figure 1 depicts these guidelines. It is worth highlighting that these guidelines are based in the following frame-

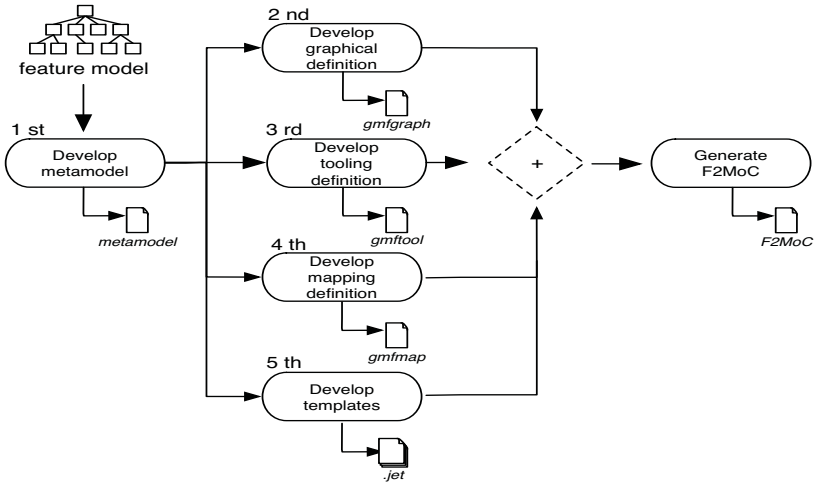

Figure 1. Guidelines to create the F2MoC

works: *(i)* Eclipse Modeling Framework[3] (EMF), *(ii)* Graphical Modeling Framework[4] (GMF) and *(iv)* Java Emitter Template[5] (JET).

As can be seen in Figure 1 these guidelines assumes that a SPL has already developed, so a SPL and consequently a feature model must be available at this point. Thus, after develop or get a feature model we need develop five different input documents, they are:

- The metamodel (*genmodel*) wich it is created based on a feature model together with some guidelines. These guidelines are the following:

[3]http://www.eclipse.org/modeling/emf/
[4]http://www.eclipse.org/modeling/gmp/
[5]http://www.eclipse.org/modeling/m2t/?project=jet#jet

  - Insert in the F2MoC's metamodel, an abstract metaclass named Diagram;
  - Insert in the F2MoC's metamodel, an abstract metaclass named DiagramElements which must have an aggregation relationship with the metaclass diagram;
  - Insert in the F2MoC's metamodel, an abstract metaclass named Features and another called RelationShip. These metaclasses must extends the metaclass DiagramElements
  - Insert in the F2MoC's metamodel, for each features that belong the feature model a metaclasse with the same name of the features. These metaclasses must extends the metaclasse Features;
  - Insert in the F2MoC's metamodel, an enumerate that represents the kind of features: mandatory, optional or alternative.
- The graphical definition model wich specifies the geometric elements that make up the graphical representation of the metamodel, such as rectangles or text labels (*gmfgraph*). This file is created using the following guidelines:
  - The submetaclasses of Features must be represented as rectangular blocks;
  - The RelationShip must be represented by line.
- The tooling definition model comprises things related to editor palettes, menus, etc. (*gmftool*). This file is created based on the following guidelines:
  - All subclass of Feature must be created a menu;
  - For each RelationShip must be create a menu.
- The mapping model links the first three models together (*gmfmap*);
- The templates are used to realized transformation. The templates must be created base on the following guidelines:
  - For each "alternative" feature the engineer must develops templates using the design pattern named Abstract Factory;
  - For each "optional" feature the engineer must develops templates using the design pattern called Builder;
  - For each "or" feature the engineer must develops templates using the design pattern called Chain of Responsibility.

In the following subsection is described how we have developed the F2MoC applied these guidelines.

3.1. Developing the F2MoC

The SPL and the feature model that we have devised and used in this paper is describe by [Durelli et al. 2010]. The domain of this SPL is mobile robots application[6]. In Figure 2 presents the feature model of which is used how the basis for the development of F2MoC. This feature model has 37 features and it can derivates approximately 17 members. Having the feature model the domain engineer must create the metamodel using the guidelines previously defined in Section 3.

In Figure 3 we present the F2MoC's metamodel which was obtained based on the guidelines previously defined. As we can see, colors were used to facilitate the visualization and the understanding of how the mapping between the feature model was performed for the metamodel. For instance, the features that represents sensors (Figure 2), i.e., the "red" ones, were represented in the metamodel by metaclasses using the same color.

[6]Mobile robots are usually equipped with contact, distance and visual sensors that enable them to perceive the environments and avoid collisions, allowing for safe autonomous navigation.

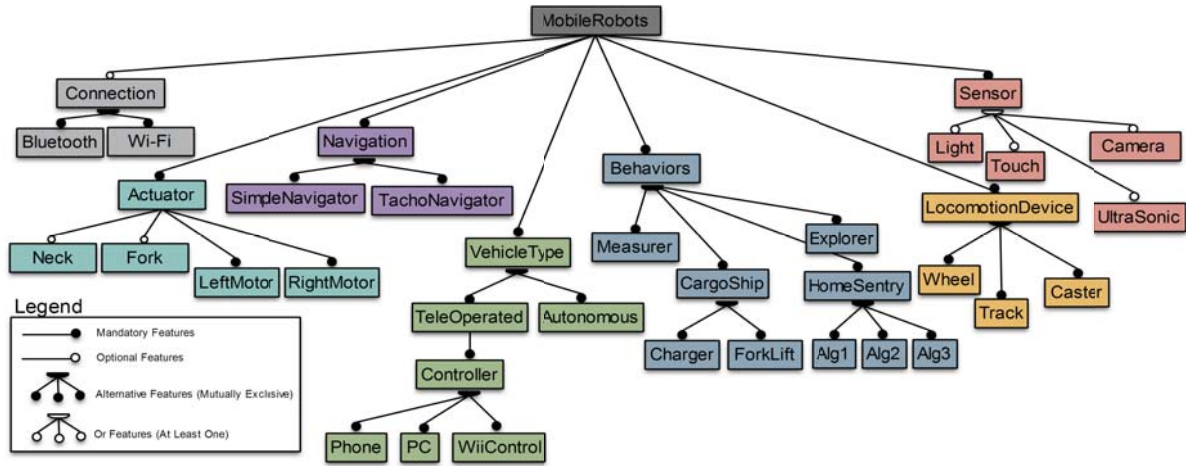

Figure 2. Feature Model of Mobile Robots

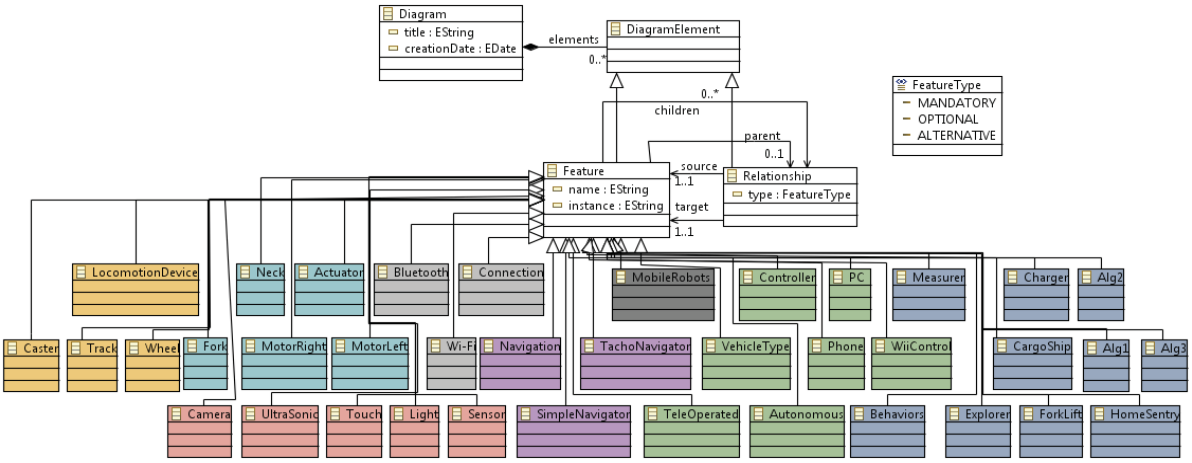

Figure 3. MetaModel's F2MoC (*genmodel*)

After developing the F2MoC metamodel, the concrete syntax of the DSL must be developed as show in Figure 1 (*graphical definition*). In Figure 4.(a) and (b) shows the file *gmfgraph*, which defines the visual aspects of the F2MoC. This file was developed using the guidelines previously defined in Section 3. For instance, all subclass of Features were represented using a "rounded rectangle" (Figure 4.(a)) and the all RelationShip were represented by a "polyline" (Figure 4.(b)).

According to Figure 1 the next file that must be developed is the *gmftool*. The Figure 4.(c) and (d) depicts how the *gmftool* was developed. As can be seen for each Features and RelationShip were created a menu, just like previously defined in Section 3. The next file developed was the *gmfmap* which is depicts in Figure 4.(e) and (f).

As can be seen in Figure 1 the fifth guideline is create the templates. Thus, we have developed such templates, in Figure 5 we show an example of the sort of code template that we have developed following the guidelines previously defined.

In this chunk of code (Figure 5), all occurrences of ***<c:get select="$f/@name">*** will be replaced by the name of all features related to sensors, e.g., *UltraSonicSensor*, *TouchSensor*, *Camera* and *LightSensor*.

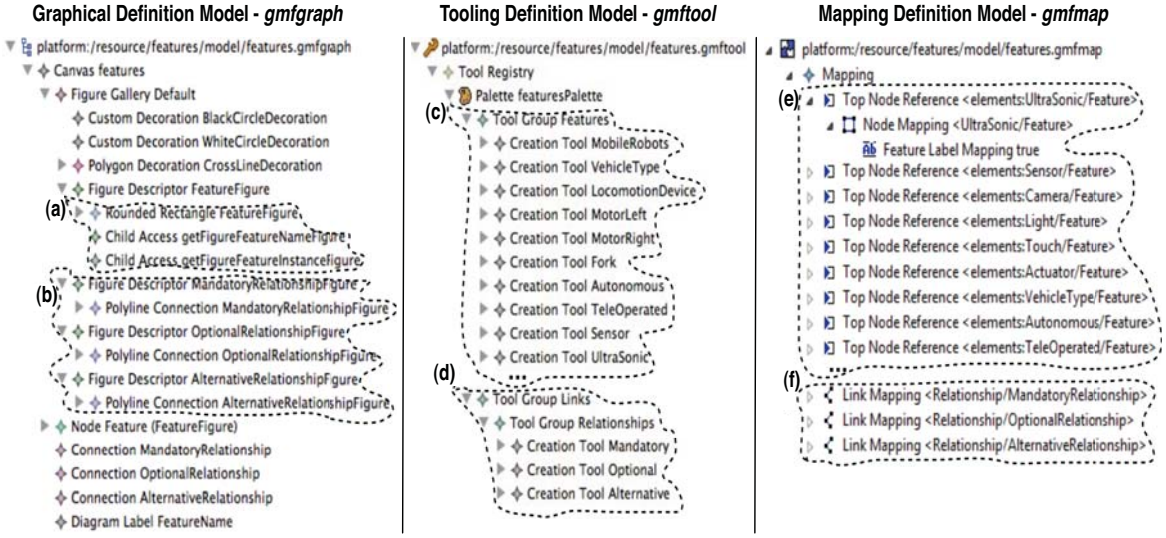

Figure 4. Arquivos necessario para o desenvolvimento da DSL

```
package com.br.ufscar.lejos.sensors.builder;

<c:iterate select="sensors/features" var="f">
   import lejos.nxt.<c:get select="$f/@name">;
</c:iterate>
import lejos.nxt.SensorPort;

public class Builder<c:get select="$f/@name"> extends
BuilderSensor{

   public void buildSensor(String sensorPort) {
         SensorPort port = null;
                  if(sensorPort.equals("S1")){
                           port = SensorPort.S1;
                  }else if (sensorPort.equals("S2")){
                           port = SensorPort.S2;
                  }else if (sensorPort.equals("S3")){
                           port = SensorPort.S3;
                  }
                  else{
                           port = SensorPort.S4;
                  }
               this.sensor = new <c:get select="$f/@name">(port);
         }
}
```

Figure 5. Chunk of template code

4. F2MoC

This section presents an overview of the F2MoC that we have developed. Figure 6 depicts an overview of instantiation of a member of SPL presented in [Durelli et al. 2010].

As we can see the F2MoC comes with a unified IDE for DSL and Java, implemented over Eclipse IDE [7]. In Figure 6.(a) depicts the view of the application engineer when he is assembling the feature models. These feature models are assembled using the action drop-and-drag of the components that are available at the palette, this palette can be seen at Figure 6.(c). Figure 6.(b) depicts all projects related to the DSL F2MoC.

Afterward assembled the feature model it is possible perform transformations model-to-code (M2C). In Figure 7 we show an example of chunck of code generated. This chunk of code has been generated based on sort of code template shown in Figure 5.

[7] www.eclipse.org

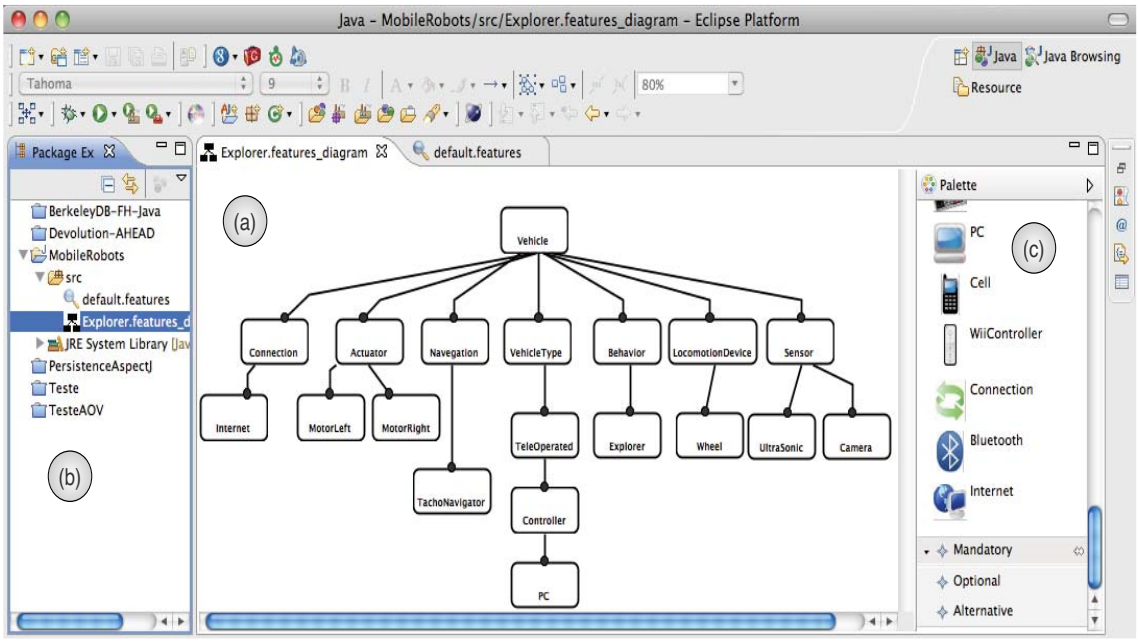

Figure 6. Overview of the F2MoC

As can be seen, all occurrences of ***<c:get select='$f/@name'>*** are replaced by the name "*UltraSonicSensor*".

```
package com.br.ufscar.lejos.sensors.builder;

import lejos.nxt.UltraSonicSensor
import lejos.nxt.SensorPort;

public class BuilderUltraSonicSensor extends BuilderSensor{

   public void buildSensor(String sensorPort) {
        SensorPort port = null;
                if(sensorPort.equals("S1")){
                        port = SensorPort.S1;
                }else if (sensorPort.equals("S2")){
                        port = SensorPort.S2;
                }else if (sensorPort.equals("S3")){
                        port = SensorPort.S3;
                }
                else{
                        port = SensorPort.S4;
                }
            this.sensor = new UltraSonicSensor(port);
        }
}
```

Figure 7. Chunk of generated code

After performing the transformation the code can be embedded in the mobile robot.

5. Concluding Remarks

Systematic techniques of reusing already established in software engineering such as SPL and MDD, have as main objective provide highest level of reuse and ease development. Several studies in the literature indicate that MDD can be used together with SPL to assist and speed up the development of SPL's members. Thus, this paper has presented step-

by-step how to build a DSL called F2MoC which provides support in the instantiation of members of an SPL by M2C transformations.

F2MoC is an straightforward development process for SPL members, through which the application engineer can assemble a SPL's member just by dragging-and-dropping features. An intrinsic advantage of F2MoC is that the feature model devised by developers is used to automatically generated source code (i.e., M2C).

Long term future work involves conducting experiments to evaluate the level of reuse provided by F2MoC and creating a repository with versioning strategies.

Acknowledgements

The authors would like to thank the financial support provided by FAPESP (grant number 2009/00632-1) and CAPES (grant number 0340-11-1).

References

Avila-García, O., García, A. E., and Rebull, E. V. S. (2007). Using software product lines to manage model families in model-driven engineering. *Proceedings of the 2007 ACM symposium on Applied computing*, 2(1):1006–1011.

Czarnecki, K., Antkiewicz, M., Kim, C. H. P., Lau, S., and Pietroszek, K. (2005). Model-driven software product lines. *ACM*, 19(5):126–127.

Durelli, R., Conrado, D., Ramos, R., Pastor, O., Camargo, V., and Penteado, R. (2010). Indentifying features for ground vehicles software product lines by means of annotated models. *ACM/IEEE International Conference on Model Driven Engineering, Languages and Systems*, 13:160–165.

Fowler, M. (2009). A pedagogical framework for domain-specific languages. *IEEE Softw.*, 26:13–14.

Freeman, G., Batory, D., and Lavender, G. (2008). Lifting transformational models of product lines: A case study. *Springer-Verlag*, 15(2):16–30.

Iris, G., Holger, P., and Markus, V. (2007). Integrating model-driven development and software product line engineering. *Proceedings of the 2007 ACM*, 3(2):120–124.

P. Clements, L. N. (2001). *Software product lines: practices and patterns*. Addison-Wesley Longman Publishing.

Polzer, A., Kowalewski, S., and Botterweck, G. (2009). Applying software product line techniques in model-based embedded systems engineering. *Communications of the ACM*, 22:2–10.

Stahl, T., Voelter, M., and Czarnecki, K. (2006). *Model-Driven Software Development: Technology, Engineering, Management*.

Trujillo, S., Azanza, M., and Diaz, O. (2007). Generative metaprogramming. *ACM*, 13(3):105–114.

van Deursen, A., Klint, P., and Visser, J. (2000). Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, 35:26–36.