Product Line Development using Multiple Domain Specific Languages in Embedded Systems

Susumu Tokumoto

Fujitsu Laboratories Limited, Software & Solution Laboratories,
4-1-1, Kamikodanaka, Nakahara-ku, Kawasaki-shi, Kanagawa, Japan
tokumoto.susumu@jp.fujitsu.com
http://jp.fujitsu.com

Abstract. In model driven development (MDD), much meaning can be given to the model using a domain specific language (DSL), and the code generation rate can be increased. Model-based product line development is possible using code generation to realize variability. In this paper, we describe the development of line tracer robots for a contest, where we achieved a high rate of code generation by using two DSLs, the characteristics of which supplement each other. Structure is described by a high generality DSL and behavior by a high specificity DSL. Furthermore, various kinds of products were able to be developed from one product line efficiently by using code generation from two DSLs to realize variability.

Keywords: Domain Specific Language, Model Driven Development, Software Product Line

1 Introduction

In recent years, due to the diversification, intensification, and complexification of user and market needs, the ability to rapidly provide various types of products has become an important competitive advantage. Software product line (SPL) is one approach to solving such issues. SPL is a method for developing a variety of products efficiently by reusing the common parts of the product line as a core asset and switching the individual parts of each products as a variation point which capability is called variability.

One technique for effective reuse is to incorporate model driven development (MDD) based on the SPL component meta model. [1]

We have worked on the effective development of a variety of products by using two or more domain specific languages (DSL) to design models, where each DSL allows different variabilities to be realized its model. A high code generation rate was achieved by gradually raising the degree of the domain specificity of the DSLs, synchronized with the progress of the development process.

2 Problems

2.1 Problem of variability over various kinds of products

In SPL, code generation is a technique for realizing variability. If variability is expressible in the model, a system can be constructed with a high tolerance for modifications and derivations.

However, in the case that source code is generated from a DSL description, if variabilities are not expressible in the DSL's specified domain, these variabilities cannot be realized by the code generation, and a different technique of realizing variability should be chosen (left side of Fig.1), e.g., structural DSL cannot realize variability of behavior.

It becomes a problem when considering many kinds of products because the variability that can be realized from one DSL is limited.

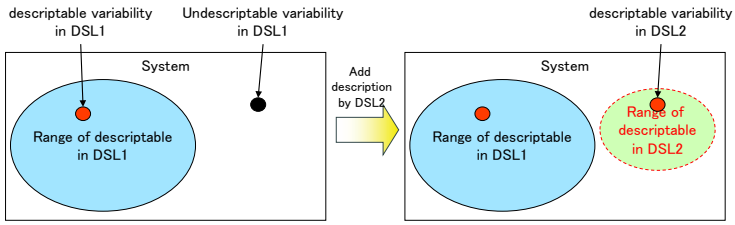

Fig. 1. Problem of variability over various kinds of products

2.2 Problem of code generation rate and generality

Generally, a DSL with low generality can achieve a high code generation rate because a lot of information of the specified domain is added to the meta model of the DSL, but projects that can use the DSL are limited to ones that suit the domain. On the other hand, a DSL for which generality is high might have a lower code generation rate because little information of the specified domain is added to the meta model of the DSL. There is a trade-off relating to the degree of code generation rate and that of domain specificity, and it is difficult to satisfy both.

3 Solutions

To solve the problems described in the preceding chapter, we propose a development process that uses two or more DSLs.

1. Define products that apply SPL, and analyze variabilities.

2. Select or design a DSL that can achieve variabilities. It is not necessary to consider all the variabilities. Raise the degree of the domain specificity of the adopted DSL as the development process advances. As a result, the code generation rate is raised.
3. Implement the DSL tool (if a new DSL was necessary).
4. Design a product with the adopted DSL while considering variabilities that remain.
5. If the product design is not complete, return to step 2.

The idea is to describe the outline with a high generality DSL, and to generate the code where the domain dependency is small in the early stage of the development process. As the development process advances, adopt a higher domain specificity DSL, and generate code which was not able to be generated at the early stage. In such a development process, by using multiple mutually supplementing DSLs, a high code generation rate is achievable and the variability of many kinds of products is expressible with tolerance for modifications. (right side of Fig.1).

4 Case study

The proposed process was applied in the development of line tracer robots for the Embedded Technology Software Design Robot Contest (ET Robot Contest). [2]

4.1 Realizing variabilities by two DSLs

In ET Robot Contest 2009, two kinds of robots, an RCX (LEGO MINDSTORMS RCX) of four-wheel type and an NXT (LEGO MINDSTORMS NXT) of two-wheel inverted pendulum type, could be selected from. We entered both an NXT team and RCX team.

There are only minor differences in the basic rules between NXT and RCX, so required functions are almost the same. Accordingly, we regarded unique parts, such as sensors and actuators, as variation points and the rest as core assets for reuse in both types of robots.

Furthermore, the contest required robots to run on both the inside and outside lanes which have different features. Therefore, we considered the running methods that suit these features as variation points.

To realize these two aspects of variation points we use two DSLs. One is a componentization DSL which realizes variability of structure; in other words, it deals with differences between RCX and NXT parts. The other is a strategy DSL which realizes variability of behavior; in other words, it deals with differences among running methods.

The relation between the DSLs and the variabilities is shown in Fig.2.

If, for example, it becomes necessary to add a new type of robot which has a different sensor and actuator, the componentization DSL makes adding it to the product line easy. Similarly if a different running method is needed, we can use the strategy DSL to ease addition of it.

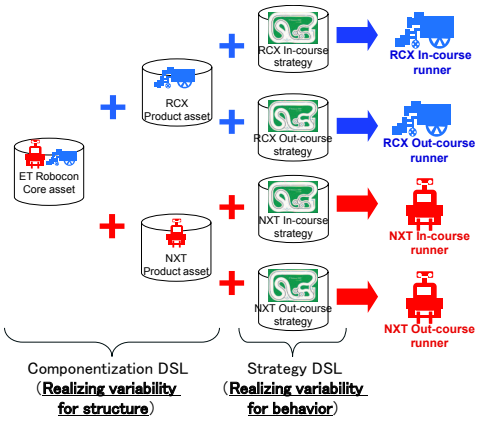

Fig. 2. Realizing variability by two DSLs

4.2 Componentization DSL and strategy DSL

The meta model of the componentization DSL consists of "component", "port", "connector", "interface" and "task". "Components" have their "ports" connected to each other by a "connector". "Ports" have "interfaces" which fix the data type. "Components" send and receive data through a "connector". "Tasks" drive "components" in the order we model.

The following is the design process with the componentization DSL.

1. Describe models of component definition.
2. Describe source code for component implementation.
3. Deploy components and connect their ports.
4. Schedule the timing for when a component is triggered during a cycle.
5. Transform models to source code with the componentization DSL tool.

The meta model of the strategy DSL consists of "actions" and "judgements". "Action" means how to drive the robot. "Judgement" judges whether the trigger of an "action" transition happened near the robot. To design the model of the strategy DSL, which is like a state diagram regarding "action" as state and "judgement" as transitions, we put "judgements" between "actions", and connect these. The strategy DSL tool transforms models to strategy data which represent relationships between "actions" and "judgements".

During the design of the architecture of the product we also considered how to design the strategy DSL. Based on this architecture, we designed the framework for reading strategy data by using the componentization DSL.

Fig. 3 shows the design process with both DSL tools.

Table 1 shows the code generation rate result as the ratio of the lines of the codes generated by the DSLs to the total LOC. We achieved a high code generation rate of 75.6% by using both DSLs compared with only the componentization DSL. Furthermore, we think that maintainability is improved by describing the model from an appropriate view by each DSL.

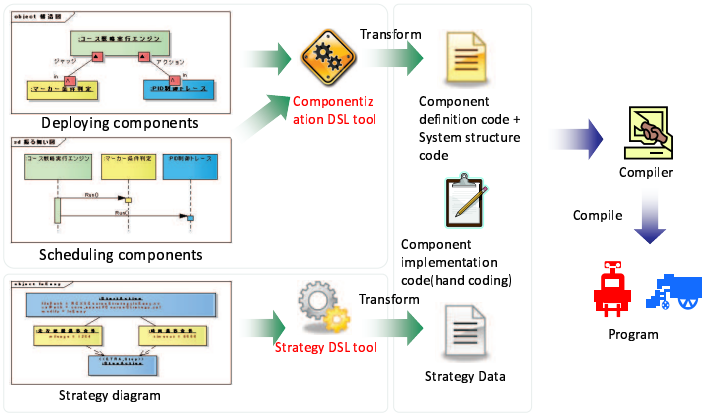

Fig. 3. Process of design by componentization DSL and strategy DSL

Table 1. LOC generated by tools and code generation rate in RCX

	generated by componentization DSL	generated by componentization DSL & strategy DSL	total LOC
LOC	4,567	5,422	7,173
Code generation rate	63.7%	75.6%	-

5 Conclusion

In this paper, we proposed a product line development process using two or more DSLs, and gave a case study of line tracer robot development to show the realization of variabilities of many kinds of products and achievement of a high code generation rate.

Future work includes verification that the process can be scaled to allow us to build large-scale systems and evaluation that dealing with multiple DSLs for variabilities can pay. The tools for multiple DSLs management might help latter problem[3].

References

1. J.Kato, Y.Goto "MDA in Product Line Engineering for Embedded Systems" Proceedings of Embedded Systems Symposium 2007, pp. 54–63, 2007 (Japanese version only)
2. ET Robot Contest, `http://www.etrobo.jp`
3. A.Hessellund, K.Czarnecki, A.Wasowski "Guided Development with Multiple Domain-Specific Languages" ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems, Nashville, Tennessee, 2007