Robots and their variability – A societal challenge and a potential solution

Thomas Buchmann, Johannes Baumgartl, Dominik Henrich and Bernhard Westfechtel
Computer Science Department
University of Bayreuth
Bayreuth, Germany
firstname.lastname@uni-bayreuth.de

***Abstract*—A robot is essentially a real-time, distributed embedded system operating in a physical environment. Often, control and communication paths within the system are tightly coupled to the actual hardware configuration of the robot. Furthermore, the domain contains a high amount of variability on different levels, ranging from hardware, over software to the environment in which the robot is operated. Today, special robots are used in households to perform monotonous and recurring tasks like vacuuming or mowing the lawn. In the future there may be robots that can be configured and programmed for more complicated tasks, like washing dishes or cleaning up or to assist elderly people. Nowadays, programming a robot is a highly complex and challenging task, which can be carried out only by programmers with dedicated background in robotics. Societal acceptance of robots can only be achieved, if they are easy to program. In this paper we present our approach to provide customized programming environments enabling programmers without background knowledge in robotics to specify robot programs. Our solution was realized using product line techniques.**

***Index Terms*—software product line; model-driven development; dsl; code generation; robot;**

I. Introduction

A robot is essentially a real-time, distributed embedded system operating in a real environment. Often, control and communication paths within the system are tightly coupled to the actual physical configuration of the robot. Furthermore, the domain contains a high amount of variability on different levels. Variability in hardware comprises, e.g., robot arms with many degrees of freedom, sensors with different resolution (e.g. a simple camera, a color camera or a depth-camera), different types of grippers which may be attached to the robot's arms, etc. On the software side there is also a high degree of variability, ranging from algorithms which take into account the hardware variability, but also which contain variability concerning different requirements like energy consumption, accuracy or execution performance. Last but not least, the actual environment in which the robot is operated may vary as well.

Nowadays, special robots are used in households to perform monotonous and recurring tasks like vacuuming or mowing the lawn. Currently we observe an increasing number of such devices. In the future there may be robots that can be configured and programmed for more complicated tasks, like washing dishes or cleaning up or to assist elderly people.

Currently, programming a robot is a highly complex and challenging task, which can be carried out only by programmers with dedicated background in robotics. Societal acceptance of robots can only be achieved, if they are easy to program. Nowadays, robots need to be programmed because they do not show the required autonomy to understand when and how certain tasks have to be performed. In this paper we present our approach to provide customized programming environments enabling programmers without background knowledge in robotics to specify robot programs. The solution we propose in this paper is based on model-driven software development and software product line engineering.

Software product line engineering [1] deals with the systematic development of products belonging to a common system family. Rather than developing each instance of a product line from scratch, reusable software artefacts are created such that each product may be composed from a collection of reusable artefacts – the platform. Commonalities and differences among different prodcuts may be captured in a *variability model*. Commonly, *feature model*s [2] are used for this purpose. Binding variability (by selecting or deselecting the respective features) in *feature configurations* describes the characteristics of particular products.

Model-driven Software Engineering (MDSE) [3] puts strong emphasis on the development of high-level models rather than on the source code. Models are not considered as documentation or as informal guidelines how to program that actual system. In contrast, models have a well-defined syntax and semantics. Moreover, MDSE aims at the development of *executable* models. The Eclipse Modeling Framework (EMF) [4] has been established as an extensible platform for the development of MDSE applications. It is based on the Ecore metamodel which is compatible with the OMG Meta Object Facility (MOF) specification [5]. Ideally, software engineers operate only on the level of models such that there is no need to inspect or edit the actual source code, which is generated automatically from the models.

II. Contribution

In this section we present our contribution. We sketch, how product line technology and model-driven software engineering is used to create a family of configurable development environments for easy robot programming.

A. Overview

As stated in the previous section, the robotics domain includes a large amount of variability, ranging from variability in hardware over variability in software to variability within the environment the robot is operating in. Furthermore, programming a robot is a very challenging task, which can not be achieved without a dedicated background in robotics. To reduce the complexity and make programming robots easier and even possible for non-programmers, end users will only interact with a domain-specific language (DSL), which abstracts from all underlying tasks like scene detection, planning algorithms, etc.

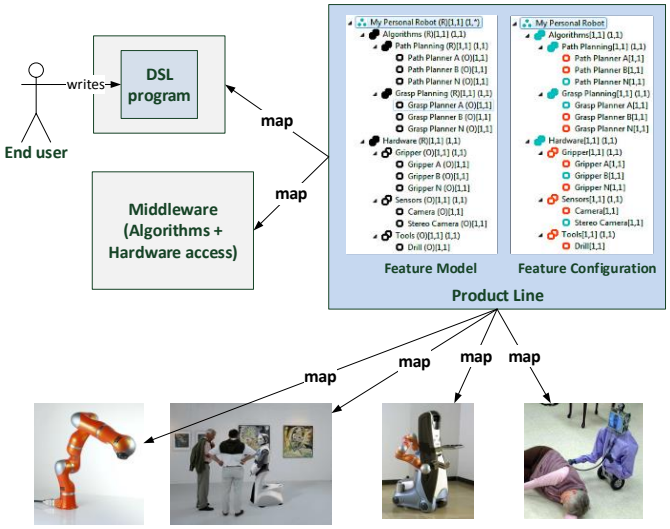

Fig. 1: Conceptual overview of our contribution.

Figure 1 depicts the main building blocks of our contribution. The end user writes programs in a special DSL. The code generated from the DSL programs uses a special middleware. It contains all neccessary algorithms for environment perception, planning, execution, and communication with the hardware. The high amount of variability which is contained in all of the aforementioned artefacts is managed with the help of product line technology. Feature models are used to capture commonalities and differences. E.g. a certain algorithm implementation may differ with respect to accuracy, safety constraints, or energy consumption. Varying hardware, like absence or presence of certain sensors may also affect the actual algorithm implementation. Furthermore, the language the end user is interacting with should not contain any fragments, that could lead to code which can not be performed on the attached hardware [6].

B. Technical realization

In this subsection we provide insights into the current technical realization of our approach.

From the software product line perspective, the main challenge is the fact that we have to address variability on different layers of abstraction and even on different meta layers. As we want to provide a development environment which is tailored to the used hardware (i.e. the robot), the programming language, i.e. the textual DSL has to be configured accordingly. This is achieved, by using FAMILE on the DSL's artefacts, like the Xtext[1] grammar file, the underlying Ecore model describing the language's abstract syntax tree (AST) and the Acceleo[2] code generation template files. Please note that these artefacts are heterogeneous, i.e. each of them is based on a different metamodel. The only thing which is common to all of them is that the respective metamodels are based on Ecore. FAMILE provides support for managing heterogeneous product lines [7].

Figure 2 depicts on the right-hand side a cut-out of the Xtext grammar file for the DSL. In the middle, the mapping model is shown, while the corresponding feature configuration is located on the left-hand side. In case of a simple path planning algorithm, the DSL parser rule for the place operation should not contain any parameters, while with an *advanced* planning algorithm the parameter should be present. The specified feature configuration only uses the simple one, thus the assignment in the Xtext grammar is suppressed.

Access to the hardware is provided by a C++ library which also encapsulates the implementation of different algorithms used for path planning, grasp planning, collision detection, scene perception, etc. The library implementation also constitutes a superimposition of all variants. In its current state, we generate specific cmake makefiles before we invoke the C++ compiler. Currently, we are working on providing a C++ discoverer for the MoDisco framework. Like the already existing Java discoverer, it will allow to parse arbitrary C++ source code files into a corresponding Ecore-based model representation. Once the new discoverer is completed, FAMILE is able to map features to C++ source code fragments directly. Figure 3 gives an overview about the different models and tools involved in this project.

Currently the project is still under development, but the experiences with FAMILE in terms of addressing and managing variability of the different software artefacts at different meta levels and levels of abstraction are very promising. We are planning to test the resulting DSLs with undergraduate students in order to get results on their usability for standard robotic tasks (e.g. sort, pick and place, peg in hole, etc.).

III. Example

In this section we outline a prototype *Personal Robot* that operates based on the proposed DSL [8]. The used hardware is a common small industrial robot with a multi degree of freedom gripper and a depth camera mounted on the robot's wrist. The program loads the static environment and initializes two sensors that observe a certain volume of the robot's workspace and the robot itself.

The program's task is to pick all objects detected inside of the first sensing volume and place them on the tray located

[1] http://www.eclipse.org/Xtext
[2] http://www.eclipse.org/acceleo

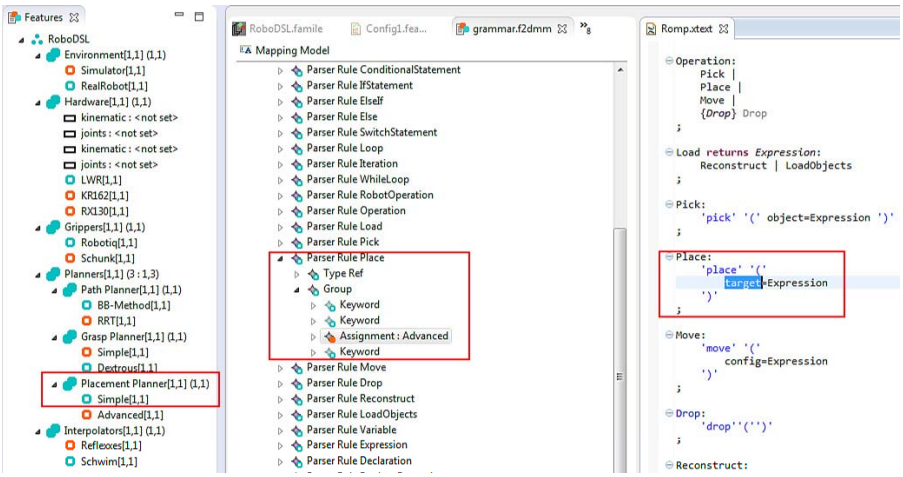

Fig. 2: Cut-out of the mapped Xtext grammar file.

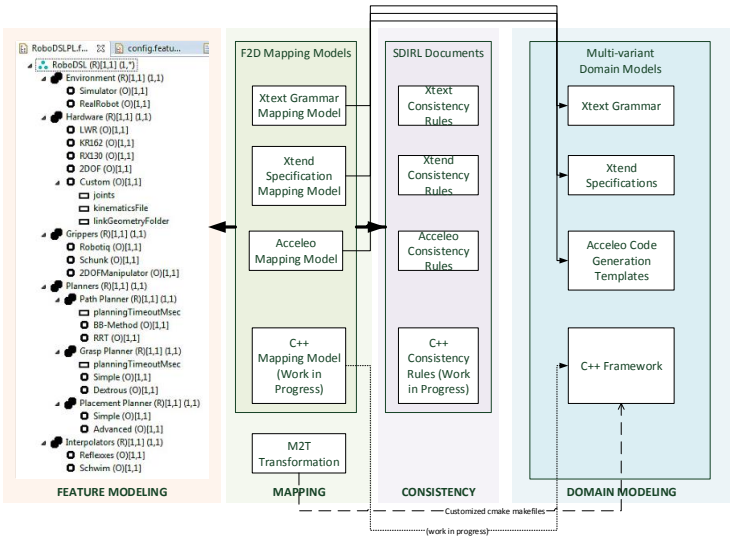

Fig. 3: Conceptual overview about the usage of FAMILE in the product line for DSLs for robot programming.

inside the second sensing volume. The task includes a lot of challenging problems that relate to each other. First the objects that should be manipulated must be reconstructed using the physical depth sensor. Considering that a solution for the *next best view problem* has to be found. A feasible task-constrained grasp has to be planed for every reconstructed object and the corresponding placement pose [9]. Furthermore, the program has to plan a collision-free path in-between the grasp and placement pose.

Regarding the variability of hardware, the inputs and outputs of all building blocks have to match, especially after a hardware change. For example the object representation reconstructed by the sensor has to match with the other algorithms. The grasp planner has to correspond with the gripper and object representation. Those constraints and dependencies are covered by the product-line approach. Hence, a hardware configuration change affects the middle-ware and the configured DSL. The configured DSL only contains language elements which describe operations that can be performed on the attached hardware.

Going into the internals of a robotic program a crucial question is: What is about the order of execution? The program has to ascertain this order. Additionally, all exceptions during the execution have to be considered and resolved. At the DSL tier a selection of an object by a certain property has to regarded. The order of execution of the algorithm is also to decide considering for example reachability, task and time constraints. In our example (igure 4) the order of execution is large objects first, since small objects can be placed on the tray between the larger ones more easily.

Our proposed DSL covers the mentioned problems, enabling the programmer to focus on the task itself. Considering that, the DSL has functionalities for loading the environment and known objects, sensors that observe a certain volume (Figure 4a), a robot and methods to instruct the robot to grasp (Figure 4b), place (Ffigure 4c), and drop an object.

(a) (b) (c) (d)

Fig. 4: Snapshots of the example program: (a) sensor based reconstruction, (b),(c) automatic order of execution, evaluation of the object, picking and placing, and (d) the result of the completed program.

Additionally, some easy to use conditions to select objects according to their properties like color, volume or size and loops are included.

IV. Related Work

Only recently model-driven software development techniques have been applied to the robotics domain. While most approaches [10], [11], [12] aim at supporting developers of robotics algorithms, only a few address higher level DSLs [13], [14]. So far, there is no approach, that aims at supporting non-programmers in specifying programs for robot tasks. In his PhD thesis [15] Gherardi presents an approach for variability modeling and resolution in component-based robotics systems. It differs from our approach in terms of the different layers of abstraction and also meta-layers where variability is addressed. Furthermore, the toolchain we use for software product line development follows an established development process.

V. Conclusion and Future Work

In this paper, we presented an approach that is dedicated to provide customized programming environments enabling programmers without background knowledge in robotics to specify robot programs. To cope with the high amount of variability which is present in the robotics domain, we employ a software product line approach. Using our sophisticated tool FAMILE, we are able to map features to the heterogeneous artefacts of our product line, like language grammars, models, validation constraints, code generation templates and source code and derive concrete products from it. We showed an example configuration for a robot with a gripper and a corresponding DSL that empowers the end user to specify programs for pick and place tasks.

We showed that our concept provides a significant improvement in developing robot programs. Since programming robots nowadays is very difficult, this problem needs to be solved if robots should be used in future to assist people in their everyday tasks. Future work on our solution comprises research on DSLs which can also be used by non-programmers. Furthermore, a hardware protocol is needed which works similarly to a plug-and-play mechanism. Pluggable hardware components like sensors, cameras, grippers, etc. can then be used to bind variability in feature configurations automatically.

References

[1] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Boston, MA, 2001.

[2] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University, Software Engineering Institute, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.

[3] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[4] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF Eclipse Modeling Framework*, 2nd ed., ser. The Eclipse Series. Boston, MA: Addison-Wesley, 2009.

[5] OMG, *Meta Object Facility (MOF) Core Specification*, OMG, January 2006, version 2.0.

[6] J. Baumgartl, T. Buchmann, D. Henrich, and B. Westfechtel, "Towards easy robot programming: Using dsls, code generators and software product lines," in *Proceedings of the 8th International Conference on Software Paradigm Trends (ICSOFT 2013)*, J. Cordeiro, D. Marca, and M. van Sinderen, Eds. ScitePress, Jul. 2013, pp. 548–554.

[7] T. Buchmann and F. Schwägerl, "A Model-driven Approach to the Development of Heterogeneous Software Product Lines," in *Proceedings of The Ninth International Conference on Software Engineering Advances (ICSEA 2014)*, IARIA. IARIA XPS Press, October 2014, accepted for publication.

[8] T. Buchmann, J. Baumgartl, D. Henrich, and B. Westfechtel, "Towards a domain-specific language for pick-and-place applications," in *Proceedings of the Fourth International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob 2013).*, U. P. S. Christian Schlegel and S. Stinckwich, Eds. arXiv.org, 2013.

[9] J. Baumgartl and D. Henrich, "Gpu-based power-grasp and placement planners for unknown environments," *Joint 45th International Symposium on Robotics - ISR 2014 and 8th GERMAN Conference on Robotics - ROBOTIK 2014*, 2014.

[10] A. Steck, D. Stampfer, and C. Schlegel, "Modellgetriebene Softwareentwicklung für Robotiksysteme," in *AMS*, 2009, pp. 241–248.

[11] S. Dhouib, S. Kchir, S. Stinckwich, T. Ziadi, and M. Ziane, "Robotml, a domain-specific language to design, simulate and deploy robotic applications," in *Simulation, Modeling, and Programming for Autonomous Robots*, ser. Lecture Notes in Computer Science, I. Noda, N. Ando, D. Brugali, and J. Kuffner, Eds. Springer Berlin Heidelberg, 2012, vol. 7628, pp. 149–160.

[12] A. Bubeck, F. Weisshardt, T. Sing, U. Reiser, M. Hagele, and A. Verl, "Implementing best practices for systems integration and distributed software development in service robotics - the care-o-bot robot family," in *System Integration (SII), 2012 IEEE/SICE International Symposium on*, 2012, pp. 609–614.

[13] U. P. Schultz, D. J. Christensen, and K. Stoy, "A Domain-Specific Language for Programming Self-Reconfigurable Robots," in *Workshop on Automatic Program Generation for Embedded Systems (APGES)*, 2007, pp. 28–36.

[14] J. F. Inglés-Romero, A. Lotz, C. V. Chicote, and C. Schlegel, "Dealing with Run-Time Variability in Service Robotics: Towards a DSL for Non-Functional Properties," in *Proceedings of the 3rd International Workshop on Domain-Specific Languages and models for ROBotic systems*, E. Menegatti, Ed., Tsukuba, Japan, 2012.

[15] L. Gherardi, "Variability modeling and resolution in component-based robotics systems," Ph.D. dissertation, University of Bergamo, 2013.