Towards A Domain-specific Language For Pick-And-Place Applications

Thomas Buchmann[1], Johannes Baumgartl[2], Dominik Henrich[2] and Bernhard Westfechtel[1]

Abstract—**Programming robots is a complicated and time-consuming task. A robot is essentially a real-time, distributed embedded system. Often, control and communication paths within the system are tightly coupled to the actual physical configuration of the robot. Thus, programming a robot is a very challenging task for domain experts who do not have a dedicated background in robotics. In this paper we present an approach towards a domain specific language, which is intended to reduce the efforts and the complexity which is required when developing robotic applications. Furthermore we apply a software product line approach to realize a configurable code generator which produces C++ code which can either be run on real robots or on a robot simulator.**

I. INTRODUCTION

A robot is essentially a real-time, distributed embedded system. Robot systems consist of different hardware components and different sensors which results in a very complex and highly variable system architecture. Often, control and communication paths within the system are tightly coupled to the actual physical configuration of the robot. As a consequence, these robots can be assembled, configured, and programmed only by experts. While this is the state of the art for robot programming nowadays, it is evident that using model-driven software engineering, and domain specific languages in particular, could provide great benefits to this domain by raising the level of abstraction and reducing complex and recurring programming tasks.

Model-driven software engineering [1], [2] puts strong emphasis on the development of high-level models rather than on the source code. Models are neither considered as documentation nor as informal guidelines how to program the actual system. In contrast, models have a well-defined syntax and semantics. Moreover, model-driven software engineering aims at the development of *executable* models. *Code generators* are used in model-driven software engineering, to transform the specification of higher-level models into source code. A *domain-specific language (DSL)* is a programming or specification language which is dedicated to a particular problem domain.

Software product line engineering (SPLE) [3], [4], [5] 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 artifacts are created such that each product may be composed

[1]T. Buchmann and B. Westfechtel are with Chair of Applied Computer Science I (Software Engineering), University of Bayreuth, 95447 Bayreuth, Germany firstname.lastname@uni-bayreuth.de

[2]J.Baumgartl and D. Henrich are with Chair of Applied Computer Science III (Robotics and Embedded Systems), University of Bayreuth, 95447 Bayreuth, Germany firstname.lastname@uni-bayreuth.de

from a library of components. Furthermore, it provides means to capture and manage the variability of a particular application domain. In common approaches, *feature models* [6] are used for that purpose.

In this paper, we present the work in progress of our domain-specific language for pick-and-place applications and especially the configurable code generator which produces C++ code.

II. A DOMAIN SPECIFIC LANGUAGE FOR PICK-AND-PLACE APPLICATIONS

As stated in Section I, programming a robot is a very complex task. Resulting programs highly depend on the robot's hardware and the environment in which the robot is being operated. Thus, our approach - whose basic ideas are presented in [7] - aims at enabling programmers without dedicated knowledge in the robotics domain to specify robot applications.

```
1   program pack_box
2
3   color RED (255,0,0)
4⊖  object redbox {
5⊕      objectConfig boxcfg {⬚}
9⊕      geometry {⬚}
12      color:=RED
13  }
14
15  /*depth camera */
16⊕ sensor cam {⬚}
22
23⊕ robot rb1 {⬚}
31
32  mylist : RSet<ObjectDeclaration> := cam::perceive
33
34⊖ foreach (ding : ObjectDeclaration in mylist) {
35      ding.pick(rb1)
36      ding.place(redbox)
37  }
```

Fig. 1. A small example of our DSL code.

The core part of our approach is a declarative domain-specific language for pick and place applications. We chose to start with this domain, since it covers basic robotic tasks like moving robots, grasping objects and placing them at a different location. These tasks, which sound easy at first glance, include inherently complex subtasks like object modeling, path planning, grasp planning, and placement planning. To empower users without dedicated background in those tasks, we abstracted from those concepts. Instead, modeling and planning operations are implemented in a C++ framework, which is used by the code generator presented

in Section III. As a consequence, the DSL code can be kept simple and human readable as shown in Figure 1.

A. Design Decisions

The most difficult task when designing a domain-specific language is to find the right level of abstraction as well as the required keywords. A basic question is whether object and hardware declarations are required in the DSL or not.

The sample DSL code in Figure 1 contains various declarations of different types. In its current status the DSL allows declarations, for e.g. colors, objects, sensors, and robots as well as object and robot configurations (c.f. lines 2 - 41 in the sample).

While declarations of sensors and robots are useful when the generated code is meant to be run using several robots, declarations might be obsolete when using an educated distribution planner to assign a subtask to a specific sensor or robot. However, in this paper we focus on one robot with one sensor network capable to model objects to manipulate with, where those hardware declarations are just convenient.

Dependent on used hardware, requirements for the algorithms might vary. Those dependencies should be implemented as constraints on the feature model of the product line and not be part of the DSL, since the concrete algorithms are transparent to the user, likewise the interaction with the concrete hardware.

The second design decision is concerned about the keywords that should be provided by the DSL. The current version of the DSL comprises keywords for object, sensor, and robot declarations and configurations. Furthermore keywords for built-in data types and control structures are included. The keyword `object` in the declarative part can be used to define static environment or known objects. Following an object-centered approach, objects are linked with hardware by keywords for object manipulation (`pick` and `place`), robot movement (`move`), and operations on sensors (e.g. `perceive`). A concrete (planning) algorithm must be available for each of the keywords. However, different realizations concerning one keyword might exist. Those are selected depending on the hardware.

B. Implementation

We decided to use the Xtext[1] framework for our textual DSL. Xtext allows the specification of a context-free grammar of a language. It uses ANTLR[2] as a parser generator, which means that is able to parse LL(*) grammars. Furthermore, the Xtext framework allows to enrich the Xtext grammar specification with context-sensitive information, which is used to unparse a text into an Ecore-based tree representation. The resulting, automatically generated editor comprises full-fledged support for syntax highlighting and code completion.

III. Configurable Code Generator

The DSL code needs to be compiled into executable code in order to be run on real robots or within a simulator. To this end, we use the Accelo[3] framework which provides an implementation of the *OMG MOF Model to Text standard* [8]. Acceleo can be used to specify code generators for arbitrary Ecore-based metamodels.

The target platform of the code generator is GeNBot - a C++ framework which comprises different algorithms for path planning, grasp planning and placement planning as well as a modular interface to different robots (Kuka LWR, Kuka KR16-2, Stäubli RX130) and simulators.

Acceleo provides a template-based code generation engine equipped with its own template language MTL. OCL[4] is used to retrieve model information dynamically, which is used in the templates to generate code.

Figure 2 shows a small cutout of our code generation templates which is used to initialize a LWR robot controller. The code formatted in blue color between square brackets depicts dynamic code fragments which are extracted from the model (e.g. the DSL code) at runtime. Text formatted in black color is static text which is used as it stands each time the template is invoked.

The code which is produced by the template above is shown in Listing 1. The code contains fragments which are neccessary to initialize a Kuka LWR robot controller in the GeNBot framework. This code is necessary in every application which is intended to be run on this type of hardware. But it also contains some variable parts like the number of joints for example so that it could not be reused by plain copy and paste in "traditional" programming approaches.

As stated in [7], our approach aims at integrating a product line approach to cover the variability which may occur in the target domain due to changing hardware (robots, sensors) and software (algorithms used for planning tasks etc.). Thus, we started to integrate our FAMILE environment, which is dedicated to the model-driven development of software product lines [9], [10].

```
181 [template private robotPhysicalInitializer(aRobot : RobotDeclaration)]
182 /* Instantiate the robot controller */
183 [RobotPrefix()/]::BaseType::Ptr [aRobot.name/] =
184     GeNBot::Factory::buildRobotController<
185         [RobotPrefix()/],
186         [HaltInterpolator()/]<[RobotNumberOfJoints()/]>,
187         [JointInterpolator()/]<[RobotNumberOfJoints()/]>,
188         [NSAInterpolator()/]<[RobotNumberOfJoints()/]>
189     (
190         [RobotPrefix()/]::BaseType::RobotIKPtr
191             (new [InverseKinematics(KinematicsFile())/]),
192         [RobotPrefix()/]::BaseType::RobotFKPtr
193             (new [ForwardKinematics(KinematiksFile())/])),
194         [IpoCycle()/]
195     );
196
197 [RobotPrefix()/]::BaseType::RobotIKPtr
198     [aRobot.name/]_inverse_kinematics_ptr
199         (new [InverseKinematics(KinematiksFile())/]);
200 [/template]
```

Fig. 2.   A cutout of the code generator templates.

Listing 1. Cutout of the generated C++ code

```
1   /* Instantiate the robot controller */
2   GeNBot::LWRRobotController::BaseType::Ptr rb1 =
        GeNBot::Factory::buildRobotController<
3       GeNBot::LWRRobotController,
4       GeNBot::HaltInterpolator<7>,
5       GeNBot::ReflexxesJointInterpolator<7>,
6       GeNBot::ReflexxesNSAInterpolator6D<7>
7       (
8         GeNBot::LWRRobotController::BaseType::
            RobotIKPtr
9           (new GeNBot::LWR_ik_AC()),
10          GeNBot::LWRRobotController::BaseType::
            RobotFKPtr
11            (new GeNBot::LWRFK(std::string("/path/
              to/kinematicsFile.xml"))),
12          0.034f
13      );
14
15  GeNBot::LWRRobotController::BaseType::RobotIKPtr
16    inverse_kinematics_ptr
17      (new GeNBot::LWR_ik_AC());
```
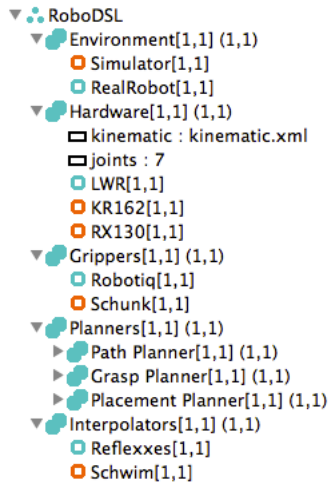

Fig. 3.   A sample feature configuration.

Currently, we are addressing the variability which concerns the code generator. Depending on the target platform (simulator, real robot) different building blocks of the C++ framework are used in the generated C++ code. Furthermore, three types of robots (Kuka LWR, Kuka KR16-2, Stäubli RX130) and different planning algorithms are supported. Our FAMILE toolchain uses feature models [6] to capture commonalities and variabilities of the product line. Figure 3 shows a sample feature configuration of the product line, which is used as an input during the code generation process to bind the variability. Elements marked with cyan colored circles depict features which are included in the current feature configuration, while orange colored circles mark excluded ones. Features also may have attribute values, e.g. feature `Hardware` contains the attribute `joints`. In its current state, the configuration (e.g. selecting / deselecting) the appropriate features in the feature model has to be done manually.

IV. FUTURE WORK

In its current state, our approach already covers variability on the level of the code generator. E.g. different code is being generated depending on the feature configuration which is passed to it. But variability in robotics hardware does not only affect the generated code (by initializing and using appropriate building blocks of the GeNBot framework), it may also concern the language itself. The presence / absence of hardware or software might result in the inclusion or exclusion of language constructs. In this case, the end user cannot specify DSL programs which cannot be run on the target hardware. As a consequence, variability on the level of the grammar of the DSL is required. Our toolchain FAMILE was built to support model-driven software product line engineering for arbitrary Ecore-based domain models. As an Xtext grammar is parsed into an Ecore-based syntax tree (as the Xtext editor is specified with Xtext itself) FAMILE can be used to map features from the feature model to grammar production rules. As a result, language elements decorated with feature expressions evaluating to false (in case the respective features are excluded from the current feature configuration) are omitted.

Furthermore, Acceleo also provides an Ecore-based model for its abstract and concrete syntax. While the connection between feature model and code generation templates is realized via Acceleo queries at the moment, we are currently working on using the feature mapping capabilities of FAMILE for it as well. Unfortunately it does not work out of the box, as Acceleo (contrastingly to Xtext) does not use a parser which automatically creates instances of this Ecore model in memory.

In order to support automatic configuration depending on the used hardware, a dedicated interface to the hardware as well as a protocol providing the required information (which will be used for an automatic configuration of the feature model) is necessary. This will also be addressed in future work.

Finally, we plan to extend the language to address other robotic application domains apart from pick and place as well.

V. RELATED WORK

In [7], we present the basic ideas behind our approach, which is intended to provide textual DSLs for robotic applications, which can be adapted at runtime according to the actual robot configuration. While [7] offers a conceptual overview, we present the first version of a DSL for pick-and-place applications and a configurable code generator which creates C++ code in this paper.

In [11], the authors present an approach which uses a DSL to handle run-time variability in programs for service robots. The approach presented by Inglés-Romero et al. aims to support developers of robotic systems (e.g. experts in the robotics domain) while our approach is not restricted to robotic experts only. Even regular programmers without a dedicated background in robotics are able to write robotic programs with our DSL. Furthermore, the DSL is only able

to express variability information. It is not possible to specify the behavior of the robot.

Steck et al. present an approach [12] that is dedicated to a model-driven development process of robotic systems. They present an environment called *SmartSoft* [13] which provides a component based approach to develop robotics software. The SmartSoft environment is based on Eclipse and the Eclipse Modeling Project[5]. It uses Papyrus[6] for UML modeling. By using a model-driven approach, the authors focus on a strict separation of roles throughout the whole development life-cycle. Again, experts in the robotics domain are addressed with this approach while our approach doesn't require expert knowledge in robotics.

RobotML [14], a modeling language for robot programs also aims to provide model-driven engineering capabilities for the domain of robot programming. RobotML is an extension to the Eclipse-based UML modeling tool Papyrus. Papyrus puts strong emphasis on UML's profile mechanism, which allows domain-specific adaptations. RobotML provides code generators for different target platforms, like Orocos, RTMaps, Arrocam or Blender/Morse. The approach presented by Dhouib et al. addresses developers of robot programs or algorithms, while our approach can also be used by regular programmers (of course robotic experts can use it as well and may gain an increase in productivity).

Bubeck et al. present in [15] an overview about best practices for system integration and distributed software development in service robotics. Furthermore, the authors develop *BRIDE*[7], a graphical DSL for ROS developers. Using BRIDE, new ROS nodes or ROS-based systems can be specified in a graphical way and corresponding C++ or Python code may be generated. In addition, the required launch files for the ROS environment including the relevant parameters and dependencies are generated as well, similar to the approach which we used in our case study as described in [7]. Similar to the approaches discussed above, BRIDE also addresses robot experts only.

In [16], Schultz et al. present an approach for a domain-specific language intended for programming self-configurable robots. The DSL is targeted towards the ATRON self-reconfigurable robot. Like all other approaches mentioned in this section, it aims to provide a higher-level of abstraction for robot experts.

In his PhD thesis [17] 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. Finally, the DSL described in this paper addresses programmers without a dedicated background in robotics, while [17] requires a robotics expert to provide algorithms and the variability model and additionally a software engineering expert.

---

[5]http://www.eclipse.org/modeling/

[6]http://www.eclipse.org/papyrus

[7]http://ros.org/wiki/bride

VI. CONCLUSION

In this paper we presented our approach towards easy robot programming for personal robots. We demonstrated the feasibility of our approach by presenting a small and declarative domain-specific language for pick and place applications. Furthermore, a product line approach was used to realize a configurable code generator for C++.

REFERENCES

[1] D. S. Frankel, *Model Driven Architecture: Applying MDA to Enterprise Computing*. Indianapolis, IN: Wiley Publishing, 2003.

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

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

[4] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Germany: Springer Verlag, 2005.

[5] D. M. Weiss and C. T. R. Lai, *Software Product Line Engineering: A Family-Based Software Development Process*, Boston, MA, 1999.

[6] 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.

[7] 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-PT 2013)*, J. Cordeiro and M. van Sinderen, Eds., Reykjavik, Iceland, Jul. 2013, pp. 147–157.

[8] OMG, *MOF Model to Text Transformation Language, Version 1.0*, formal/2008-01 ed., OMG, Needham, MA, Jan. 2008.

[9] T. Buchmann and F. Schwägerl, "FAMILE: tool support for evolving model-driven product lines," in *Joint Proceedings of co-located Events at the 8th European Conference on Modelling Foundations and Applications*, ser. CEUR WS, H. Störrle, G. Botterweck, M. Bourdells, D. Kolovos, R. Paige, E. Roubtsova, J. Rubin, and J.-P. Tolvanen, Eds. Building 321, DK-2800 Kongens Lyngby: Technical University of Denmark (DTU), Jul. 2012, pp. 59–62.

[10] T. Buchmann and F. Schwägerl, "Ensuring well-formedness of configured domain models in model-driven product lines based on negative variability," in *Proceedings of the 4th International Workshop on Feature-Oriented Software Development*, ser. FOSD '12. New York, NY, USA: ACM, 2012, pp. 37–44. [Online]. Available: http://doi.acm.org/10.1145/2377816.2377822

[11] 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 (DSLRob-12, co-located with SIMPAR 2012)*, E. Menegatti, Ed., Tsukuba, Japan, 2012.

[12] A. Steck, D. Stampfer, and C. Schlegel, "Modellgetriebene Softwareentwicklung für Robotiksysteme," in *AMS*, ser. Informatik Aktuell, R. Dillmann, J. Beyerer, C. Stiller, J. M. Zöllner, and T. Gindele, Eds. Springer, 2009, pp. 241–248.

[13] A. Steck and C. Schlegel, "Towards Quality of Service and Resource Aware Robotic Systems through Model-Driven Software Development," in *Proceedings of the First International Workshop on Domain-Specific Languages and models for ROBotic systems (IROS - DSLRob)*, Taipei, Taiwan, 2010.

[14] 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. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34327-8_16

[15] 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.

[16] 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.

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