Modeling Robot and World Interfaces for Reusable Tasks

Robert Heim[1], Pedram Mir Seyed Nazari[1], Jan Oliver Ringert[2], Bernhard Rumpe[1], and Andreas Wortmann[1]
[1]Software Engineering, RWTH Aachen University
[2]School of Computer Science, Tel Aviv University, Israel

Abstract— **Robotics applications involve robots that perform tasks by interacting with specific worlds. Most applications are intertwined with and tied to fixed robots and worlds. Changes and evolution of a robot or world have an invasive and often unpredictable impact on the application software.**

We propose making the models of robots and worlds explicit in robotics applications and separate these by introducing application-specific and platform-independent interfaces. This separation allows modular model-driven development of robotics applications and enables the reuse and adaptation of models and applications without need for invasive modifications.

We present a framework with a family of modeling languages for conceptual, platform-independent applications, tasks, robots, and worlds. The model-driven RoboTask framework integrates these languages with a runtime architecture to execute robotics tasks using a planner and mappings from conceptual models to actual platforms. This enables a separation of domain concerns from software development concerns and modification of applications without invasive impacts on their separated constituents. We believe that the enabled reuse and adaptation lead to more efficient development and higher quality software for robotics applications.

I. Introduction

A robotics application is a system of robots performing tasks within defined worlds. Even a simple application requires the interaction of various software modules controlling and representing application logic, robots, and world. Modifications of such applications lead to invasive changes due to the tight coupling of tasks, robots, and world. This intertwining impedes reuse and adaptation, and ultimately increases cost and risk of developing robotics applications.

Current directions to increase the reuse of robotics applications are component-based software engineering (CBSE), model-driven engineering (MDE), and combinations thereof. CBSE aims to increase reuse of software components, which provide stable interfaces to allow their exchange between applications. Such components are exchanged as programming language artifacts. Hence, they are platform-specific, many requirements and dependencies are implicit, and reuse is still hampered. MDE aims to increase abstraction and reuse by making models the primary development artifacts. Such models can be concise, platform-independent, and describe concepts in terms of the solution domain. Code generators transform models to platform-specific implementations. Current approaches to MDE in robotics however do not enforce a distinction between the domain-specific concepts of an application, which consists of reusable tasks, robots, and worlds, and their concrete implementations. Therefore, the resulting applications are usually tightly coupled to functionality and platforms and ultimately hardly reusable [1], [2].

We present the RoboTask framework and its modeling languages for development and execution of reusable robot tasks, robots, and worlds. Application models aggregate a domain model (domain data types) the application operates on, a robot model, and world model. The latter describe properties and actions of robot and world on a conceptual level. Tasks are expressed as sequences of goals, which refer to domain, robot, and world models. Hence, they are restricted in their expressiveness and independent of the participating platforms[1] and can be composed from goals by non-technical users. The goal language allows for more expressiveness and models are defined by technical staff. Models of these languages represent a conceptual understanding of the application available for planning, without a binding to concrete platforms.

From application and task models, code for a predefined MontiArcAutomaton [3], [4] software architecture is generated to map conceptual robot and world models to platform code. At run-time, this architecture executes tasks and translates their goals into PDDL [5] problems for an embedded planner, which determines the sequences of actions required to fulfill the passed goal. The sequences refer to the conceptual robot and world model actions and mappings ground these to the platforms. RoboTask

- enables reuse of robots and worlds with different tasks and goals;
- separates concerns of users, who compose goals to tasks, from concerns of domain experts, who describe tasks decoupled from the executing systems, from concerns of robot and world experts, who map robot and world actions and properties to platforms and take care of error conditions and safety features;
- liberates domain experts from describing how goals are fulfilled: instead actions and properties are defined, and actual task performance is delegated to a planner;
- allows partition of complex tasks into intermediate goals, which reduces planning effort and increases behavior predictability.

This research has partly received funding from the German Federal Ministry for Education and Research under grant no. 01IM12008C. The responsibility for the content of this publication is with the authors.

[1]In this context, “platform” may refer to robots or environment and denotes the hardware and software necessary to provide the service of a certain application.

Section II illustrates the RoboTask modeling languages and their usage. Afterwards, Sect. III describes the framework and its code generation. Section IV highlights related work, before Sect. V describes case studies and discusses observations. Finally, Sect. VI concludes.

II. Modeling Languages and Concepts

RoboTask employs MDE to separate problem domain solutions from their concrete platform-dependent executions. We describe the domain-specific languages to model platform-independent concepts, such as domain types, and robot and world interfaces. Furthermore, we introduce two languages that allow modeling solutions as tasks and goals based on the platform-independent models.

A. RoboTask Language Family

The language family of RoboTask consists of six modeling languages: Application, Robot, World, and UML/P class diagrams (CDs) [6] describe platform-independent domain concepts, Task and Goal describe solutions.

Class diagrams describe domain types, which define common concepts of the application domain, e.g., properties of rooms and items. Models of other languages use these types. The Robot language allows definition of conceptual robot types by specifying their platform-independent interfaces of actions and properties. Actions describe capabilities of the robot in terms of preconditions required for executing the action and postconditions that are supposed to hold after its execution. Both, pre- and postconditions may refer to properties (of robot and world) and domain data types. Properties describe characteristics of the robot's state. World models describe actions and properties of the world in the same way. The Application language combines a robot and a world that the robot operates in and thereby provides a platform independent set-up to define solutions.

Models of the Goal language are bound to a specific domain and consist of a name and an optional list of typed parameters. They describe desired states with Boolean expressions over properties of robots and worlds, local variables, and simple control structures (such as conditionals and loops) similar to the logic programming language GOLOG [7]. Additionally, goals may reference other goals to combine multiple goals to an overall goal. The Task language is developed to be used by non-technical users and thus only allows to define solutions as sequences of references to parametrized Goal models. A task is considered finished if each of its goals was fulfilled subsequently. Tasks and goals are platform-independent, because they solely rely on robot and world models. Hence, they can directly be reused on different platforms, making solutions platform-independent.

B. Example

Consider a company producing transport service robots: due to different requirements, the company fabricates multiple types of robots. Nonetheless, the concepts of provided indoor transport services are similar: robots can move between rooms, which may be open or closed, and can pick up and deliver items. To increase reuse between different contexts, a software engineer at the aforementioned company defines the indoor transport service (ITS) application, which models those common concepts independent of the specific robot and world used in different contexts. Figure 1 illustrates the concepts of applications on the example of the `ITSApplication`, which combines the robot `ITSRobot` and world `ITSWorld` and is used to integrate them during generation of the software architecture. Robot and world use the domain `ITSDomain` which provides common data types (e.g., `Room`). The model `ITSRobot` requires any robot usable with the `ITSApplication` to provide at least the actions `goTo`, `pickUp`, and `unload`, as well as the properties `location` and `isLoaded`. Similarly, the `ITSWorld` defines that any world usable with this application provides the action `openDoor`. Furthermore, robot and world define pre- and postconditions of actions. For instance, the robot's action `goTo(Room r)` has the effect that afterwards the robot's location is in room `r`.

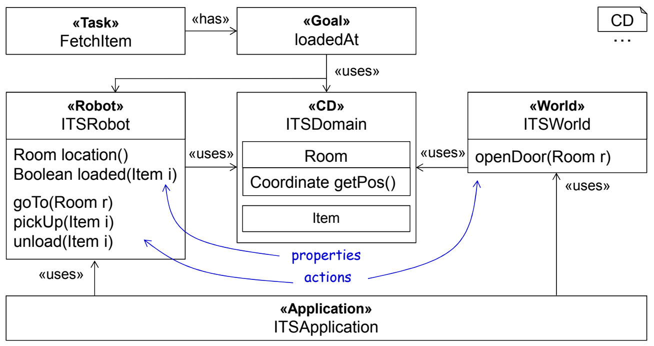

Fig. 1. Models of the RoboTask modeling languages and their relations.

```
                                          Task
1 domain ITSDomain;
2
3 task FetchItem(Room src, Item i, Room dst){
4   loadedAt(src,i);
5   unloadedAt(dst,i);
6 }
```

Fig. 2. The task `FetchItem` must be instantiated with three parameters and uses these to instantiate two goals. It is considered fulfilled if all goals are fulfilled in the specified order.

Based on this, consider a task `FetchItem`, which instructs the robot to enter a room, pick up the requested item, and return to a destination. This task consists of the goal sequence `loadedAt`, `unloadedAt`. The task model in Fig. 2 references the required domain (l. 1) and defines the task name with its parameters (l. 3). Afterwards, a sequence of the two goals (ll. 4-5) follows. These define the conditions for parts of this task to be fulfilled.

The goal `loadedAt` is illustrated in Fig. 3 and requires an `ITSRobot` robot named `rob` (l. 2) as it checks the properties `location` and `isLoaded` of the robot. The model defines its domain (l. 1), the goal name and parameters (l. 4), and Boolean expressions over properties of the robot

```
                                              Goal
1 domain ITSDomain;
2 robot  ITSRobot as rob;
3
4 goal loadedAt(Room r, Item i) {
5   rob.getLocation() == r;
6   rob.isLoaded(i);
7 }
```

Fig. 3. The goal `loadedAt` defines a Boolean condition over properties of the `ITSRobot` robot named `rob`.

(ll. 5-6). Section III describes how concrete applications implement robot and world models by binding their actions and properties to concrete platforms.

C. Implementation of RoboTask Languages

We implemented the language family of RoboTask using the MontiCore [8] language workbench. MontiCore generates components for parsing models and provides infrastructure for checking well-formedness rules via *context conditions* [9], language integration [10], and code generation [11]. RoboTask uses language aggregation, embedding, and language inheritance features of MontiCore to compose the individual languages into an integrated language family [12].

Using UML/P CDs and the JAVA/P [11] modeling language (as expression language for pre- and postconditions of ROBOT and WORLD models, as well as for statements of GOAL models) allows to reuse existing context conditions checking and code generation infrastructures. Since a PDDL planner relies on the pre- and postconditions, the expressions are restricted to be transformable into PDDL. To this effect, all languages are subject to various additional intra-language context condition checks regarding validity of their models, for example, that actions of a particular robot are named uniquely. Aggregation into a language family allows to check inter-language properties, e.g., whether robot properties referenced within goals are combined correctly into expressions, which evaluate to Boolean values. The framework checks completeness and type compatibility properties for all models.

MontiCore enables the a-posteriori integration of modeling languages via *symbol tables* (ST) [12]. STs store *entries* that encapsulate the "essence" of model elements and provide these to the language integration mechanisms (e.g., ROBOT models provide information about the contained actions to other languages but hide their pre- and postconditions). Thus, the MontiCore language integration framework allows to exchange, e.g., the JAVA/P statements embedded into GOAL models with statements of another modeling language without invasive modifications. Furthermore, with STs, both intra-language and inter-language constraints can be defined easily using the context condition framework of MontiCore: when MontiCore parses a model, it creates the ST and performs manually implemented intra- and inter-language condition checks. All languages of RoboTask are strongly typed to allow type compatibility checks between models of different languages.

With these languages, the RoboTask framework generates parts of the run-time system required to operationalize robot and world models. The next section explains how RoboTask models are transformed into platform-specific program code.

III. USING THE ROBOTASK FRAMEWORK

Although task, robot, and world models conceptually allow for planning, they have to be transformed into appropriate PDDL problems first. Furthermore, the resulting plans, sequences of actions, need to be translated to platform calls to perform tasks in the real world. The RoboTask framework takes care of both. We first introduce the RoboTask development methodology before we describe the framework.

A. RoboTask Development Methodology

With RoboTask, we aim for a clear separation of concerns between (i) users, who know what the robot is supposed to do and define tasks accordingly, (ii) domain experts, who understand software development, know the domain concepts, and can express these concepts using CDs, actions, and properties, and (iii) robotics experts, who know how to map action and properties onto the platforms.

To this end, the *domain expert* (cf. Fig. 4) begins development with definition of the domain model. This model captures the complete domain from a conceptual perspective and hence describes the types available and how these can be used. The `ITSDomain`, for instance, describes rooms with coordinates. Based on this, the *domain expert* models both robot and world with their respective actions and properties. Domain model, robot, and world may be defined iteratively and until these are considered conceptually complete. Afterwards, the *domain expert* also describes the goals that can be fulfilled with this application. As goals refer to robot and world properties, development of goals requires understanding of CDs, Boolean expressions, and control structures. Hence, we do not consider their definition within concerns of the *user*. The *user* only may refer to goals to compose tasks. Following a descriptive approach liberates domain experts from defining how robot and world fulfill goals and leaves this effort for the planner.

Parallel to development of goals, *robot expert* and *world expert* develop mappings from robot model and world model to the specific platforms by implementing the interfaces generated for these. This implementation also acts a glue code between domain concepts (e.g., `Room`) and platforms. As MontiArcAutomaton allows development, integration, and composition [13] of code generators with minimal effort, RoboTask generally enables *robot expert* and *world expert* to use the programming language of their choice.

B. The RoboTask Framework

The RoboTask framework (cf. Fig. 4) parses models of tasks and goals that refer to robots, worlds, and domain models, and models of applications, which refer to the available robot and world. Together with a MontiArcAutomaton software architecture model [3], [4], RoboTask transforms these into an executable system that employs Metric-FF [14]

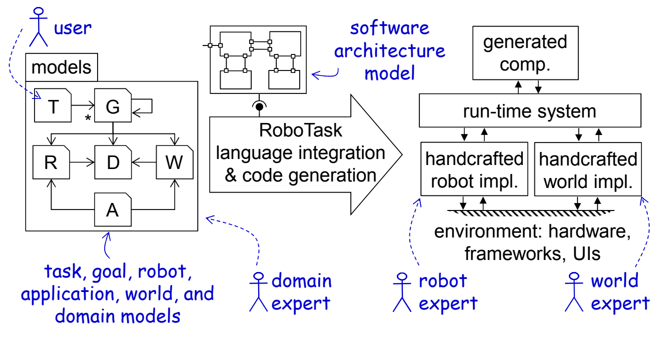
user
models
T
G
R
D
W
A
task, goal, robot, application, world, and domain models
domain expert
software architecture model
RoboTask language integration & code generation
generated comp.
run-time system
handcrafted robot impl.
handcrafted world impl.
environment: hardware, frameworks, UIs
robot expert
world expert

Fig. 4. The RoboTask framework parses models of tasks, goals, robots, world, applications, and a software architecture to transform these into an executable high-level robot task execution framework interfacing the Metric-FF planner and platforms.

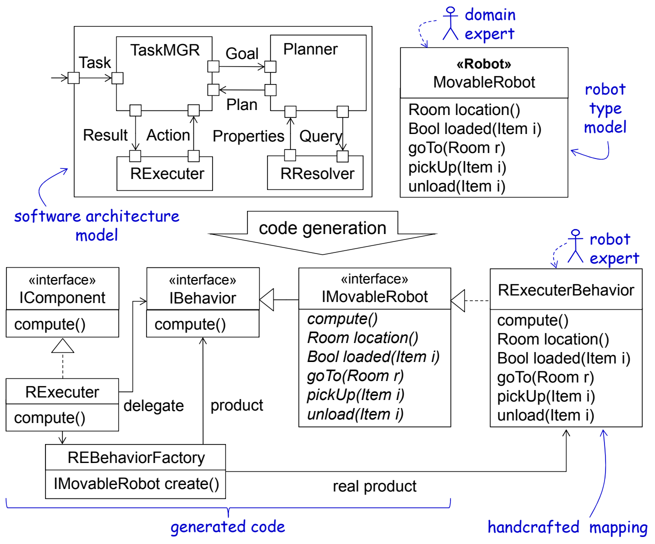

Fig. 5. Code generation mechanisms employed by RoboTask to ground actions and properties in the participating platforms.

to solve tasks. To this effect, the *robot expert* and *world expert* are responsible for adding handcrafted implementations of the conceptual robot and world models that interface the respective platform and exchanges messages via the MontiArcAutomaton run-time system.

In the MontiArcAutomaton Component & Connector (C&C) [15] software architecture, components encapsulate functionality as black boxes with stable interfaces of directed typed ports. Unidirectional connectors connect ports and are the only means for component interaction. Components are either composed, and their behavior emerges from the composition of the behaviors of their sub-components, or they are atomic, and their behavior is defined with a behavior modeling language or a target platform-specific implementation. The software architecture depicted in Fig. 5 (simplified) contains a task manager component `TaskMGR` to process coordination, planning, and execution of incoming tasks models, a `Planner` component to interface Metric-FF for actual planning, a component `RExecuter` to execute actions on the robot, and a component `RResolver` to retrieve values of robot properties. Please note that the description omits similar components for execution of actions of the world, retrieving the world's properties, and extension points.

From this architecture model MontiArcAutomaton generates platform-specific code (bottom part of Fig. 5) that employs the delegator pattern and the factory pattern to facilitate exchange of component behavior descriptions: concrete components, such as `RExecuter`, implement the interface `IComponent`, which defines that components are invoked using the method `compute`. Component `RExecuter` delegates calls to this method to a member that implements the interface `IBehavior` and is produced by the factory `REBehaviorFactory`. This factory produces instances of the actual component behavior implementation `RExecuterBehavior` (the "handcrafted robot implementation" depicted in Fig. 4) that implement `IBehavior`. This enables to treat both generated and handcrafted component behavior implementations uniformly and to exchange these at run-time. RoboTask extends this with an interface generated from the robot model and different generation of the behavior factory. In this example, an interface `IMoveableRobot` is generated that extends the signature of `IBehavior` with methods representing the actions and properties of the robot model. Furthermore, RoboTask updates the factory `REBehaviorFactory` to produce instances of `IMoveableRobot` instead of `IBehavior`. This enables the *robot expert*, responsible for providing an implementation of the robot model `MovableRobot`, to develop component behavior implementing `IMoveableRobot` with proper methods. The same applies to world models.

This framework enables modeling of robot tasks that rely on conceptual robots and worlds instead of actual platforms and thus facilitates a separation of concerns. Interfacing Metric-FF and binding actions and properties to powerful robot platforms enables reuse of domain tasks and goals with different robots and worlds with minimal effort.

IV. Related Work

RoboTask employs MDE to enable platform-independent description of domain and robot expertise. As such, it is related to other applications of MDE to robotics.

MDE has been successfully applied to different aspects of robotics applications [16]. Current robotics MDE research aims to reduce the complexity of imperative or event-driven programming [17], [18], [19], [20], to describe kinematics, or geometric relations [21], [22], or to model software architectures for robotic systems [2], [3], [23], [24], [25], [26]. These approaches have in common that they target language and tool users from the solution domains, i.e., software developers. Languages and tools for problem domain users tend to be technical [27], [28], [29] and, for instance, impose understanding of automata to these users.

Currently, there is few related work on modeling platform-independent, reusable, and plannable robotics applications. We assume this is due to the inherent complexity of robotics applications: even simple applications require the participation of experts from multiple fields, such as computer vision or path planning. Thus, applications are developed with small regard for reusability [2], [30].

Plannable robotics applications usually consist of an (untyped) knowledge base containing symbols and rules for both robot and world, and a software system interacting with this knowledge base (cf. [2], [31]). This requires that domain experts are knowledge representation experts as well and prohibits the separation of concerns required between domain experts modeling the application type and software engineers implementing these as applications for concrete robots and worlds (cf. [32], [33])

Similar approaches to platform-independent robot programming are [34] and [35]. In [34], tasks are sequences of actions, which require "resource components" [34]. Tasks reference abstract concepts and concrete robots implement these. For each robot, domain experts have to develop a new XML-based DSL with proper code generators. Thus, this approach requires the domain expert to have "profound knowledge in programming a certain robot respective robot class" [34], which we aim to avoid. With RoboTask, the domain expert only needs to comprehend the domain model and the robot and world models. The framework presented in [35] describes "processes" imperatively as graphs of actions, which are supposed to be defined by non-technical users. Its run-time system translates actions to robot capabilities and takes care of execution.

V. Case Study and Discussion

We have evaluated the RoboTask modeling languages and framework with two case studies representing logistics applications. Students performed these case studies and their reports lead to various improvements on languages and framework.

A. Logistics Case Studies

The logistics applications comprised of between 1 and 3 tasks, with between 1 and 4 goals and used between 2 and 8 actions and queries to fulfill tasks. We performed the logistics case studies on two Festo Robotino[2] robots operated with SmartSoft [2] and ROS [36], respectively.

The developers of the first case study reported that modeling simple logistics tasks was straightforward, but describing more complex tasks required effort due to the lack of Goal expressiveness. The integration of conditionals and loop statements into Goal is due to this feedback. Furthermore, the first version of RoboTask included additional languages that required definition of further abstraction layers. The case study revealed that these layers introduced unnecessary complexity and thus were removed. Thereby, RoboTask gained a better separation of concerns. Developers also reported that handcrafting the translation from action and properties to platform code enabled to realize capabilities not directly provided by the platform: for instance, they mapped the action `pickUp` (cf. Fig. 1) to user interaction on the Robotinos.

[2]Festo Robotino website: `http://www.festo-didactic.com/int-en/news/the-new-robotino.htm`

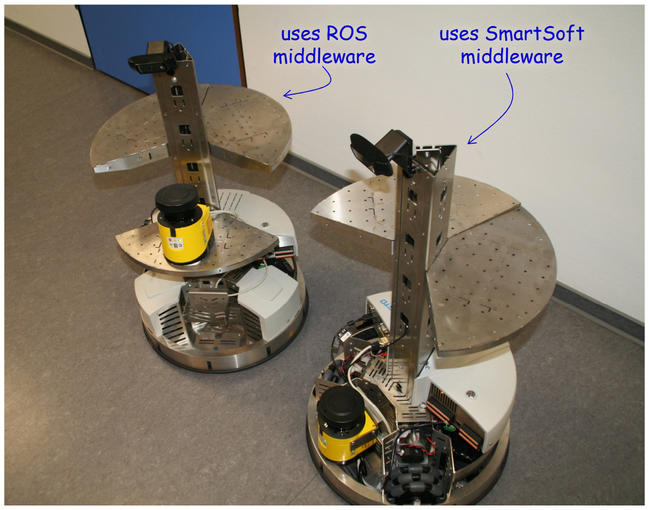

Fig. 6. Robotino robots used in the logistics case studies with with SmartSoft [2] and ROS [36].

B. Discussion

Current tasks can be perceived as a reduced view on goals, as goals may express the requirement, i.e., that a sequence of goals needs to be fulfilled, as well. We will investigate whether this suffices and whether the Task language should be extended with more advanced features in the future.

A major benefit of the RoboTask modeling languages to be developed as MontiCore languages is that development of language variants using language inheritance or language embedding is straightforward [10], [12], which might lead to a family of RoboTask languages for users of different technical expertise. We currently also investigate whether it is necessary and feasible to bridge the discrepancy between the typed RoboTask modeling languages and the untyped Metric-FF PDDL input and how to deal with incomplete knowledge.

VI. Conclusion

We presented the RoboTask framework to decouple the modeling of robotic tasks from specific robots and worlds. RoboTask therefore introduces applications that use common domain types, robots, and worlds. Domain experts can provide these models with little software engineering knowledge. Robot and world experts realize robot and world models via mappings to platform code. Using these models, a C&C software architecture, and the realizations of robot and world models, RoboTask generates an executable system. This system reads tasks, which comprise of goals provided by the domain expert, and executes these with help of the Metric-FF planer.

This separation between domain expert knowledge, robot and world expert knowledge, and software engineering knowledge embodied in the framework, allows modular development of robotics applications without invasive application changes caused by the inevitable needs for adaptation and evolution. Also decoupling conceptual capabilities and requirements from actual platforms increases reuse of domain concepts with different platforms. First case studies have pointed out benefits and issues of RoboTask, therefore further evaluation is ongoing in multiple projects.

References

[1] M. Hägele, N. Blümlein, and O. Kleine, "Wirtschaftlichkeitsanalysen neuartiger Servicerobotik- Anwendungen und ihre Bedeutung für die Robotik-Entwicklung," BMBF, Tech. Rep., 2011.

[2] C. Schlegel, A. Steck, and A. Lotz, "Model-Driven Software Development in Robotics : Communication Patterns as Key for a Robotics Component Model," in *Introduction to Modern Robotics*. iConcept Press, 2011.

[3] J. O. Ringert, B. Rumpe, and A. Wortmann, "From Software Architecture Structure and Behavior Modeling to Implementations of Cyber-Physical Systems," in *Software Engineering 2013 Workshopband*, ser. LNI, Stefan Wagner and Horst Lichter, Ed., vol. 215. GI, Köllen Druck+Verlag GmbH, Bonn, 2013, pp. 155–170.

[4] ——, *Architecture and Behavior Modeling of Cyber-Physical Systems with MontiArcAutomaton*, ser. Aachener Informatik-Berichte, Software Engineering. Shaker Verlag, 2014, no. 20.

[5] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL—The Planning Domain Definition Language," Yale Center for Computational Vision and Control, Tech. Rep., 1998.

[6] B. Rumpe, *Modellierung mit UML*. Springer, 2004.

[7] H. J. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. B. Scherl, "GOLOG: A Logic Programming Language for Dynamic Domains," *Journal of Logic Programming*, vol. 31, no. 1-3, pp. 59–83, 1997.

[8] H. Krahn, B. Rumpe, and S. Völkel, "Monticore: a framework for compositional development of domain specific languages," in *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 12, 2010, pp. 353 – 372.

[9] S. Völkel, *Kompositionale Entwicklung domänenspezifischer Sprachen*, ser. Aachener Informatik-Berichte, Software Engineering Band 9. 2011. Shaker Verlag, 2011.

[10] M. Look, A. Navarro Perez, J. O. Ringert, B. Rumpe, and A. Wortmann, "Black-box Integration of Heterogeneous Modeling Languages for Cyber-Physical Systems," in *Proceedings of the 1st Workshop on the Globalization of Modeling Languages (GEMOC)*, 2013.

[11] M. Schindler, *Eine Werkzeuginfrastruktur zur agilen Entwicklung mit der UML/P*, ser. Aachener Informatik-Berichte, Software Engineering, Band 11. Shaker Verlag, 2012.

[12] A. Haber, M. Look, P. Mir Seyed Nazari, A. Navarro Perez, B. Rumpe, S. Voelkel, and A. Wortmann, "Integration of Heterogeneous Modeling Languages via Extensible and Composable Language Components," in *Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development*, 2015.

[13] J. O. Ringert, B. Rumpe, and A. Wortmann, "Multi-Platform Generative Development of Component & Connector Systems using Model and Code Libraries," in *1st International Workshop on Model-Driven Engineering for Component-Based Systems (ModComp 2014)*, 2014.

[14] J. Hoffmann, "Extending FF to Numerical State Variables," in *Proceedings of the 15th European Conference on Artificial Intelligence*, July 2002.

[15] N. Medvidovic and R. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages," *IEEE Transactions on Software Engineering*, 2000.

[16] A. Ramaswamy, B. Monsuez, and A. Tapus, "Model-driven Software Development Approaches in Robotics Research," in *Proceedings of the 6th International Workshop on Modeling in Software Engineering*, 2014.

[17] H. Mühe, A. Angerer, A. Hoffmann, and W. Reif, "On reverse-engineering the KUKA Robot Language," in *First International Workshop on Domain-Specific Languages and Models for ROBotic Systems*, 2010.

[18] J.-C. Baillie, A. Demaille, Q. Hocquet, and M. Nottale, "Events! (Reactivity in urbiscript)," in *First International Workshop on Domain-Specific Languages and Models for ROBotic Systems*, Oct. 2010.

[19] A. Angerer, R. Smirra, A. Hoffmann, A. Schierl, M. Vistein, and W. Reif, "A Graphical Language for Real-Time Critical Robot Commands," in *Proceedings of the Third International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob 2012)*, 2012.

[20] J. Baumgartl, T. Buchmann, and D. Henrich, "Towards Easy Robot Programming: Using DSLs, Code Generators and Software Product Lines," *8th International Conference on Software Paradigm Trends (ICSOFT-PT'13)*, 2013.

[21] 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*, 2007, pp. 28–36.

[22] M. Frigerio, J. Buchli, and D. G. Caldwell, "A Domain Specific Language for kinematic models and fast implementations of robot dynamics algorithms," in *Proceedings of the Second International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob 2011)*, 2011.

[23] 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*, 2012.

[24] A. Nordmann and S. Wrede, "A Domain-Specific Language for Rich Motor Skill Architectures," in *Proceedings of the Third International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob 2012)*, 2012.

[25] H. Bruyninckx, M. Klotzbücher, N. Hochgeschwender, G. Kraetzschmar, L. Gherardi, and D. Brugali, "The BRICS component model: a model-based development paradigm for complex robotics software systems," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 2013.

[26] D. Stampfer and C. Schlegel, "Dynamic state charts: composition and coordination of complex robot behavior and reuse of action plots," *Intelligent Service Robotics*, vol. 7, no. 2, pp. 53–65, 2014.

[27] U. Thomas, G. Hirzinger, B. Rumpe, C. Schulze, and A. Wortmann, "A New Skill Based Robot Programming Language Using UML/P Statecharts," in *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, 2013.

[28] Vanthienen, Dominick and Klotzbuecher, Markus and De˜Laet, Tinne and De˜Schutter, Joris and Bruyninckx, Herman, "Rapid application development of constrained-based task modelling and execution using Domain Specific Languages," in *Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013.

[29] D. Vanthienen, M. Klotzbuecher, and H. Bruyninckx, "The 5c-based architectural composition pattern: lessons learned from re-developing the itasc framework for constraint-based robot programming," *JOSER: Journal of Software Engineering for Robotics*, 2014.

[30] D. Brugali, A. Brooks, A. Cowley, C. Côté, A. Domínguez-Brito, D. Létourneau, F. Michaud, and C. Schlegel, "Trends in Component-Based Robotics," in *Software Engineering for Experimental Robotics*, ser. Springer Tracts in Advanced Robotics, D. Brugali, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, vol. 30, ch. 8.

[31] S. Schiffer, A. Wortmann, and G. Lakemeyer, "Self-Maintenance for Autonomous Robots controlled by ReadyLog," in *7th IARP Workshop on Technical Challenges for Dependable Robots in Human Environments*, F. Ingrand and J. Guiochet, Eds., 2010, pp. 101–107.

[32] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, and S. Thrun, "Towards robotic assistants in nursing homes: Challenges and results," *Robotics and Autonomous Systems*, vol. 42, no. 3, pp. 271–281, 2003.

[33] J. Diprose, B. Plimmer, B. MacDonald, and J. Hosking, "How people naturally describe robot behaviour," in *Australasian Conference on Robotics and Automation (ACRA)*, Victoria University of Wellington, New Zealand, 2012, p. 9.

[34] M. Reckhaus, N. Hochgeschwender, P. G. Ploeger, G. K. Kraetzschmar, and S. Augustin, "A Platform-independent Programming Environment for Robot Control," in *Proceedings of the 1st International Workshop on Domain-Specific Languages and models for ROBotic systems (DSLrob'10)*, 2011.

[35] Andersen, Rasmus Hasle and Solund, Thomas and Hallam, John, "Definition and Initial Case-Based Evaluation of Hardware-Independent Robot Skills for Industrial Robotic Co-Workers," in *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of.* VDE, 2014, pp. 1–7.

[36] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.