2015 IEEE International Conference on Robotics and Automation (ICRA)
Washington State Convention Center
Seattle, Washington, May 26-30, 2015

# Modeling of Movement Control Architectures based on Motion Primitives using Domain-Specific Languages

Arne Nordmann Sebastian Wrede Jochen Steil

***Abstract*— This paper introduces a model-driven approach for engineering complex movement control architectures based on motion primitives, which in recent years have been a central development towards adaptive and flexible control of complex and compliant robots. We consider rich motor skills realized through the composition of motion primitives as our domain. In this domain we analyze the control architectures of representative example systems to identify common abstractions. It turns out that the introduced notion of motion primitives implemented as dynamical systems with machine learning capabilities, provide the computational building block for a large class of such control architectures. Building on the identified concepts, we introduce domain-specific languages that allow the compact specification of movement control architectures based on motion primitives and their coordination respectively. Using a proper tool chain, we show how to employ this model-driven approach in a case study for the real world example of automatic laundry grasping with the KUKA LWR-IV, where executable source-code is automatically generated from the domain-specific language specification.**

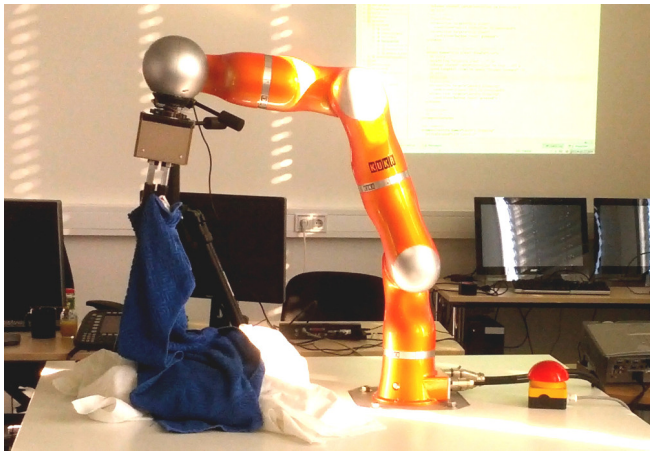


Fig. 1: Case study: automated gripping of laundry with the KUKA Lightweight Robot IV.

## I. Introduction

Model-driven and domain-specific development methods [1] are known to cope with the challenges of building complex heterogeneous systems in domains such as aerospace, telecommunication and automotive [2], which face similarly complex integration and modeling challenges as advanced robotics. In recent years, this approach has been adapted to the robotics domain [3], e.g. for specifying robot structures and control [4], coordinate representations [5] and task-level coordination [6]. The overarching goal of our work is to establish a model-driven development process focused on motion control architectures based on motion primitives. To the best of our knowledge, no such process or domain-specific language for motion control architectures in advanced robotics exists so far.

Insights from biology suggest that complex motions in animals and humans arise from the composition of so called *motion primitives*. Research over the last years successfully modeled motion primitives as the combination of autonomous dynamical systems and machine learning capabilities, e.g. Schaal et al. [7]. The domain of dynamical systems-based motion generation in robotics, however, is dominated by single, yet incompatible, experiments and solutions [8]. This calls for a unifying conceptual framework which allows the combination of different motion primitives to explore the design space of motion control architectures. This is challenging both because of the intrinsic complexity of the underlying control problems and due to the conceptual fragmentation of the domain.

As an important first step, we introduce a conceptual framework and a metamodel for motion control architectures based on motion primitives. The metamodel serves as a basis for two DSLs that model two orthogonal aspects which we identified in the domain: i) structural aspects of motion architectures focus on so called *Adaptive Modules* that model motion primitives as dynamical systems with machine learning capabilities and ii) behavioral aspects for the coordination of motion control systems. We show how the composition of these two DSLs provides support for specifying motion control architectures.

Figure 2 shows our levels of modeling that map to the standard of the Object Management Group [9] (OMG): M3, the **m**eta-**m**eta-**m**odel provides the concepts and constraints that a metamodel and the DSL has to satisfy, in our case provided by the language workbench *JetBrains MPS* [10]. M2, the **m**eta-**m**odel, provides abstractions to cover the domain of motion control architectures with motion primitives (the DSLs we introduce in the course of this paper). M1, the domain **m**odels, uses M2 to express applications or systems, exemplified in our case study in Section VI. Level M0 is the actual real-world robotics systems implementation, i.e. executable general purpose language (GPL) code that can be generated from M1. This contribution focuses on level M2 (Section III and Section V) and introduces the metamodel for motion architectures with *Adaptive Modules*. Section VI shows the levels M1 and M0 in a model-driven toolchain in the use-case depicted in Figure 1.

A. Nordmann, S. Wrede and J. Steil are with the Research Institute for Cognition and Robotics, Bielefeld University, P.O. Box 100131, Bielefeld, Germany {anordman,swrede,jsteil}@cor-lab.de

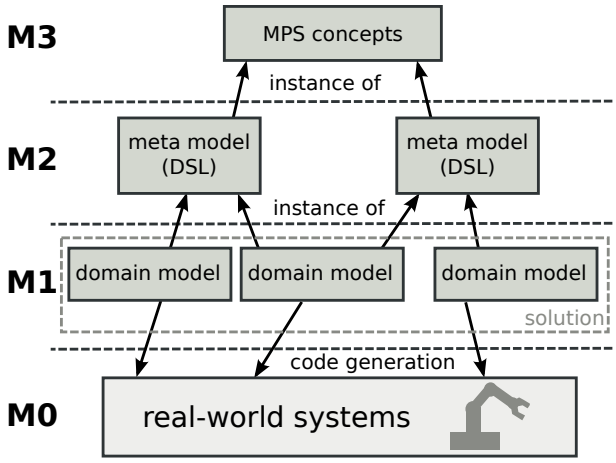



Fig. 2: Our four levels of model-driven engineering with domain-specific languages, following OMG's standard [9].

## II. Motion Control Architectures

This contribution targets the domain of control architectures for adaptive motion generation based on motion primitives and presents a conceptual framework for this domain. Typically, a robot is equipped with a set of predefined movements, especially in industrial robotics, where precision, speed and accuracy are important. Recent research, however, aims to extend motion generation on robotics to the biological richness of humans and animals.

The idea of the recent large-scale research project AMARSi[1] is to create a framework for rich motor skills by combining simple, but flexible and adaptive motion primitives to complex movements. The core idea is to model point-to-point and periodic movements by autonomous dynamical systems. The flexibility to adapt motion primitives to different situations and environments is accomplished by means of machine learning techniques.

The architectural building blocks representing motion primitives as the combination of dynamical systems and machine learning for adaption are termed *Adaptive Modules*. They resemble the computational building blocks to generate primitive, composable motions that can generalize to new situations or environments and are robust to perturbations. We consider control and learning architectures based on a hierarchy of Adaptive Modules as domain of our approach. Higher-level aspects like goals, motivations, planning or high-level perception are considered outside but adjacent to this domain.

### A. Domain Analysis

To assess the core concepts of our domain, we conducted a Feature-Oriented Domain Analysis [11] (FODA) on compliant robotics control among common robot control frameworks, interfaces of compliant robots and their applications. The resulting feature models are accompanied with a survey that we conducted together with the AMARSi project partners, covering state-of-the-art motion architectures based on motion primitives [8].

The domain analysis covered the following architectures and their respective experiments/demonstrators:

*a) Quadruped walking over unperceived rough terrain [12]:* A control architecture for walking and rebalancing the overall pose of the robot. The architecture is hierarchical and modular and couples together four basic mechanisms: i) A network of coupled central pattern generators (CPGs) (one per end-effector) to generate a periodic gait, ii) a reflex mechanism that modulates the shape of the target trajectories emitted by the CPG system if a leg hits an obstacle, iii) a proportional feedback controller for making the 12 joint angles track the output of the reflex-modulated CPG system, and iv) a higher-level, model-based control loop that stabilizes this pose when perturbed.

*b) Catching objects in flight [13]:* Catching objects with a catching point not located at the center of mass and highly non-linear dynamics, e.g. a tennis racket or a half-filled water bottle. This requires coordination between the arm reaching motion toward the predicted catching location and the hand/finger pose preparation for the actual catching, which is done by coupling two dynamical systems that have been trained individually using the coupled dynamical systems method.

*c) Mixture of controllers to learn inverse kinematics:* Control architecture inspired by the MOSAIC control architecture [14], but different controllers specialize not on different tasks, but on different regions in joint space [15].

*d) Redundancy learning [16]:* An approach utilizing the physical interaction capabilities of compliant robots with data-driven and model-free machine learning to allow fast (re)configuration of redundant robots in kinesthetic teaching. This approach facilitates a hybrid controller to join machine learning capabilities with analytical control.

*e) Humanoid upper body control [17]:* Control architecture for the iCub for three different bi-manual motion skills trained in physical human-robot interaction. Convergence of a discrete primitive triggers execution of subsequent primitives.

The analysis covered the range of point-to-point [12], [13], [17] and periodic movements [12], [17] modeled by autonomous dynamical systems and assessed functional and non-functional properties of their diverse *Adaptive Modules* and their implementations. *Functional* properties in this survey covered parametrization, data representation, learning algorithms (e.g. online vs. offline learning, supervised vs. unsupervised learning), required sensor feedback, etc. *Non-functional* properties covered implementation language, technical data representation, timing, software dependencies and availability, etc.

Results from the survey and the feature models resulting from the FODA led to the concepts and abstractions introduced in the following sections.

[1] AMARSi Project: http://www.amarsi-project.eu/

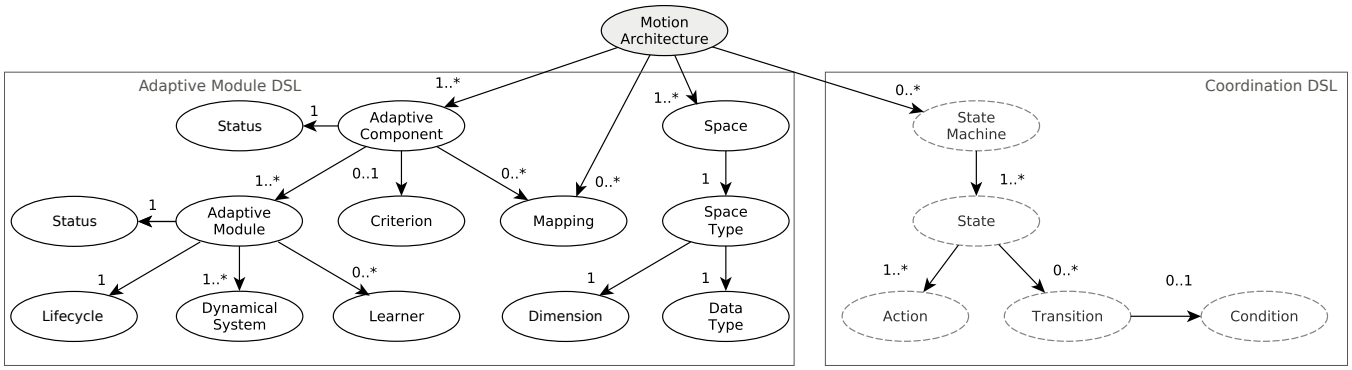



Fig. 3: Reduced metamodel of our motion architecture with the structural aspects of the Adaptive Module DSL, and the behavioral aspects of the Coordination DSL (gray, dashed). The main concepts are displayed as circles, and their *has-a* relations indicated as arrows with annotated cardinality (references between concepts are left out for the sake of clarity).

## III. Adaptive Modules

This section introduces a *conceptual framework* around the essential concept of the Adaptive Module[2], being the architectural building block to represent motion primitives and comprising dynamical systems and machine learning capabilities. A more formal metamodel of the concepts of a motion architecture in our framework is given in Figure 3.

The concept of Spaces in the Adaptive Module DSL abstracts the communication between system components. It is defined as a number of explicit variables that appear to be jointly manipulated or sensed somewhere in our motion control architecture, e.g. joint space commands for all joints of a certain robot limb. The Space Type is determined by its Data Type and Dimension. Possible data types are the ones found in the domain analysis, such as Joint Angles, Impedance, or end-effector Pose with their respective untis. Mappings map data between Spaces of different Space Type. A derived concept of a Mapping is a Transformation, defining the transformation of data between Spaces of the same Space Type (e.g. coordinate transformations), see figure 4. Mappings and Transformations are a potential extension point for dedicated models, e.g. geometric relations by Laet et al. [5]. Typical examples found in the domain are forward and inverse kinematics. Another specialization of Mappings are Adaptive Mappings that can be learned, as it was found in the redundancy resolution learning example [16] of the domain analysis.

[2]Domain concepts are denoted in typewriter font.

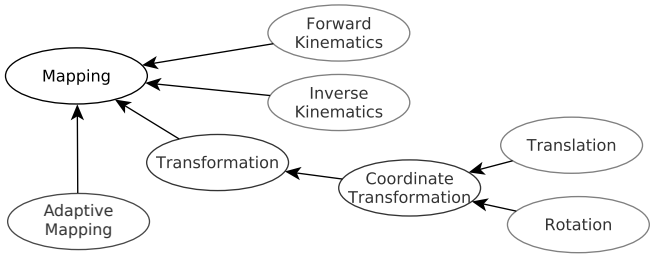



Fig. 4: Specializations of Mappings and Transformations.

An Adaptive Module is the main functional building block of the motion control architecture and represents a motion primitive, for example the gait pattern generators or reflexes used for the quadruped walking [12]. It contains one or more Dynamical Systems which generate the output dynamics. Dynamical Systems can either have periodic or non-periodic (goal-directed) dynamics. To allow adaption to new situations, new environments and additional learning input, Adaptive Modules optionally contain one or more Learners that adapt the Dynamical Systems. A common case for adaption is to *shape* the dynamics as done with kinesthetic teaching in the catching [13] and upper body control [17] examples. The Adaptive Module concept has ingoing connections from Spaces to receive goals during execution, sensor values or learning data (optional). It also has outputs to Spaces for its control output and status, and has a specific lifecycle (e.g. for execution, online and offline learning).

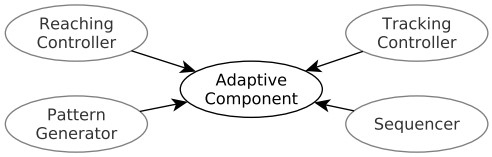



Fig. 5: Specializations of the Adaptive Component concept.

An Adaptive Component is an Adaptive Module together with its inputs and outputs, basic semantics (control logic) inside the component, and timing management. We define several specializations of Adaptive Components for the basic semantics found during the domain analysis, shown in Figure 5: Reaching Controller for goal directed movements, Tracking Controller to track changing targets as done in the mixture of controllers example [15], Sequencer for simple coordination and a Pattern Generator. Each with its specific logical wiring, determining the wiring of the contained elements and the wiring to the Spaces of their architectural context depending on its current state (e.g. learning and execution). This concept is based on the findings in our domain analysis that identified a limited

set of architectural structures that we found in the vicinity of Adaptive Modules. For this, Adaptive Components can be configured to have Mappings or Transformations on the goal input, feedback input or control output of the Adaptive Module. This allows integrating Adaptive Modules into different systems and platforms by configuring their containing Adaptive Components accordingly, if they were otherwise incompatible.

Adaptive Components can optionally specify a Criterion that indicates when the motion primitive is finished or, in case of a periodic motion primitive, following its attractor. A typical example is a Convergence Criterion for a reaching movement.

## IV. Concept Validation

The first means for validating our framework was to express the domain examples of the domain analysis introduced in Section II-A in the chosen abstractions, which was a continuous and iterative process while developing this framework and which was supported by the respective authors as part of the AMARSi project. Figure 6 shows two examples from this process. These two examples, amongst others, are re-formulated in the introduced conceptual framework in order to validate its capability to express applications of the domain.

Figure 6a shows the example of quadruped walking over unperceived rough terrain [12]. It can be formulated as one Pattern Generator (lightred) containing several coupled CPGs as Adaptive Modules, and additional Adaptive Modules for reflexes and posture control (red). An advantage over the native representation of the approaches is additional expressiveness, e.g. explicit Spaces to allow interfacing with other motion primitives. Figure 6b shows the mixture of controllers to learn inverse kinematics [15] as a second example. It is built of $n$ Tracking Controllers, including recurrent neural networks as Adaptive Modules and a scaling Transformation. Superposition of the Tracking Controller outputs results in the intended mixture of controllers.

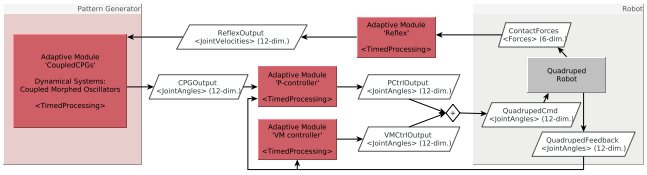
This qualitative validation shows that our framework is indeed capable to cover the functional variety of the surveyed motion architectures regarding their structural aspects. It also shows that behavioral aspects are missing in the system representations we surveyed in our domain analysis. Behavior is to a large extend not explicitly represented, but often hidden in the code of sequencing components. This comprises a major problem for our goal to formalize motion architectures, because these aspects are equally important for a comprehensive specification and for automatic generation of executable systems.

## V. Coordination

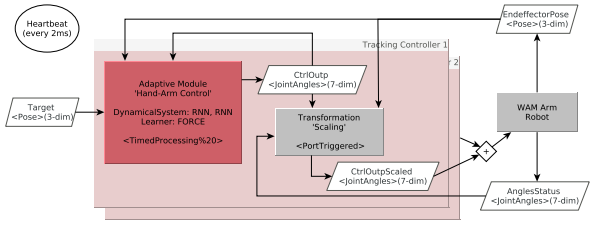
In order to express complex robotics systems, specification of behavioral aspects is necessary. We therefore introduce further abstractions and a further DSL dedicated to system coordination, as shown in Figure 3 (dashed, gray). The Coordination DSL is dedicated to express coordination aspects of motion control systems and uses the concepts of Harel state charts [18], which are common practice in coordination of robotics systems. We use the SCXML [19] model and provide additional extensions to integrate with the Adaptive Module DSL.

State Machine is the top-level abstraction of the Coordination DSL. It contains a number of States and the transitions between them. States define entry and exit Actions and conditional Transitions to other States. Actions define what is happening inside a state. As our language approach allows extension and re-use across DSLs, this is a first domain-specific extension point to the rather generic state chart concepts. Apart from generic actions like log messages, the language provides domain-specific actions like changing the learning state of an Adaptive Module or triggering execution of a motion primitive. Another action is to provide data for a certain Space, which can be used for example to provide goals for a motion primitive (resp. its corresponding Adaptive Module).

Transitions between states allow to change between states if certain Conditions are met. The Coordination DSL comprises rather generic conditions like clicking a button in the graphical user interface of the state machine or waiting a certain duration. Since it is integrated with the Adaptive Module DSL it can also use the domain-specific (motion-specific) synchronization mechanisms found in the domain analysis and provided by the Adaptive Module DSL concepts: Adaptive Modules, for example, report their status as either i) executing: it is not learning, but executing a movement, ii) online learning: The Learner is training the Dynamical Systems in online learning mode, allowing the Adaptive Module to be executed in parallel, or iii) offline learning: The Learner is training the Dynamical Systems in offline learning mode and the Adaptive Module is currently not available to perform a movement.




(a) Quadruped walking over unperceived rough terrain [12].




(b) Mixture of controllers to learn inverse kinematics [15].

Fig. 6: Two examples from the domain analysis, formulated in the introduced conceptual framework and auto-generated in a former version of our visual language.

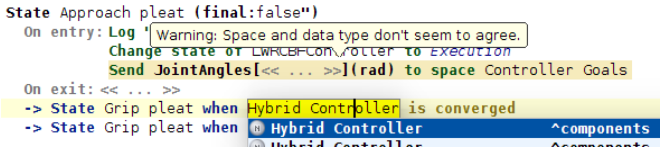



Fig. 7: Example of the Coordination DSL realized in MPS with domain-specific context help and model validation.

This is used for example to coordinate the different learning states in the redundancy learning example [16]. The state reported by `Adaptive Components` represents the status of the performed movement: i) `converged / tracking`: The reference of the movement is reached (in case of a goal-directed movement) or tracked (in case of a periodic movement) and ii) `ongoing`: Movement is still ongoing. This can be used for goal directed movements as done in the catching example [13].

These different status types are available in the Coordination DSL due to our domain-specific extensions and can be used as conditions for state transitions. This allows system coordination based on the movement and learning states of motion primitives, as well as triggering the execution of motions and learning steps. Coordination between the state machine and components, often hidden in code, implemented as state machine events and event-handlers, is now explicit in the DSL specification.

## VI. Case Study: Automatic Laundry Gripping

In addition to the conceptual validation of our approach in Section IV, we evaluated the technical feasibility of our framework in a model-driven development process and DSL toolchain in an actual use-case. This section sketches our toolchain, the specification of a robotics systems in our DSLs, and generation of the executable experiment code.

The use-case comprises a complex robotics setup combining the redundant and compliant KUKA Lightweight Robot IV, 3D perception and a number of calibration, human-robot interaction, vision, and learning modules. It is used in a real world application arising in the context of an innovation project within *it's OWL* [3], where the goal is to reduce time and costs for large automatic laundry washing facilities. The application requires to calibrate, interact, learn kinematics, identify pleats, and safely move the robot into a grasping position to feed the laundry into a further automatic process. Although quite typical for robotics application domains, systems of this complexity are not seen often in practice because manual programming and integration of such system would already be very challenging for *hand-crafted* development. It rather calls for systematic design methods as the one we propose.

In a first system state, an Adaptive Module *CalibrationLearner* is set into its learning state and learns the 6D transformation between a 3D camera and the KUKA LWR IV during a kinesthetic teaching phase, where a human samples the workspace in physical human-robot interaction with the end-effector. In a second system state, the Adaptive Component *RedundancyLearner* containing an *Extreme Learning Machine* (ELM) is configured to learn the desired redundancy resolution in different parts of the workspace. Coordination of the sub-states is done based on interaction forces of the robot and based on the learning states of the ELM Adaptive Module. The sub-states manage the component states of the *RedundancyLearner* in the different learning and interaction states.

The third system state is the actual execution where mainly the Adaptive Component named *HybridController* is active, moving the gripper to the grip poses given by the external 3D perception component while complying to the redundancy resolution learned previously by the *RedundancyLearner*. Substates like approaching, gripping, and opening the gripper are coordinated by using convergence of movements, which can be directly used in the Coordination DSL, as shown in the DSL snippet of an exemplary state in Figure 7.

We modeled the complete system involving physical human-robot interaction for calibration and training, 3D perception, and compliant robot control in our framework. A detailed system visualization is shown in Figure 8, which shows an auto-generated structural system view in Figure 8a and the behavioral view as well as the connections between the two views in Figure 8b.

### A. DSL Implementation

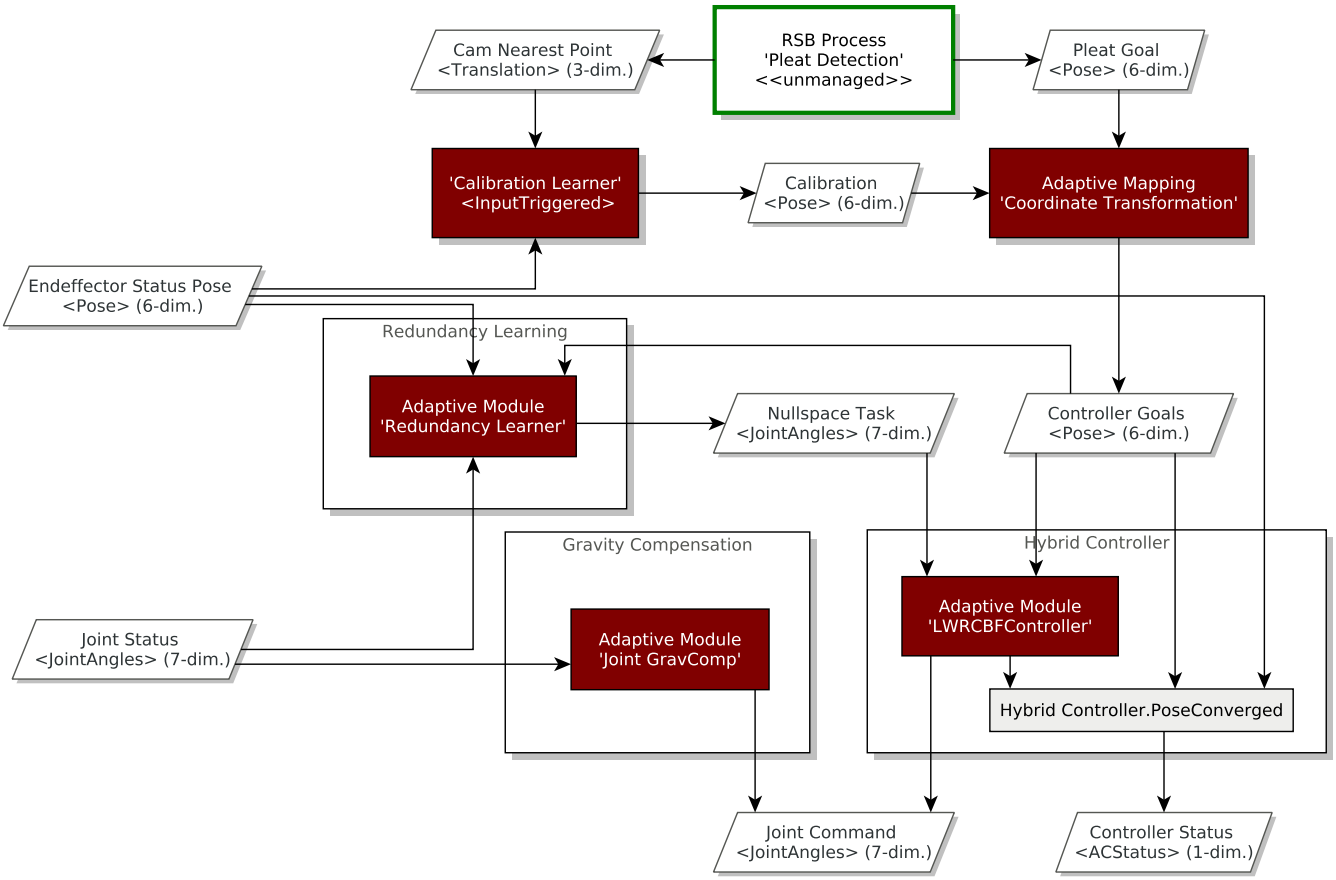
We have prototyped and built the two introduced DSLs in *JetBrains MPS* [10], a language workbench for building domain-specific languages and their projectional editors with explicit support for language modularization. We followed the language modularization and composition approach (LME&C) proposed in [20], which led the following two DSLs (among other DSLs [8]): the Adaptive Module DSL covering the structural aspects introduced in Section III and the behavioral aspects introduced in Section V covered by the Coordination DSL.

Figure 7 shows a snippet of the Coordination DSL editor of a system state being edited. Since both DSLs are integrated we have a structural model including the abstractions introduced in Section III and the behavioral model including the abstractions introduced in Section V combined. This allows to provide context help to the DSL user as well as model verification and validation across both aspects. Two examples are shown in Figure 7: i) (lower right) in the domain-specific condition of the state transition (convergence of a motion), only references to actual Adaptive Component of the structural model are allowed that provide a convergence criterion, ii) (top) within the action that provides a goal for an Adaptive Module, a warning about data type mismatch is raised due to the explicit Space specification.

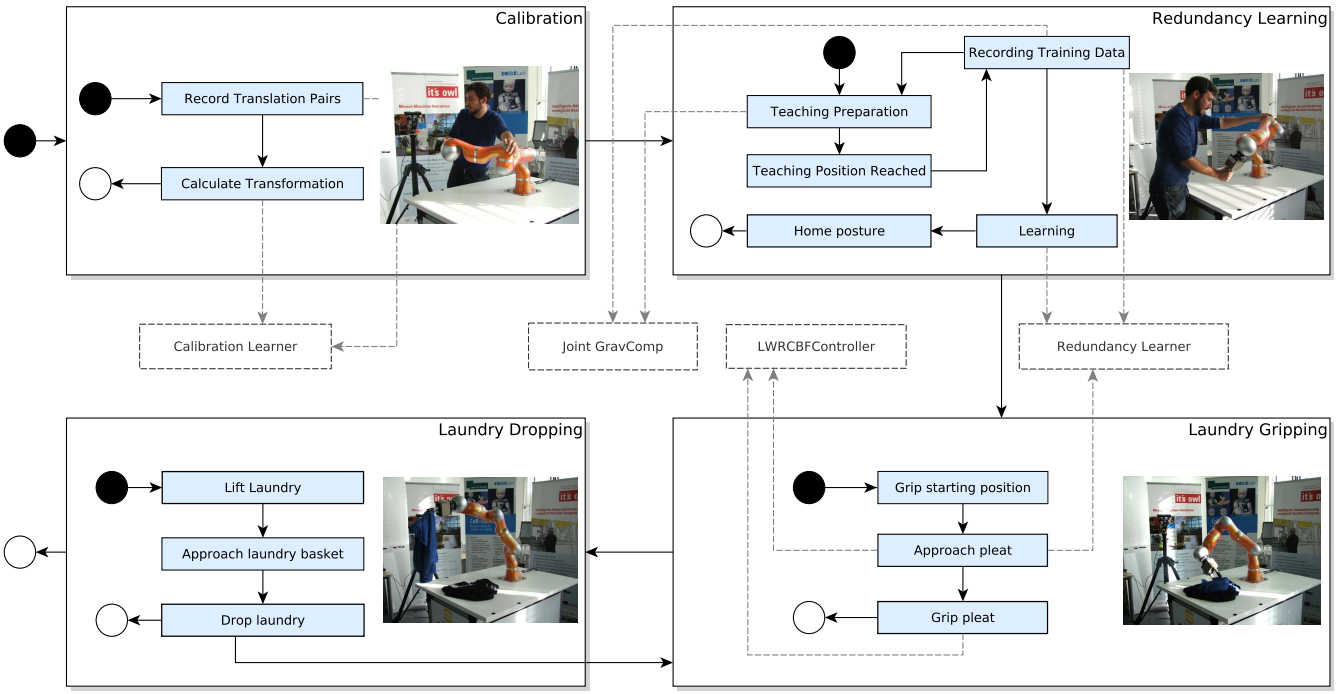
### B. Programming Model and Technology Mapping

The DSLs are integrated with our software architecture for evaluation our approach in a vertical prototype covering all aspects of a technical robotics system and eventually generate

[3]German national leading edge cluster "Intelligent Technical Systems OstWestfalenLippe" – http://www.its-owl.de




(a) Structural system view with the Adaptive Components as *white boxes*, Adaptive Modules in *dark-red*, Criterion as *grey box* and Spaces are shown as *parallelograms*. The *green box* on top shows the 3D perception.




(b) System states are shown in *blue*, *black arrows* represent state transitions, *gray-dashed arrows* indicate links to the structural model (*black-dashed*), either as condition for state transitions, or triggering different component and learning states in the Adaptive Modules.

Fig. 8: Auto-generated system visualization of the industrial use-case, manually layouted and reduced for the sake of clarity. The combination of the structural and behavioral models allows visualization of both aspects and their dependencies.

executable systems. Code generators of the DSLs target our domain-specific programming model that provides software abstractions for the concepts of our DSLs. Following best practices in robotics software engineering [21], [22], our programming model provides component-based software abstractions. We implemented an exemplary technology mapping of the programming model, which is targeted by the code generators. The C++ libraries *Compliant Control Architecture* (CCA) and *Robot Control Interface* (RCI) provide component-based software abstractions for compliant robot platforms [23] and implement the programming model. Communication is done via the robotics middleware RSB [24].

Note that this is just a reference technology mapping, as the DSL approach explicitly allows integration with other frameworks that are compatible with our programming model by providing different code generators. Those could target other frameworks and languages other than C++, e.g. UML to make a large amount of standard tools accessible.

### C. Code Generation and Experimentation

Code generators of the DSL environment target our domain-specific programming model and generate code in several general purpose languages (GPL). According to the chosen technology mapping, the generators transform the structural aspects of the specification of our use-case into graph representations of different aspects of the system for documentation (i.e. the rendered graphs shown in Figures 6 and 8), C++ code for the CCA components and dynamical systems, the C++ main file with the component configuration, the glue code for additional C++ components loaded from a software repository based on their deployment descriptors, and three CMake configuration files for software dependency handling and makefile generation. The behavioral aspects are transformed into an additional graph representation and *SCXML* [19] code that represents the specified state machine logic.

Furthermore, we generate several C++ files for the connection between the state machine and the structural components of the system. Note, how the cumbersome task of coordinating component state changes and state machine transitions can be taken care of by the generation tool chain.

Code generation of this case study generated 1 SCXML file, 4 GraphML [25] files, 12 C++ files with 1,248 source lines of code (SLOC), and 3 CMake files from the system specification done in 137 lines of Adaptive Module DSL specifying the structural part and 112 lines of Coordination DSL for coordination, a more compact and explicit specification of the system. See Table I for all numbers. The auto-generated (and manually adjusted) graph representation of the system, including its structural and behavioral aspects as well as their dependencies, rendered from the generated GraphML files is shown in Figure 8.

The generated C++ source code is compiled into two executables, the component circuit and the state machine, and was successfully executed and tested on the KUKA Lightweight Robot IV in simulation and on the hardware.

| | language | number of files | SLOC |
|---|---|---|---|
| **DSL** | Adaptive Module DSL | 1 | 137 |
| | Coordination DSL | 1 | 112 |
| **GPL** | SCXML | 1 | 360 |
| | GraphML | 4 | 4,455 |
| | CMake | 4 | 260 |
| | C++ | 12 | 1,248 |

TABLE I: Total numbers of SLOC of the DSLs (system specification) and the generated GPL code.

## VII. Discussion and Related Work

In this section we discuss our approach with respect to its conceptual differences and advantages and selected three state-of-the-art approaches for modeling and implementation of motion control architectures: i) MoveIt! [26] as a state-of-the-art motion planning framework, ii) MathWorks Simulink™ as an established tool for model-based design usable also for composition of computational units and modeling of dynamical systems, and iii) DSL-based approaches by Klotzbücher et al. [6] for force-velocity control and Vanthienen et al. [27] for constraint-based modeling and execution. We discuss these approaches from the viewpoint of the domain expert specifying a motion experiment to test a specific hypothesis.

DSL-based methods similar to Klotzbücher et al., Vanthienen et al. and the approach presented here explicitly represent domain concepts in dedicated models and elicit them from other platform-specific code. The integration of domain-specific modeling with a generator-based toolchain allows domain experts without software engineering background to formulate executable experiments in domain terms without touching any line of GPL code.

Writing experiments in terms of the domain abstractions using the projectional editing features of a DSL-IDE as shown in this work additionally helps developers to design models that are already largely correct by construction. The domain models facilitate checks on a semantic level, e.g. verifying compatibility of operations on *Spaces* or ensuring correctness properties of the *State Machines* and their relation to *Adaptive Components*, as it is exemplified in Figure 7. The language modularization approach which was briefly introduced in Section VI-A allows researchers and engineers from different sub-fields to focus on language abstractions suitable to express their particular concern. It also allows language re-use of certain concerns, as shown in this work by re-using the SCXML models for coordination, or dedicated coordinate transformation and mapping models as mentioned in Section III.

While domain-specific software frameworks like MoveIt! provide domain-specific abstractions, cf. MoveIt!'s waypoints and planners, those are usually in-language abstractions at code-level (in terms of object-oriented programming) and therefore only allow native interfacing using the implementation languages of the given framework based on the provided libraries. Verification and validation of experiments at the time of writing is hardly possible, as lots of the domain concepts are hidden in manually written code expressed with

the in-language abstractions lacking explicit semantics. The lack of accessible syntax and semantics makes the domain models also hardly accessible for analysis by additional tools.

General purpose languages and modeling environments as Simulink and Matlab do not directly provide conceptual abstractions for motion control architectures. Although quite established in the control domain, the specification of complete domain-specific experiments is a rather complex task as many extensions will be required that are not present in the rather general modeling languages of these tools. However, in the case of Simulink, domain-specific in-language abstractions can be added with dedicated tool boxes (e.g. for kinematics, machine vision, machine learning, etc.) that provide tool support, like simulation and online validation.

## VIII. Conclusion

This contribution introduces a conceptual framework based on the idea of *Adaptive Modules* that model motion primitives as dynamical systems with machine learning capabilities. The concepts are available in two domain-specific languages (DSL), which we integrated into a development process and toolchain that we evaluated in a case study. The framework and the Adaptive Module DSL were validated with a number of domain examples that showed that they are indeed able to cover the structural aspects of the domain and unify and express different research prototypes of formerly heterogeneous notation and semantics. Extensions to a Coordination DSL were introduced to cover the required behavioral aspects of complex systems and their integration with the structural system specification. A case-study showed a vertical prototype and demonstrated that we can use these DSLs and the accompanying toolchain in a model-driven development process. We specified a relevant, complex, real-world industrial application and generated the required source code to execute it on the KUKA LWR IV. While further research is necessary to judge the base hypothesis of motion primitives, we believe that the introduced concepts and DSLs may help to harmonize and accelerate research in the domain by supporting experimentation and evaluation.

## References


[1] Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[2] Arie van Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: An Annotated Bibliography. *ACM Sigplan Notices*, 2000.

[3] Arne Nordmann, Nico Hochgeschwender, and Sebastian Wrede. A Survey on Domain-Specific Languages in Robotics. In *International Conference on Simulation, Modeling and Programming for Autonomous Robots*, Bergamo, 2014.

[4] Marco Frigerio, Jonas Buchli, and Darwin G. Caldwell. Code generation of algebraic quantities for robot controllers. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2346–2351. IEEE, October 2012.

[5] Tinne De Laet, Wouter Schaekers, Jonas de Greef, and Herman Bruyninckx. Domain Specific Language for Geometric Relations between Rigid Bodies targeted to Robotic Applications. In *Workshop on Domain-Specific Languages and models for Robotic systems*, 2012.

[6] Markus Klotzbücher, Ruben Smits, Herman Bruyninckx, and Joris De Schutter. Reusable Hybrid Force-Velocity controlled Motion Specifications with executable Domain Specific Languages. In *International Conference on Intelligent Robots and Systems*, pages 4684–4689, 2011.

[7] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. *Robotics Research*, pages 1–10, 2005.

[8] Arne Nordmann and Sebastian Wrede. A Domain-Specific Language for Rich Motor Skill Architectures. In *Workshop on Domain-Specific Languages and models for Robotic systems*, Tsukuba, 2012.

[9] OMG. Object Management Group. *http://www.omg.org/*.

[10] JetBrains. Meta Programming System. *http://www.jetbrains.com/mps/*.

[11] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA)-Feasibility Study. 1990.

[12] Mostafa Ajallooeian, Soha Pouya, Alexandre Tuleu, Alexander Sproewitz, and Auke J Ijspeert. Towards Modular Control for Moderately Fast Locomotion over Unperceived Rough Terrain. *Dynamic Walking*, (2008):2013, 2013.

[13] Ashwini Shukla and Aude Billard. Coupled dynamical system based hand-arm grasp planning under real-time perturbations. *Robotics: Science and Systems*, 2011.

[14] Masahiko Haruno, Daniel Wolpert, and Mitsuo Kawato. Mosaic model for sensorimotor learning and control. *Neural computation*, 13:2201–2220, 2001.

[15] Tim Waegeman, Michiel Hermans, and Benjamin Schrauwen. MACOP modular architecture with control primitives. *Frontiers in computational neuroscience*, 7(July):99, 2013.

[16] Arne Nordmann, Christian Emmerich, Stefan Rüther, Andre Lemme, Sebastian Wrede, and Jochen J. Steil. Teaching Nullspace Constraints in Physical Human-Robot Interaction using Reservoir Computing. In *International Conference on Automation and Robotics*, 2012.

[17] Rene Felix Reinhart and Jochen Jakob Steil. Learning whole upper body control with dynamic redundancy resolution in coupled associative radial basis function networks. *IEEE International Conference on Intelligent Robots and Systems*, pages 1487–1492, 2012.

[18] David Harel and Michal Politi. *Modeling Reactive Systems with Statecharts: The Statemate Approach*. McGraw-Hill, Inc., New York, NY, USA, 1998.

[19] W3C. State Chart XML (SCML): State Machine Notation for Control Abstraction, 2014.

[20] Markus Völter. Language and IDE Modularization, Extension and Composition with MPS. *GTTSE*, 2011.

[21] Davide Brugali, Alex Brooks, Anthony Cowley, Carle Côté, Antonio Domínguez-Brito, Dominic Létourneau, Françis Michaud, and Christian Schlegel. *Software Engineering for Experimental Robotics*, volume 30 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[22] Markus Klotzbücher, Nico Hochgeschwender, Luca Gherardi, Herman Bruyninckx, Gerhard Kraetzschmar, Davide Brugali, Azamat Shakhimardanov, Jan Paulus, Michael Reckhaus, Hugo Garcia, Davide Faconti, and Peter Soetens. The BRICS Component Model: A Model-Based Development Paradigm For Complex Robotics Software Systems. In *Symposium on Applied Computing (SAC)*, Coimbra, Portugal, 2013.

[23] Arne Nordmann, Matthias Rolf, and Sebastian Wrede. Software Abstractions for Simulation and Control of a Continuum Robot. In Itsuki Noda, Noriaki Ando, Davide Brugali, and James J. Kuffner, editors, *Simulation, Modeling, and Programming for Autonomous Robots*, volume 7628 of *Lecture Notes in Computer Science*, pages 113–124, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[24] Johannes Wienke and Sebastian Wrede. A Middleware for Collaborative Research in Experimental Robotics. In *International Symposium on System Integration*, pages 1183–1190, Kyoto, 2011.

[25] GraphML. http://graphml.graphdrawing.org/.

[26] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit! [ROS Topics]. *IEEE Robotics Automation Magazine*, (March):18–19, 2012.

[27] Dominick Vanthienen, Markus Klotzbücher, Joris De Schutter, Tinne De Laet, and Herman Bruyninckx. Rapid Application Development of Constrained-Based Task Modelling and Execution using Domain Specific Languages. In *International Conference on Intelligent Robots and Systems*, 2013.