A Domain-Specific Language and Simulation Architecture for Motor Skill Exploration*

Arne Nordmann[1] and Alexandre Tuleu[2] and Sebastian Wrede[1]

I. Introduction

A core idea of current European robotics research[3] is that complex motor skills arise from combining adaptable movement primitives. This is challenging not only for reasons of the intrinsic complexity of the underlying control and architectural problems, but also due to the technological and conceptual fragmentation of the robotics domain. Our answer to this challenge is to establish a model-driven and domain-specific approach around domain-specific language (DSL) technologies.

A key to handling the complexity of robotics systems is the separation of concerns regarding the *functional architecture*, and its *software architecture*. We present in this contribution a model-based approach separating and representing these concerns explicitly with dedicated domain-specific languages and model transformations [5]. We discuss how this contributes to an improved hypothesis test cycle and easier design space exploration with code generation targeting robotics software architectures, in this example integrated with a quadruped robot platform, shown in Figure 1.

II. Domain-Specific Hypothesis Test Cycle

We aim to establish a *model-driven development process* [7], [10] organizing efforts of the partners working on rather theoretical aspects, hardware and software concepts in a way which allows eased communication, early technical integration, quantitative as well as qualitative validation and later also for automation of engineering tasks to ease experimentation and scientific analysis. A central goal of our work is to support the development of motion control architectures to ease design space exploration for experimental robotics. This requires to support the entire experimental toolchain ranging from purely functional modeling to software architectural and technical aspects such as software deployment.

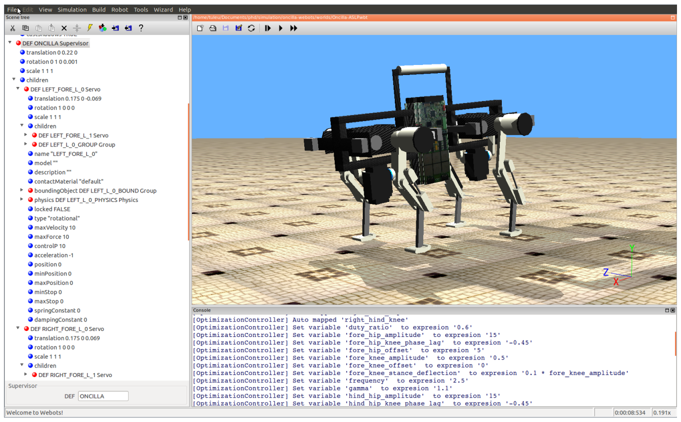
The current state of our DSL development divided the overall meta-model into a set of domain-specific languages, following a *language modularization, extension and composition* approach [8] (LME&C), to separate the DSLs by concerns.

*This work was partially funded by the European FP7 project AMARSi (grant no. 248311).

[1]A. Nordmann and S. Wrede are with the Research Institute for Cognition and Robotics, Bielefeld University, 33615 Bielefeld, Germany `{anordman, swrede} at cor-lab.de`

[2]A. Tuleu is with the BioRob Lab at EPFL, 1015 Lausanne, Switzerland `alexandre.tuleu at epfl.ch`

[3]For instance, as currently explored in the FP7 project AMARSi: http://www.amarsi-project.eu/



```
DEF ONCILLA Supervisor
  translation 0 0.22 0
  rotation 0 1 0 0.001
  scale 1 1 1
  children
    DEF LEFT_FORE_L_0 Servo
      translation 0.175 0 -0.069
      rotation 1 0 0 0
      scale 1 1 1
      children
        DEF LEFT_FORE_L_1 Servo
        DEF LEFT_L_0_GROUP Group
      name "LEFT_FORE_L_0"
      model ""
      description ""
      contactMaterial "default"
      boundingObject DEF LEFT_L_0_BOUND Group
      physics DEF LEFT_L_0_PHYSICS Physics
      locked FALSE
      type "rotational"
      maxVelocity 10
      maxForce 10
      controlP 10
      acceleration -1
      position 0
      minPosition 0
      maxPosition 0
      minStop 0
      maxStop 0
      springConstant 0
      dampingConstant 0
    DEF RIGHT_FORE_L_0 Servo
      translation 0.175 0 0.069
      rotation 1 0 0 0
      scale 1 1 1
      children
        DEF RIGHT_FORE_L_1 Servo
Supervisor
DEF ONCILLA
Welcome to Webots!
```

```
[OptimizationController] Auto mapped 'right_hind_knee'
[OptimizationController] Set variable 'duty_ratio'  to expresion '0.6'
[OptimizationController] Set variable 'fore_hip_amplitude'  to expresion '15'
[OptimizationController] Set variable 'fore_hip_knee_phase_lag'  to expresion '-0.45'
[OptimizationController] Set variable 'fore_hip_offset'  to expresion '5'
[OptimizationController] Set variable 'fore_knee_amplitude'  to expresion '0.5'
[OptimizationController] Set variable 'fore_knee_offset'  to expresion '0'
[OptimizationController] Set variable 'fore_knee_stance_deflection'  to expresion '0.1 * fore_knee_amplitude'
[OptimizationController] Set variable 'frequency'  to expresion '2.5'
[OptimizationController] Set variable 'gamma'  to expresion '1.1'
[OptimizationController] Set variable 'hind_hip_amplitude'  to expresion '15'
[OptimizationController] Set variable 'hind_hip_knee_phase_lag'  to expresion '-0.45'
```

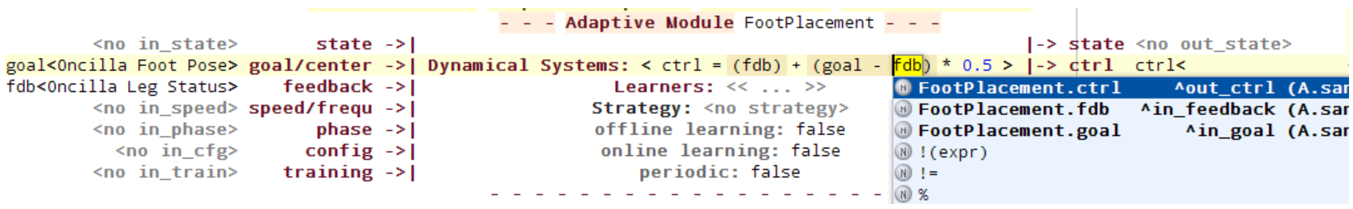
Fig. 1: Screenshot of the Oncilla simulator

The two main DSLs separate the domain along the formulation of the functional architecture, and the component-based software architecture:

The *AMARSi DSL* [5] is designed to allow compact structural description of motor control architectures. It expresses motor control systems as combinations of so-called *movement primitives*, that model flexible and adaptive movements in terms of dynamical systems. The functional building blocks of the language to generate movement primitives are the `Adaptive Module`, wrapping `Dynamical Systems` and optionally machine learning methods to adapt the `Adaptive Module` to new situations and environments. It also provides abstractions of recurring architectural patterns around `Adaptive Modules`, which we identified during the domain analysis representing different types of control strategies and internal connectivity. The AMARSi DSL mainly considers functional concepts, while hiding software architectural aspects.

The *Component DSL* instead covers software architectural aspects, organizing the system in `Components` with `States`, `Ports`, and their wiring (similar to BCM [2]).

Both DSLs are designed and to be used with *JetBrains MPS* [1], a language workbench for building DSLs and accompanying IDEs supporting model development with projectional editing. Model to model transformations between the AMARSi DSL and the Component DSL maps functional system models to software concepts. This allows for instance the automatic generation of two different views on a movement architecture: the functional (see Fig. 3) and the component-based view (see Fig. 4).



```
                              - - - Adaptive Module FootPlacement - - -
      <no in_state>       state ->|                                                 |-> state <no out_state>
goal<Oncilla Foot Pose> goal/center ->| Dynamical Systems: < ctrl = (fdb) + (goal - fdb) * 0.5 > |-> ctrl   ctrl<
fdb<Oncilla Leg Status>   feedback ->|               Learners: << ... >>   FootPlacement.ctrl     ^out_ctrl (A.sar
      <no in_speed> speed/frequ ->|              Strategy: <no strategy>   FootPlacement.fdb   ^in_feedback (A.sar
      <no in_phase>       phase ->|           offline learning: false      FootPlacement.goal     ^in_goal (A.sar
        <no in_cfg>      config ->|            online learning: false      !(expr)
      <no in_train>    training ->|                 periodic: false        !=
                                      - - - - - - - - - - - - - - - - - -   %
```

Fig. 2: `Adaptive Module` with its inputs and outputs for goal, feedback and control output. It contains a `Dynamical System` that is specified by an mathematical expression referencing inputs and outputs.

Besides modeling functional as well as software structures and generating system visualizations for improved comprehensibility, the toolchain also supports the generation of software artifacts directly applicable in robotics experiments. Targets of the Component DSL code generators (for C++) are the Compliant Control Architecture (CCA) and the Robot Control Interface (RCI) providing component-based software abstractions for experimental robot platforms [4]. Results of the code generation step are executable motion controllers, which are integrated with the sensors and actuators of advanced robots such as the quadruped robot described in the following section.

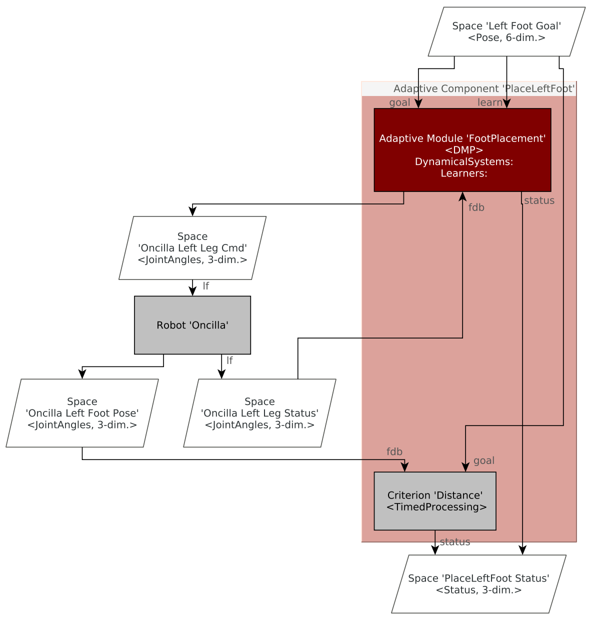
III. The Oncilla Robot as Validation Platform

The introduced DSL-based development approach has been co-developed and integrated with the Oncilla robot (an extended version of [6]). In order to quickly test a design specified in DSL models, a robot simulator has been created with a precise description of the robot kinematics and dynamics, including the particular parallel compliance and asymmetric actuation of the ASLP mechanism [6]. Figure 1 shows the model inside the Webots [3] simulation frontend. On top of the Webots model, an CCA/RCI-based interface with a taxonomic representation of the robot joints has been designed. One of its main design goal was a common abstraction between hardware and simulation, with binary compatibility. This interface is locally accessible through the AMARSi software architecture via a C++ interface, using multiple inheritance to expose the node taxonomy. It is also remotely available through RSB [9], an open-source middleware with C++, Java, Python and Common Lisp bindings for extended language and tool support.

As an example of how we foresee the DSL tool-chain to speed-up the hypotheses test cycle, Figure 2 shows writing a dynamical system as formula as a DSL expression in the projectional AMARSi DSL editor to be tested on the Oncilla platform. Proper scoping and integration in a type-system allows writing the expression with referencing inputs and outputs of the surrounding `Adaptive Module`. Furthermore, additional Adaptive Module implementations can be easily integrated with this new dynamical system at the DSL level and subsequently tested on the Oncilla robot.

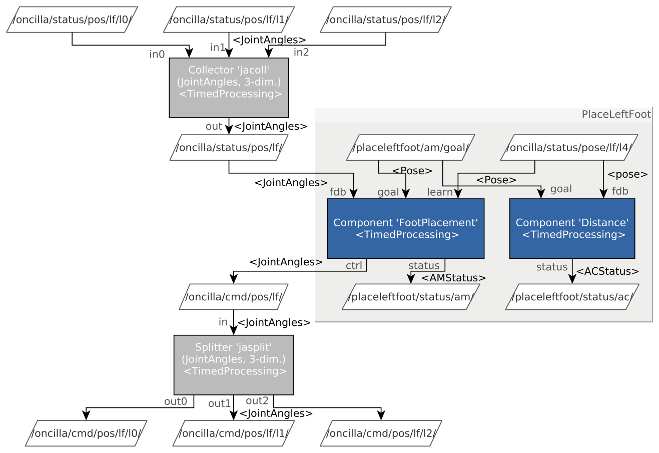
IV. Discussion

The presented approach facilitates a clean separation between functional and software architectural aspects and further concerns such as platform independence. It lifts algorithmic and system parameters typically hidden in software artifacts to a specification level. Experimentation is sped up as long as developers stay within the modeled domain.



Space 'Left Foot Goal' <Pose, 6-dim.>
Adaptive Component 'PlaceLeftFoot'
goal
learn
Adaptive Module 'FootPlacement' <DMP> DynamicalSystems: Learners:
fdb
status
Space 'Oncilla Left Leg Cmd' <JointAngles, 3-dim.>
lf
Robot 'Oncilla'
lf
Space 'Oncilla Left Foot Pose' <JointAngles, 3-dim.>
Space 'Oncilla Left Leg Status' <JointAngles, 3-dim.>
fdb
goal
Criterion 'Distance' <TimedProcessing>
status
Space 'PlaceLeftFoot Status' <Status, 3-dim.>

Fig. 3: Functional view in AMARSi concepts.



/oncilla/status/pos/lf/l0/
/oncilla/status/pos/lf/l1/
/oncilla/status/pos/lf/l2/
<JointAngles>
in0
in1
in2
Collector 'jacoll' (JointAngles, 3-dim.) <TimedProcessing>
out
<JointAngles>
PlaceLeftFoot
/oncilla/status/pos/lf/
/placeleftfoot/am/goal/
/oncilla/status/pose/lf/l4/
<JointAngles>
<Pose>
<Pose>
<pose>
fdb
goal
learn
goal
fdb
Component 'FootPlacement' <TimedProcessing>
Component 'Distance' <TimedProcessing>
<JointAngles>
ctrl
status
<AMStatus>
status
<ACStatus>
/oncilla/cmd/pos/lf/
/placeleftfoot/status/am/
/placeleftfoot/status/ac/
in
<JointAngles>
Splitter 'jasplit' (JointAngles, 3-dim.) <TimedProcessing>
out0
out1
out2
<JointAngles>
/oncilla/cmd/pos/lf/l0/
/oncilla/cmd/pos/lf/l1/
/oncilla/cmd/pos/lf/l2/

Fig. 4: Component-based system view in the concepts of the Component DSL.

DSL-based approaches still heavily rely on complex tools such as language workbenches, which makes interoperability at the level of DSLs an open issue. Projectional editing is a powerful concept, but usability for users from non-technical domains still has to be improved. An important aspect for the presented approach has been the integrability of existing concept implementations in legacy code and their dynamic representation at the language level which we found to be well supported in MPS.

Future work will focus on improved representation of dynamic model aspects and an improved expression language to directly specify dynamical systems as well as improved code generation.

REFERENCES

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

[2] M. Klotzbuecher, N. Hochgeschwender, L. Gherardi, H. Bruyninckx, G. Kraetzschmar, D. Brugali, A. Shakhimardanov, J. Paulus, M. Reckhaus, H. Garcia, D. Faconti, and P. Soetens. The BRICS Component Model: A Model-Based Development Paradigm For Complex Robotics Software Systems. In *28th ACM Symposium on Applied Computing (SAC)*, Coimbra, Portugal, 2013.

[3] O. Michel. Webots: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):39—-42, 2004.

[4] A. Nordmann, M. Rolf, and S. 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.

[5] A. Nordmann and S. Wrede. A Case-Study on a Domain-Specific Language for Robotics Motion Architectures. In *ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, page submitted, 2013.

[6] A. Sproewitz, A. Tuleu, M. Vespignani, M. Ajallooeian, E. Badri, and A. Ijspeert. Towards Dynamic Trot Gait Locomotion - Design, Control and Experiments with Cheetah-cub, a Compliant Quadruped Robot. *International Journal of Robotics Research*, submitted, 2013.

[7] M. Völter. A Pattern Language for Building Sustainable Software Architectures. *Computer methods and programs in biomedicine*, September 2005.

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

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

[10] J. Zhang and B. Cheng. Model-based development of dynamically adaptive software. In *ICSE 2006*, 2006.