Towards a Model-Centric Cognitive Architecture for Service Robots

Andreas Steck
University of Applied Sciences Ulm
Department of Computer Science*, 89075 Ulm, Germany
steck@hs-ulm.de

Abstract

The development of service robots has gained more and more attention over the last years. Advanced robots have to cope with many different situations and contingencies while executing concurrent and interruptable complex tasks. To manage the sheer variety of different execution variants the robot has to decide at run-time for the most appropriate behavior to execute. That requires task coordination mechanisms that provide the flexibility to adapt at run-time and allow to balance between alternatives.

1 Introduction, Motivation and Related Work

Recent advances in robotics and cognitive sciences have stimulated expectations for emergence of a new generation of robotic devices that interact and cooperate with people in ordinary human environments. Robotic systems which should be able to operate in everyday life have to manage the high dynamics and complexity of real-world environments. The huge amount of different situations and contingencies becomes overwhelming and can hardly be pre-programmed, even by the most skilled robot engineer. The sheer variety of execution variants in open environments does neither allow an optimal solution nor an exhaustive assignment of proper reactions to situations. Thus, the robot has to balance between different alternatives and to decide for the most appropriate one at run-time. Reliably performing complex tasks over long periods of time requires task coordination mechanisms which can manage complexity as well as address robustness. Therefore, at design-time the developer specifies variation points and corridors in which the robot can make decisions and operate at run-time. Binding left open variation points at run-time may require to consult simulators, symbolic planners or analysis tools. This demands for explicit descriptions of relevant properties and parameters of the robot, its resources and its capabilities. Different views on partial aspects of a robot (mechanical, electrical and even software) can be provided by different models as is already common practice at design-time. However, these design-time models also need to be accessible at run-time to support run-time reasoning in order to take advantage of the information which is explicated in the models. Therefore, useful information needs to be extracted out of the design-time models and transformed into representations which can be exploited at run-time. Exploiting those resources demands for effective coordination and planning capabilities. [Schlegel *et al.*, 2010]

Managing execution variants and binding left-open variation points at run-time requires extensions of task coordination mechanisms. Task coordination languages [Verma *et al.*, 2005] have successfully been introduced into robotics, but have been developed with limited robot platforms and capabilities. The power of those concepts can now be exploited and the progress in robotics allows to proceed the development of those ideas which have strongly influenced SMARTTCL, which is part of this work.

In *CRAM* [Beetz *et al.*, 2010] a two layer architecture is proposed to coordinate the skills. *CRAM* relies on reasoning mechanisms to infer control decisions rather than preprogramming them. In the here proposed approach the focus is to take advantage from prior knowledge and to provide this as procedural knowledge in form of reusable action plots. At design-time purposefully left open variation points are bound just before execution. This balancing between preprogramming, reasoning and planning provides flexibility in given situations as well as robustness.

2 The overall Approach

The approach is to provide task coordination mechanisms that can manage the huge amount of different execution variants by exploiting design-time models at run-time. Simulators, symbolic planners and analysis tools are, for example, used to bind purposefully left open variation points. That requires to bring together ideas and approaches from currently separated technologies including robotics, cognitive sciences and software engineering. The different aspects and views are represented by different models created with different tools which are based on different representations. As each tool has its specific representation the models have to be transformed accordingly. Model transformations are widely used in the domain of software engineering and a huge community provides tools like the *Eclipse Modeling Project*, which could be tailored to robotics as already done, for example, in [Schlegel *et al.*, 2010] without re-inventing the wheel.

*Collaborative Center of Applied Research on Service Robotics (ZAFH Service Robotics) www.zafh-servicerobotik.de

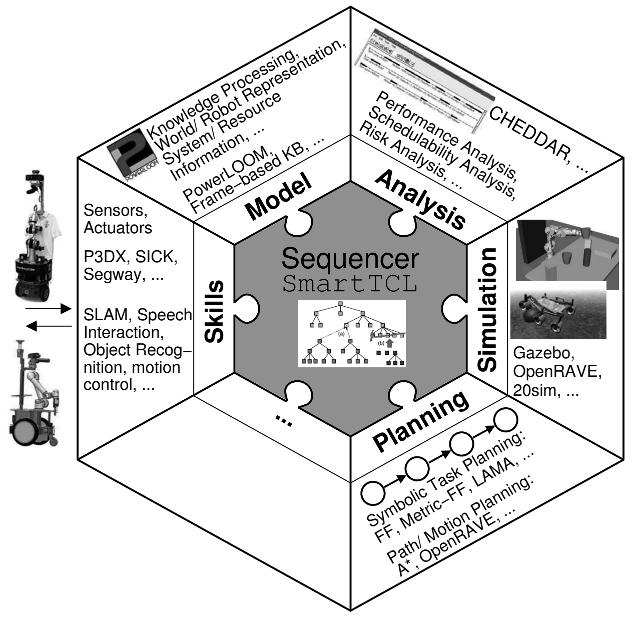

Figure 1: The sequencer coordinates the overall system.

In the SMART ROBOTICS ARCHITECTURE (SARA) the sequencer has the control over the whole system (fig. 1). The sequencer is implemented with the TASK COORDINATION LANGUAGE (SMARTTCL) [Steck and Schlegel, 2010], which is an extension to *Common Lisp*. It bridges the gap between continuous processing and event driven task execution and mediates between symbolic and subsymbolic mechanisms of information processing. Subsymbolic mechanisms ensure flexibility and reactivity since they typically allow short cycle times. Symbolic mechanisms ensure goal oriented activities and are able to reason about the steps which are necessary to achieve higher-level goals. The sequencer is the place to store procedural knowledge on how to configure skills to behaviors, when to use a symbolic task planner and what kind of action plots are suitable to achieve certain goals. It supports hierarchical task decomposition and situation-driven task execution. At run-time a task-tree is dynamically created and modified depending on situation and context. The action plots in SMARTTCL perform dynamic online reconfigurations and orchestration of the software components of the whole system, including skills, analysis, planning and simulation. They furthermore have encoded how to access the different models, how to extract partial views and how to transform between them at run-time.

Symbolic task planners like *Metric-FF* and *LAMA* are, for example, used to bind left open variation points at run-time. The developer specifies the *PDDL* models at design-time which are enriched with the *facts*, extracted from the models (e.g. *KB*) at run-time. Which planner to use, what partial knowledge to forward to it and how to import the generated plan into the task tree is encoded in the action plot of the node which is responsible to bind the specific variation point. Executing the generated plan is thus in the responsibility of SMARTTCL and consequently its contingency handling mechanisms are applied to ensure that the execution remains in the specified corridor. The responsible node decides whether the failure can be fixed locally or a complete new plan has to be generated.

Another example how to bind left open variation points at run-time is the integration of simulators. A node in the task tree calls, for example, a physics simulator to experience the appropriate maximum velocities of a forklift robot taking the current payload into account.

SARA has been used to build and run several real-world scenarios. This includes, for example, cleaning up a table. Several contingencies, like no object could be found on the table, an object could not be grasped and problems in the path-planning are handled. The objects include, for example, cups that could be stacked into each other and have to be thrown into the kitchen sink and beverage and crisp cans that have to be thrown into the trash bin. A beverage can, can be stacked into a crisp can. The left open variation point which objects to stack into each other is bound by the symbolic task planner *Metric-FF* depending on the set of recognized objects and the constraints stored in the *KB*.

3 Conclusion and Future Work

This work contributes to a novel approach for robotic system engineering bridging the gap between design-time and run-time model-usage for exploiting purposefully left open variation points for run-time decisions. Using a sequencer as the central coordination mechanism proved to accomplish the goal to flexibly manage and orchestrate available skills and resources to master the huge amount of execution variants at run-time. Future work will extend the task coordination and selection process and further integrate run-time model usage as well as *Quality of Service (QoS)* aspects.

References

[Beetz *et al.*, 2010] M. Beetz, L. Mösenlechner, and M. Tenorth. CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments. In *IEEE/RSJ Int. Conf. on Intelligent RObots and Systems (IROS)*, Taipei, Taiwan, 2010.

[Schlegel *et al.*, 2010] C. Schlegel, A. Steck, D. Brugali, and A. Knoll. Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering. In *Int. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)*, volume 6472 of *LNCS*, pages 324–335. Springer, Darmstadt, Germany, 2010.

[Steck and Schlegel, 2010] A. Steck and C. Schlegel. SmartTCL: An Execution Language for Conditional Reactive Task Execution in a Three Layer Architecture for Service Robots. In *Int. Workshop on DYnamic languages for RObotic and Sensors systems (DYROS/SIMPAR)*, pages 274–277. Springer, Darmstadt, Germany, 2010.

[Verma *et al.*, 2005] V. Verma, A. Jónsson, R. Simmons, T. Estlin, and R. Levinson. Survey of Command Execution Systems for NASA Spacecraft and Robots. In *"Plan Execution: A Reality Check" Workshop at The Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2005.