Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering

Conference Paper · November 2010

DOI: 10.1007/978-3-642-17319-6_31 · Source: DBLP

4 authors, including:

Christian Schlegel
Hochschule Ulm
72 PUBLICATIONS **889** CITATIONS
SEE PROFILE

Davide Brugali
University of Bergamo
87 PUBLICATIONS **940** CITATIONS
SEE PROFILE

Design Abstraction and Processes in Robotics: From Code-Driven to Model-Driven Engineering

Christian Schlegel[1], Andreas Steck[1], Davide Brugali[2], and Alois Knoll[3]

[1] Department of Computer Science, University of Applied Sciences Ulm
89075 Ulm, Germany
{schlegel,steck}@hs-ulm.de
[2] Department of Computer Science and Mathematics, University of Bergamo
Dalmine 24044, Italy
brugali@unibg.it
[3] Department of Informatics, Technische Universität München
85748 Garching bei München, Germany
knoll@in.tum.de

Abstract. Advanced software engineering is the key factor in the design of future complex cognitive robots. It will decide about their robustness, (run-time) adaptivity, cost-effectiveness and usability.

We present a novel overall vision of a model-driven engineering approach for robotics that fuses strategies for *robustness by design* and *robustness by adaptation*. It enables rigid definitions of quality-of-service, re-configurability and physics-based simulation as well as for seamless system level integration of disparate technologies and resource awareness.

We report on steps towards implementing this idea driven by a first robotics meta-model with first explications of *non-functional properties*. A model-driven toolchain provides the model transformation and code generation steps. It also provides design time analysis of resource parameters (e.g. schedulability analysis of realtime tasks) as step towards *resource awareness* in the development of integrated robotic systems.

1 Introduction

The difference of robotics compared to other domains is, e.g., neither the huge number of different sensors and actuators nor the diversity of hardware- and software platforms. It is the *deviation between design-time and run-time optimality* due to the overwhelming size and tremendous complexity of the *problem space* and the *solution space*.

The problem space is huge: as uncertainty of the environment and the number and type of resources available to the robot increase, the definition of the best matching between current situation and correct robot resource exploitation becomes overwhelming, even for the most skilled robot engineer. *The solution space is huge*: in order to enhance robustness of complex robotic systems, existing cognitive methods and techniques need to adequately exploit robotic-specific resources. This means that the robotic engineer should master highly heterogeneous technologies in order to integrate them in a consistent and effective way.

N. Ando et al. (Eds.): SIMPAR 2010, LNAI 6472, pp. 324–335, 2010.

The overall objectives of the proposed novel robot development process are: (i) to promote the synergy between robotics, cognitive system engineering, and software engineering in order to face with new instruments the challenges to build next generation cognitive robotic systems, (ii) to reduce time and cost of the entire development process from requirements specifications of new robotic systems to the reuse of existing software and hardware resources, (iii) to enhance robotic system ability to exploit at best internal and external resources according to the dynamic evolution of the operational context, (iv) to facilitate experimentation of advanced cognitive technologies with increasingly complex robotic platforms and in real-world application scenarios.

In order to achieve this goal, we propose a new design process that aims to reduce both the robotic system engineering problem space and solution space.

2 Design Abstraction for Robotic System Engineering

The *solution space can be managed* by giving the robot engineer the ability to formally model and relate the different views relevant to robotic system design (including their intrinsic variability) and to reason about their correctness by means of offline simulation techniques (i.e., *robustness by design*).

The *problem space can be mastered* by giving the robot the ability to reconfigure its internal structure and to adapt the way its resources are exploited according to its understanding of the current situation (i.e., *robustness by adaptation*). This can be possible thanks to the adoption of online model simulation and (re)configuration techniques.

We think that both *robustness by design* and *robustness by adaptation* can be achieved by raising the level of abstraction at which both, the system engineer and the robot reason about the problem and the solution spaces. This means to rely, as for every engineering endeavour, on the power of models.

Robustness by design refers to the definition of a novel software and system development process that leverages the power of software models to seamlessly and consistently transform application-specific requirements to platform-specific control systems and to validate each individual design decision by means of simulation tools. *Robustness by adaptation* refers to the ability of the control system to reason about and manipulate the exact same software models at run-time in order to autonomously adapt them according to changing operational conditions and to casually reflect the adaptations of the models in the run-time system.

This innovative robot development process illustrated in fig. 1 moves the focus from trying to find all-time optimal solutions at design-time to making the best possible decisions at run-time and thus providing the foundations to achieve both robustness by design and robustness by adaptation.

Engineering of robust robotic systems is tackled by providing means to cope with the unavoidable deviation between design-time and run-time optimal solutions based on exploiting the power of software models. Designers will use offline simulation and analysis tools to validate the robot control system configurations, while the robot control system will use online simulation to manage adaptation to changes in its internals or in its operational context.

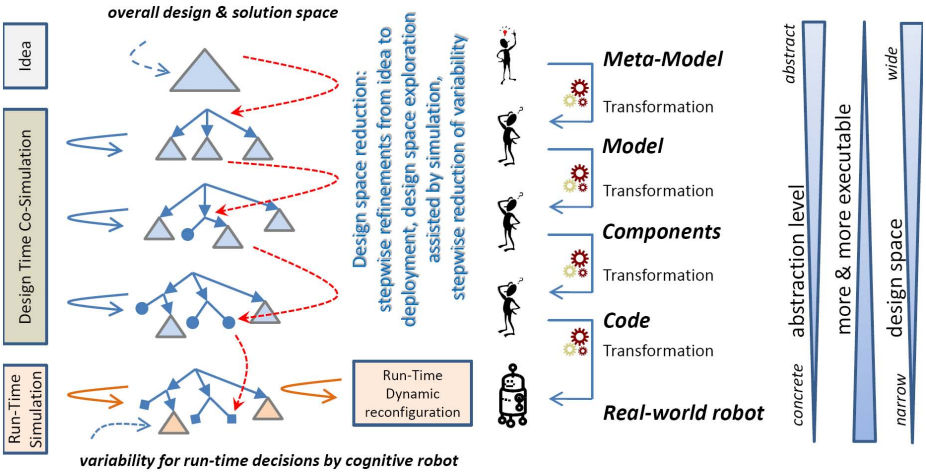

Fig. 1. A novel workflow seamlessly bridging design-time and run-time model-usage for exploiting purposefully left open variation points for run-time decisions

Simulation to support design decisions (model-driven design) or to generate embedded software (model-driven implementation) is considered as a way to master the complexity in system design. Simulation can be done at the intermediate steps of the model transformation and the development process for better predictability and explication. *Offline simulation* supports robustness by design. In this approach, simulation is also a key approach to check and validate alternative robot control system configurations, operational conditions and parameter settings at run-time. Critical situations can be checked online, but before the robot would have to deal with them in real world. This allows to select the most appropriate next action based on simulation of various options and not based on complex reasoning. Run-time monitoring of the selected configuration by simulation-based predictions of expectations can support safety issues and fault detection. Hence, *online simulation* supports robustness by adaptation.

Dynamic reconfiguration: The proposed approach reflects the current trend in the development of self-adaptive systems [1], which is to support increasing automation of decision making with respect to adaptation of systems by exploiting software models not only at design time but also at run-time [2]. These models provide a high-level basis for reasoning efficiently about variability aspects of a system, offer enough details to reflect the run-time variation of its internal and external behavior, and enable its design specifications to evolve while the system is running. Engineers can develop dynamically adaptive systems by defining several variation points (resources, algorithms, control strategies, coordination policies, cognitive mechanisms and heuristics, etc.). Depending on the context, the system dynamically chooses suitable variants to realize those variation points. These variants provide better *Quality of Service (QoS)*, offer new services that did not make sense in the previous context, or discard some services

that are no longer useful [3]. The approach allows to integrate in a model-driven environment mechanisms to monitor and reason about control system configurations at run-time, to validate alternative control system configurations with online simulation tools, and to automatically reconfigure the control system.

Non-functional properties are considered being mandatory for embodied and deployed robots in real-world. Their relevance arises from at least two observations of real-world operation: (i) some components have to guarantee *QoS* (e.g. adequate response times to ensure collision avoidance), (ii) the huge number of different behaviors needed to handle real world contingencies (behaviors cannot all run at the same time and one depends on situation and context aware configuration management and resource assignment). For example, if the current processor load does not allow to run the navigation component at the highest quality level, the component should switch to a lower quality level, resulting in a reduced navigation velocity, but still ensuring safe navigation. Instead of allocating all resources statically, the system needs to be able to adapt itself dynamically thereby taking into account appropriate *QoS* parameters and resource information.

So far, fundamental properties of robotic systems required for system level integration are typically not being made detailed enough nor explicit and they are not yet addressed in a systematic way in most robotics software development processes. Thus, these properties cannot be taken into account neither during design, development and system deployment (e.g. composability out of *COTS* components to foster reuse, ensure maturity and robustness) nor at run-time to support dynamically changing run-time configurations.

3 State-of-the-Art and Related Work

Robotics is a science of integration rather than a fundamental science, and the integration becomes ever more complex [4]. It is far too easy to say *details can be worked out* and it is often comfortable to focus on one aspect in isolation. Robots are complex systems that depend on systematic engineering. Currently, there are many tools that enable to separately design, test and deploy every single aspect of a robot (mechanical parts [5,6], control and algorithmic [7,8,9,10,11], software architecture [12,13,14], etc.). Typically, the different aspects of a robot are separately developed (following a separate process supported by a separate tool), and manually integrated afterwards. This leads to solutions very difficult (when not impossible) to evolve, maintain, and reuse. Systematic engineering processes are not applied at system level.

Software development in robotics is too complex and too expensive because (i) there is too little reuse, (ii) technology changes faster than developers can learn, (iii) knowledge and practices are hardly captured explicitly and made available for reuse, (iv) domain experts cannot understand all the technology stuff involved in software development.

In recent years, the Model-Driven Engineering (MDE) paradigm [15,16,17,18], has proven to mitigate many of the aforementioned limitations, in domains where systems are also highly complex, need a multi-disciplinary development

approach, and require high reliability and robustness, such as: embedded systems [19,20,21,22,23], automotive [24] and home automation [25,26]. *Artist2* [22] addresses highly relevant topics concerning realtime components and realtime execution platforms. *AUTOSAR* [24] comprises software components and their model-driven design. The ongoing *RT-Describe* project [27] addresses resource aspects and run-time adaptation. The *OMG MARTE* [28] activity provides a standard for modeling and analysis of realtime and embedded systems. They provide a huge number of *non-functional properties* to describe both, the internals and externals of a system. All these approaches can be reused for robotics but need to be extended towards model usage at run-time to address the deviation between design time and run-time optimality inherent in robotics. The robotics domain is far more challenging than many *DRE* systems in e.g. automotive or avionics due to the context and situation dependent dynamic reconfigurations of interactions and prioritized resource assignments at run-time.

The most advanced approach of Models@Run.Time [29] is the ongoing *DiVA* project [30] showing promising results in adapting business system architectures. However, it is not obvious how these achievements can be tailored to robotics where not only the system architecture but also the component level needs to be accessible for run-time adaptivity.

Tremendous code-bases (robotic frameworks, tools, libraries, middleware systems etc.) [31,32,33,34,35,36] coexist without any chance of interoperability and each tool has attributes that favors its use. Interestingly enough, new middleware systems are still introduced in robotics (like *ROS* [33]) although advanced standards like *DDS* [36] already provide publish-subscribe mechanisms even with *QoS* support. The robotics community still does not make the step towards *MDSD* that is define meta-models as a meaningful representation for robotic systems to overcome the tight coupling between robotic concepts and implementation technologies.

An introduction into robotics component-based software engineering (*CBSE*) can be found in [37,38]. The BRICS project [39] specifically aims at exploiting *MDE* as enabling approach to reducing the development effort of engineering robotic systems by making best practice robotic solutions more easily reusable. The *OMG Robotics Domain Task Force* [40] develops a standard to integrate robotic systems out of modular components. The *3-View Component Meta-Model* V^3CMM [41] comprises three complementary views (*structural view*, *coordination view* and *algorithmic view*) to model component-based robotics systems. *GenoM3* [42] proposes scripting mechanisms to bind component skeleton templates. These ongoing activities are covering first steps towards MDE in robotics. However they do not yet make the step towards seamless migration from design time to run-time model usage as proposed here.

4 The SmartSoft MDSD Toolchain

The SmartSoft project [43,44,45] has developed an open source toolchain based on the *Eclipse Modeling Project* [46] that supports model-driven development and integration of robotic software components on a variety of middleware

infrastructures. Its current state fits well into the proposed overall development process although it does not yet exhaustively model a robotics system with all views. However, there is already a huge benefit in even partially managing and explicating *non-functional properties*, e.g. for design-time realtime schedulability analysis, and in supporting the dynamic wiring pattern for run-time dynamic reconfiguration.

The instantiation of the novel work flow by SmartSoft concepts is illustrated in fig. 2. The development process starts with an idea. The model is enriched during development time until it finally gets executable in form of deployed software components. *First*, the system is described in a model-based representation (*platform independent model - PIM*) which is independent of the underlying framework, middleware structures, operating systems and programming languages. In this design phase, several system properties are either unknown or only known as requirements.

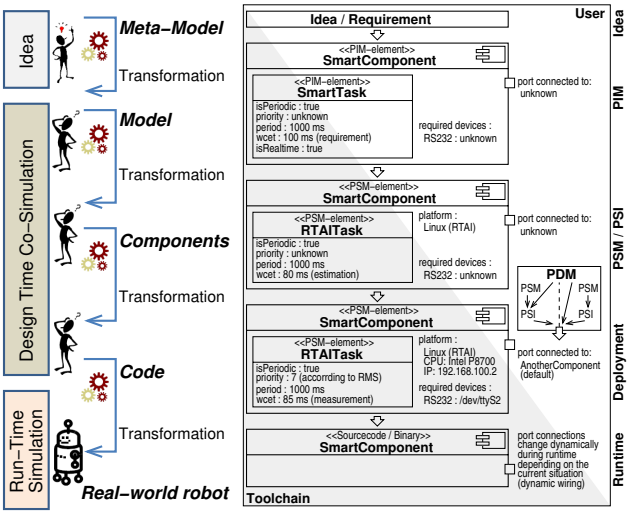

Fig. 2. The development process at-a-glance

Second, after successfully checking the *PIM*, it is transformed into a *platform specific model (PSM)* that provides bindings for platform specific refinements. Some characteristics can still be unknown and are added not until the deployment of the component. Finally, the *PSM* is transformed into the platform specific implementation (*PSI*).

Third, components are deployed to the target computers and robots. Their capabilities and characteristics are defined by the *platform description model (PDM)* which provides further bindings for so far left open model elements. Further model checkings are performed to verify constraints attached to the components against the capabilities of the system (e.g. is executable only on a certain type of hardware, needs one serial port, fulfills realtime guarantees etc.). *Fourth*, the system is run according to the deployment model. Certain properties can still be unknown and will be reasoned during run-time resulting in finally complete bindings of variation points. For example, the set of activated components, the configuration of the components and the wiring between the components can change during run-time depending on situation and context.

4.1 Development of Components Using SmartMARS

SmartMARS can be seen as both, an abstract meta-model as well as the current implementation as a UML profile. The basic concept behind the meta-model [44] are loosely coupled components offering/requiring services. Services are based on strictly enforced interaction patterns [45] providing precisely defined semantics. They decouple the sphere of influence and thus partition the overall complexity of the system since component internal characteristics can never span across components.

The major steps to develop a SmartSoft component are depicted in fig. 3. The component developer focuses on modeling the component hull, which comprises for example service ports and tasks – without any implementation details in mind. The generated component hull provides both, a stable interface towards the user-code (inner view of component) and a stable interface towards the other components (outer view of component). Due to the stable interface towards the user-code, algorithms and libraries can be integrated independent of any middleware structures. The integration is assisted and guided by the toolchain. Fig. 4 illustrates how the glue logic looks like to link existing libraries (or code

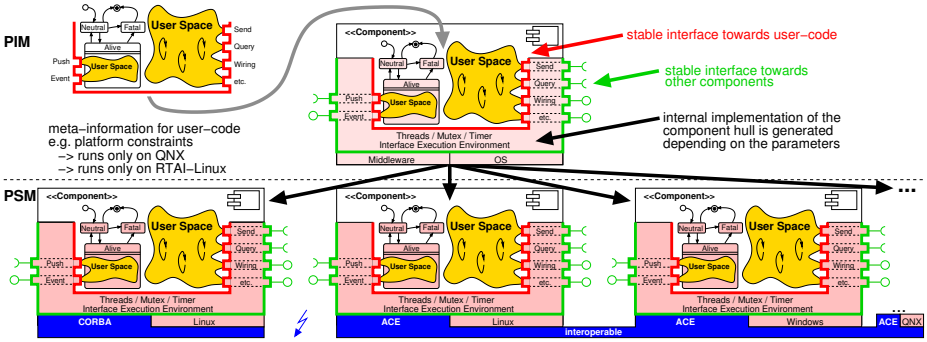

Fig. 3. Refinement steps of component development

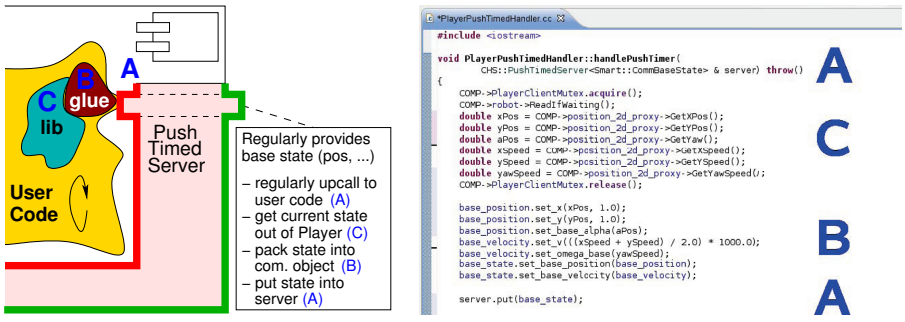

Fig. 4. Glueing user-code to service ports: example of a service port (based on a push-timed pattern) to regularly provide the base state (pose, velocity)

generated by other tools) to the generated component hull. The generation gap pattern [47] protects the user-code from modifications by the code generator. Tags on the component are used to indicate user code constraints (e.g. runs only on RTAI-Linux) and need to be set in the model by the user. The stable interface towards the other components ensures replacement and rearrangement of components.

4.2 Mapping of Abstract Concepts to Specific Technologies

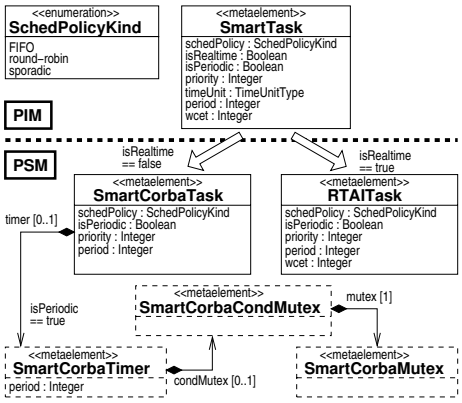

Fig. 5. *PIM* to *PSM* transformation

Fig. 5 details the transformation of the *PIM* and the *PSM* by the example of the *SmartTask* meta-element (illustrated using the *CORBA* based *PSM*, but the concepts are generic to any *PSM*). The *SmartTask* (*PIM*) comprises several parameters which are necessary to describe the task behavior and its characteristics independent of the implementation. In case the attribute `isRealtime` is set to `true`, the *SmartTask* is mapped onto a *RTAITask*. For the *RTAITask* the `wcet` and the `period` needs to be set according to the platform as described by the *PDM*. These parameters can be passed to appropriate tools like *CHEDDAR* [48] to perfom schedulability analysis. In case the target platform does not offer hard realtime capabilities, the toolchain reports this missing property. In case the attribute `isRealtime` is set to `false`, the *SmartTask* is mapped onto a non-realtime *SmartCorbaTask*. In case, the attribute `isPeriodic` is set to `true`, the toolchain extends the *PSM* by elements needed to emulate periodic tasks. The interface (inner view) of the *SmartTask* towards the user-code is stable independent of the internal mechanisms selected by the MDE toolchain.

4.3 Realtime Guarantees / Non-Functional Properties / Simulation

Selected views of the overall software model can be transformed into specific representations for evaluation by external tools (e.g. *CHEDDAR*). For example, schedulability analysis is required for hard real-time tasks and one has to export their parameters and their processor assignment. The M2M transformation rules for converting the appropriate parts of the software model into the desired target format are specified in the toolchain. At design time, the schedulability analysis including the model transformation can be triggered by the robot engineer (offline simulation) as soon as the deployment information and target platform characteristics are added to the model (fig. 6). At run-time, the very same analysis can be triggered by the task sequencing component (online simulation). The

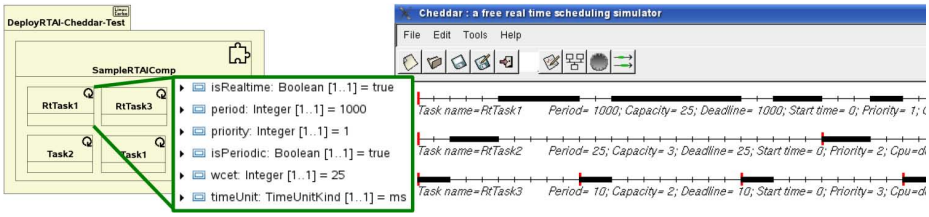

Fig. 6. *CHEDDAR* simulation based on the model parameters of the realtime tasks

task sequencer [49] comprises task nets to describe action plots. Task nets are expanded during run-time to executable behaviors. Prior to enabling a configuration, the selection of a certain configuration (e.g. set of activated components) can be based on the result of the schedulability analysis at run-time. How to invoke this analysis and its results (feasible or not) can be represented as behavior usable inside task nets. This already illustrates the advantages of explicating parameters in a modeling level for design-time and run-time simulation. It is one example of *QoS* and *resource awareness* in robotic systems.

4.4 Dynamic Wiring and Online Reconfiguration

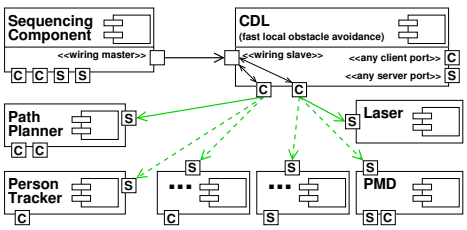

Fig. 7. Online dataflow adaptation

Fig. 7 shows the application of the wiring pattern [50] which supports run-time configuration of the communication paths between components. A *wiring master port* can be used to rearrange the client connections of components at run-time if these exhibit a *wiring slave port*. For example, the *CDL* component can receive its intermediate goals either from a *path planner* or a *person tracker* and its distance information from either the *laser* or the *PMD* component. *Dynamic wiring* is among others needed by the task sequencer for execution of task nets. The wiring pattern is used to compose the various components to different behaviors.

Dynamic wiring is also used to temporarily replace the robot component with the robot simulator component at run-time (Gazebo [32] with physics engine). This online simulation is used to determine at run-time the maximum allowed velocities taking into account the current payload of the robot. Prior to real-world execution, the variation point of the maximum allowed and save velocity is bound to a situation-dependent value. Again, this configuration is also represented as behavior (see above, same as for run-time schedulability analysis).

5 Real-World Application and Evaluation

The model-driven toolchain has been used to build numerous robotics components (navigation, manipulation, speech, person detection and recognition, simulators) reusing many already existing libraries within the component hulls (*OpenRAVE, Player/Stage, GMapping, MRPT, Loquendo, VeriLook, OpenCV*, etc.). These components are reused in different arrangements (fig. 8) to implement, for example, *Robocup@Home* scenarios (*Follow Me*, *Shopping Mall*, *Who-is-Who*).

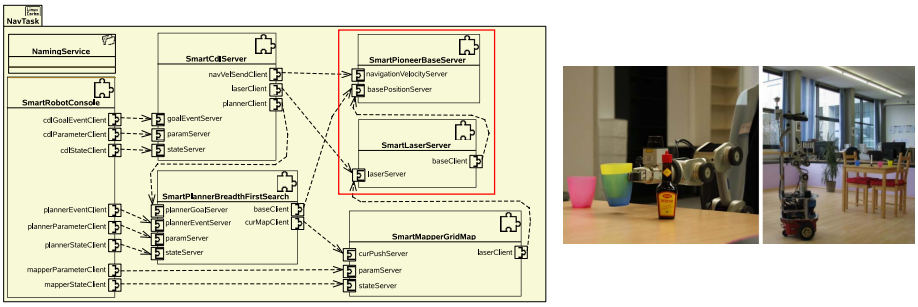

Fig. 8. Deployment diagram of a navigation task. Switching to simulation is done by replacing services (e.g. Stage/Gazebo component provides services of P3DX and LRF).

The model-driven approach, even in its current state, already proved to significantly reduce efforts of maintaining the various scenarios and composing new ones out of existing building blocks.

6 Conclusion and Future Work

We propose a novel design process for robotic system engineering which provides the ground for processes for the overall lifecycle of complex robotic systems (development, simulation, testing, deployment, maintenance). It allows to collaborate at the meta-models, transformations etc. and to compete at their implementation. It provides the freedom to choose whatever implementational technology is most adequate in a target domain. Furthermore, it provides the ground for a component market based on models, interoperability and services (e.g. value-adding expertise can be made available by custom-models). Most important, it provides a perspective of how to tackle *robustness by design* and *robustness by adaptation* by seamlessly bridging design-time and run-time model-usage for exploiting purposefully left open variation points for run-time decisions.

Future work will focus on further exploiting run-time dynamic reconfiguration and more extensive integration of run-time simulation coordinated by task-nets for which the current state of the model-driven design process, toolchain and component model (dynamic wiring, resource annotations) already proved to be a suitable starting point.

Acknowledgment

Work presented in sections 4 and 5 has been conducted within the *ZAFH Servicerobotik* (http://www.zafh-servicerobotik.de/), which receives funding from state of Baden-Württemberg and the European Union. Jan Broenink, Cristina Vicente-Chicote and Paul Valckenaers substantially contributed to the ideas detailed in sections 1 and 2.

References

1. Bencomo, N., Grace, P., Flores, C., Hughes, D., Blair, G.: Genie: Supporting the model driven development of reflective, component-based adaptive systems. In: ICSE, pp. 811–814 (2008)
2. Blair, G., Bencomo, N., France, R.B.: Models@run.time. IEEE Computer (2009)
3. Elkhodary, A., Malek, S., Esfahani, N.: On the Role of Features in Analyzing the Architecture of Self-Adaptive Software Systems. In: 4th Workshop on Models@run.time, MODELS 2009 (2009)
4. Bruyninckx, H.: Robotics Software: The Future Should Be Open. IEEE Robotics & Automation Magazine 15(1), 9–11 (2008)
5. ANSYS Simulation Driven Product Development, `http://www.ansys.com/`
6. Adams for Multibody Dynamics, `http://www.mscsoftware.com/Contents/Products/CAE-Tools/Adams.aspx`
7. Matlab/Simulink, `http://www.mathworks.com/`
8. 20-SIM, `http://www.20sim.com/`
9. SCILAB, `http://www.scilab.org/`
10. MODELICA, `http://www.modelica.org/`
11. Labview Robotics, `http://www.ni.com/Robotics`
12. SHE / POOSL, `http://www.es.ele.tue.nl/she`
13. The Vienna Development Method Portal, `http://www.vdmportal.org/`
14. Overture: Open-source Tools for Formal Modelling, `http://www.overturetool.org/`
15. OMG Model-Driven Architecture, `http://www.omg.org/mda/`
16. Völter, M., et al.: Model-Driven Software Development: Technology, Engineering, Management. Wiley, Chichester (2006)
17. Bézivin, J.: On the unification power of models. Journal of Systems and Software 4(2), 171–188 (2005)
18. Innovations in Systems and Software Engineering Special Issue Model based development methodologies, vol. 5(1). Springer, Heidelberg (2009)
19. MEDEIA: Model-driven embedded system design environment for the industrial automation sector, 7FP EU Project, `http://www.medeia.eu/`
20. SATURN: SysML bAsed modeling, architecTUre exploRation, simulation and syNthesis for complex embedded systems, 7FP EU Project, `http://www.saturn-fp7.eu/`
21. QUASIMODO: Quantitative System Properties in Model-Driven Design of Embedded Systems, 7FP EU Project, `http://www.quasimodo.aau.dk/`
22. ArtistDesign, European Network of Excellence on Embedded System Design, 7FP EU Project, `http://www.artist-embedded.org/`
23. Truscan, D., Lundkvist, T., Alanen, M., Sandström, K., Porres, I., Lilius, J.: MDE for SoC design. Innovations Syst. Softw. Eng. 5, 49–64 (2009)

24. AUTOSAR: AUtomotive Open System ARchitecture, http://www.autosar.org/
25. POBICOS: Platform for Opportunistic Behavior in Incompletely Specified, Heterogeneous Object Communities, 7FP EU Project, http://www.ict-pobicos.eu/
26. AMPLE: Aspect Oriented, Model Driven and Product Line Engineering, 6FP EU Project, http://www.ample-project.net/
27. RT-Describe, http://www.esk.fraunhofer.de/EN/press/pm0911RTDescribe.jsp
28. OMG MARTE, A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2, ptc/2008-06-08, Object Management Group (June 2008)
29. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer 42(10), 22–27 (2009)
30. Morin, B., Barais, O., Nain, G., Jezequel, J.-M.: Taming Dynamically Adaptive Systems using models and aspects. In: IEEE 31st International Conference on Software Engineering (ICSE), pp. 122–132 (2009)
31. Robot Standards and Reference Architectures (RoSTa), http://wiki.robot-standards.org/index.php/Middleware
32. Player/Stage/Gazebo, http://playerstage.sourceforge.net/
33. ROS: Robot Operating System, http://www.ros.org/
34. Microsoft: Microsoft Robotics Developer Studio, http://msdn.microsoft.com/en-us/robotics/default.aspx
35. OMG CORBA, http://www.corba.org
36. OMG DDS: Data Distribution Service for Real-time Systems (DDS) v1.2, http://www.omg.org/technology/documents/dds_spec_catalog.htm
37. Brugali, D., Scandurra, P.: Component-based Robotic Engineering. Part I: Reusable building blocks. IEEE Robotics and Automation Magazine (2009)
38. Brugali, D., Shakhimardanov, A.: Component-based Robotic Engineering. Part II: Models and systems. IEEE Robotics and Automation Magazine (March 2010)
39. BRICS: Best Practice in Robotics, http://www.best-of-robotics.org/
40. OMG Robotics: OMG Robotics Domain Task Force, http://robotics.omg.org/
41. Alonso, D., Vicente-Chicote, C., Ortiz, F., Pastor, J., Álvarez, B.: V^{3}CMM: a 3-View Component Meta-Model for Model-Driven Robotic Software Development. Journal of Software Engineering for Robotics 1(1), 3–17 (2010)
42. Mallet, A., Pasteur, C., Herrb, M., Lemaignan, S., Ingrand, F.: GenoM3: Building middleware-independent robotic components. In: Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), Anchorage (2010)
43. SmartSoft, http://smart-robotics.sf.net/
44. Schlegel, C., Haßler, T., Lotz, A., Steck, A.: Robotic Software Systems: From Code-Driven to Model-Driven Designs. In: Proc. 14th Int. Conf. on Advanced Robotics (ICAR), Munich, Germany (2009)
45. Schlegel, C.: Communication Patterns as Key Towards Component Interoperability. In: Software Engineering for Experimental Robotics. STAR Series, vol. 30, pp. 183–210. Springer, Heidelberg (2007)
46. Eclipse Modeling Project, http://www.eclipse.org/modeling/
47. Vlissides, J.: Pattern Hatching - Generation Gap Pattern (1996), http://researchweb.watson.ibm.com/designpatterns/pubs/gg.html
48. The Cheddar project, http://beru.univ-brest.fr/~singhoff/cheddar/
49. Schlegel, C., Illmann, J., Jaberg, H., Schuster, M., Wörz, R.: Integrating Vision-Based Behaviors with an Autonomous Robot. VIDERE - Journal of Computer Vision Research, 32–60 (2000) ISSN 1089-2788
50. Schlegel, C.: Communication Patterns as Key Towards Component-Based Robotics. Int. Journal of Advanced Robotic Systems 3(1), 49–54 (2006)