Fault Avoidance in Development of Robot Motion-Control Software by Modeling the Computation*

Yury Brodskiy, Robert Wilterdink, Stefano Stramigioli, and Jan Broenink

Robotics and Mechatronics, Faculty EEMCS,
University of Twente, The Netherlands
y.brodskiy@me.com,
J.F.Broenink@utwente.nl

Abstract. In this article, we present the process of modeling control algorithms as means to increase reliability of software components. The approach to developing Embedded Control Software (ECS) is tailored to Component-Based Software Development (CBSD). Such tailoring allows to re-use the ECS development process tools in a development process for robotics software. Model-to-text transformation of the ECS design tool is extended to model-to-component transformation suitable for CBSD frameworks. The development process and tools are demonstrated by a use case.

1 Introduction

The quest for safety and autonomy of a robot is extremely complex, and strongly connected to concepts of reliability. Robots are designed to perform a variety of tasks in diversified conditions. However, development of a robotic application is a complex and error-prone process which requires integration of results from many engineering domains (software, mechanical, electrical and control engineering). This is especially important in development of robot motion-control software, as this part of the system is critical for reliability of robots. Thus, it is crucial that fault avoidance techniques are exercised to a full possible extent during development of the software.

Current research in software development for robots is focused on Component-Based Software Development (CBSD) [18,19]. CBSD supports the development and reuse of large-grained pieces of robotics software through component-based software frameworks such as ROS [39], Orocos [20,34], SmartSoft [36,37]. To gain the advantages provided by a component-based software framework, the software has to be structured into independent components. This need for structuring of software has triggered research on the application of Model-Driven Engineering (MDE) techniques to CBSD.

* The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics).

D. Brugali et al. (Eds.): SIMPAR 2014, LNAI 8810, pp. 158–169, 2014.

In CBSD, models are used to design and analyze the software architecture [38]. Following the concept of separation of concerns as presented in [21], a software architecture can be described by models formulated in terms of Computation, Composition, Connection [26], Configuration [18] and Coordination [27], referred to as 5C. Composition and Connection indicate the software architecture of the system. Configuration and Coordination describe parametric and discrete states of the system. Computation stands for the actual algorithm implemented in the component, for example of a control law such as a PID controller. Modeling the Computation is not part of the existing CBSD practice.

To make sure that modeling the Computation *becomes* a part of a well-defined development process, we combine two relevant development processes – The Robotic Application development Process (RAP) [29] and the Embedded Control Software (ECS) design trajectory [15,14]. RAP has been proposed by the BRICS project as a result of simultaneously tailoring of over 25 different development processes to the robotics domain and CBSD. The ECS design trajectory represents practices in development of software for embedded applications and explicitly advocates modeling of the algorithm of a software component (controller) [14]. Both development processes use modeling of software to achieve a higher quality product. Nevertheless, these development processes have different perspectives on software modeling. We argue that combining these models results in a uniform description of the software on the modeling level, thus allowing to gain the benefits from both development processes.

Throughout this article, the combination of these two development processes is discussed. Relevant terms used in MDE in application to CBSD are adopted from [21,10]. The dependability related terms are adopted from [1]. The motivation for modeling the Computation to gain software reliability improvement is presented in Section 2. The combination of RAP and ECS design trajectory is discussed in Section 3. The integration of tools used in RAP and the ECS design trajectory is presented in Section 4. The resulting tool-chain is demonstrated by a use case presented in Section 5.

2 Modeling the Computation to Increase Software Dependability

Modeling the Computation is describing the actual algorithm to be implemented. The result of Modeling the Computation is a *computation model*. It represents the mathematical nature of the algorithm. Such model leaves out platform, operating system, framework or programming-language specific elements. A computation model can be constructed based on several different meta-models: transfer functions, logic circuits, Bayesian networks, neural networks, bond graphs.

A computational model serves three purposes:

- A model improves understanding of functioning of the algorithm, thus revealing the points where robustness should be improved.
- A model, supported by simulation tools, can be used to study the behavior of the algorithm, in an attempt to verify its qualities.

– A model can be automatically transformed to a formulation needed for the next step in the development process.

These purposes overlap with means of improving a system's reliability, typical for a development process: fault prevention, fault removal and fault forecasting (Figure 1).

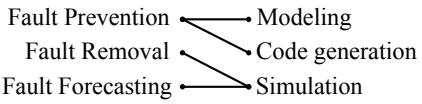

Fig. 1. Mapping of reliability means on purposes of a Computation model

A developer models the Computation to obtain a simplified but competent representation of the algorithm (the model of algorithm). This is achieved by choosing the modeling language that supports an effective and simple representation of the algorithm that is being developed, i.e. the chosen modeling language is more expressive than a general-purpose implementation language. The general-purpose language might obscure the design intent due to a vast number of implementation details, while a model allows to focus on design of the algorithm avoiding implementation issues [38]. The model of algorithm provides a clear expression of the design intent and thus prevents logic faults. It especially concerns the class of development, human-made, deliberate, nonmalicious faults [1] as the trade-offs made at development time are made explicit.

Faults (class of development, human-made, non deliberate, nonmalicious [1]) can be prevented by modeling the Computation with appropriate tool support. The composition rules, defined by the meta-model, can be used to prevent illegal constructs that would result in faults, which are typical unintended actions done by mistake. This enforcement of composition rules on model level also prevents faults occurring due to misunderstanding between collaborating teams [13], which also classify as non-deliberate, human-made faults. Furthermore, appropriate tool support allows to automate the transformation of models to formulations needed for next steps in the development process. This allows reducing significantly the number of human-made faults as the developer is excluded form the transformation process.

Simulation of the algorithm, one of the purposes of modeling the Computation, supports both fault removal and fault forecasting means (Figure 1). This fault removal process is similar to the approach of iteratively refining models, used in the ECS design trajectory [15,14]. In simulation, a designer has full control over the system states, outputs and inputs, thus any situation or behavior can be tested, and the response can be verified. An example of such use of simulations is fault modeling and fault injection [13]. This approach prescribes building a model of faulty behavior in addition to modeling expected behavior. This approach can be used to evaluate performance of the proposed fault-tolerant control. Overall, use of simulateable models simplifies and increases the quality of the fault removal and fault forecasting processes.

3 Modeling the Computation in the Development Process

Modeling of software in a development process can be done from different perspectives (5C as described in [21]), moreover modeling software from each perspective leads to improvement in software quality. Thus by modeling software from all perspectives allows to maximize the benefits from using models in development process. A development process has to indicate at which step each model has to be developed. Furthermore, the steps at which the models are being synchronized and transformed into implementation have to be indicated such that inconsistencies in models are avoided.

The development processes noted above (Section 1) use modeling of software as one of the main activities. Nevertheless, these development processes have different perspectives on software modeling. The goal of combining these development processes is to indicate steps of synchronization and transformation of models into implementation. That would enable modeling software from all perspectives, and development of appropriate tool support as demonstrated later.

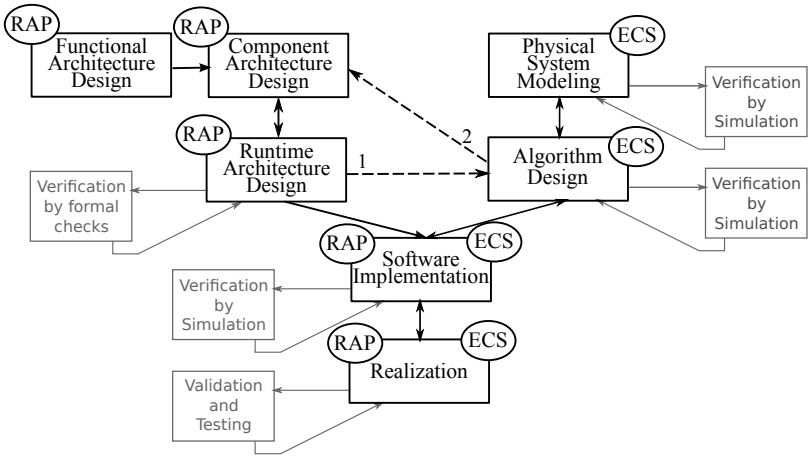

Fig. 2. The proposed ECS trajectory combined with architectural elements form RAP

The updated ECS trajectory (Figure 2) combines the modeling procedures of the ECS trajectory [16] and RAP [29]. Development steps inherited from RAP, indicated with RAP tag, represent the process of modeling the software architecture. Development steps inherited from ECS, indicated with ECS tag, represent the process of modeling the algorithm of the application. Steps, indicated by both tags, are prescribed by both RAP and ECS, and require information about the algorithm and the software architecture. By combining modeling processes of RAP and ECS the information about algorithm and software architecture is captured inform of models before the *Software Implementation* step. These

models are complimentary to each other as they contain different type of information about the software system. The combination of these models represents a uniform model of a software component, which covers all 5 aspects of a software system (5C [21]).

To maintain synchronization between models from RAP and ECS two additional interlinks (the dashed lines nr. 1 and nr. 2 (Figure 2)) between architecture design and algorithm design are added. These interlinks indicate information flow during modeling and emphasize the iterative style of model development.

Line nr. 1 (Figure 2) indicates that the constraints imposed by the runtime architecture have to be taken into account during Algorithm design. The analysis of the effects of the run-time architecture on the algorithm performance allows to forecast possible failures and develop algorithms that tolerate these.

Line nr. 2 (Figure 2) indicates that a detailed knowledge about algorithm functioning is required for building the efficient component architecture. For example, a component architecture built without that knowledge might result in unreasonable communication requirements (such extreme band-with or unachievable small latency). Inclusion of component architecture into Algorithm Design step allows to identify these issues at the modeling stage and can lead to restructuring the component architecture.

At the *Software Implementation* step, the models of the algorithm (Computation or Coordination aspects developed in the *Algorithm Design* step) are combined with the model of software architecture (Configuration, Connection, Composition aspects developed in the *Runtime Architecture Design* step). The automation of this *Software Implementation* step is discussed in Section 4.

4 Tool Integration

The goal of the tool integration is, as indicated by the design trajectory (Figure 2), to facilitate combining the models of a component resulted from RAP and ECS steps. Moreover, if these models contain sufficient information about all of the 5 aspects of a software system (5C), the *Software Implementation* step can be automated, such that the models are transformed into forms/artifacts necessary in the consecutive steps of the development process.

Table 1. Tools and their focus in modeling

Configuration & Connection & Composition	Computation & Coordination
BRIDE[9]	MatLAB[30]
SmartSoft[36]	**20-sim[23]**
OpenRTM[33]	LabView[31]
Proteus[35]	OpenModelica[32]
TERRA[3]	Dymola[24]

The development process (Figure 2) can be used with a number of tools that provide similar functionality such as presented in Table 1.

To exemplify the tool integration, the tools BRIDE and 20-sim are used as suggested by the authors of RAP [29] and the ECS design trajectory [15].

The BRIDE toolchain [9] has been developed in the project BRICS to support RAP. It facilitates the design of components and component compositions for robotic application deployment. It offers graphical editors to model components using the BRICS Component Model (BCM) [21], with model-to-model transformations to the software frameworks Orocos RTT [34,22] and ROS [39].

20-sim was developed for the design of mechatronic systems and supports the ECS design trajectory as development methodology. It embodies the concept of concurrent design of mechanical, electrical and control parts of the system. 20-sim provides modeling primitives for designing the Computation. It can be used for both modeling of control algorithms and physical systems behaviour. 20-sim allows a graphical approach to hierarchical structuring of algorithms.

4.1 CBSD- and Computation-Model Integration

The transformation of any Computation meta-model into a component meta-model used by the CBSD tools (like BRIDE) results in a model reduction since the concepts of computation are not supported by such tools. To preserve the computation elements a second artifact is generated – the code (Figure 3). The combination of the generated code and the model completely represent a single component which can be used in the next phase of the development process, i.e. *Realization*.

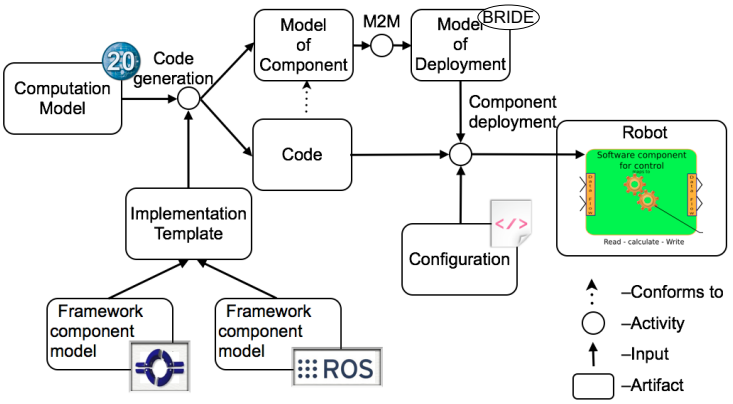

Fig. 3. Model-to-component transformation

The integration between BRIDE and 20-sim is structured as a 2-step process [12]. First 20-sim transforms the Computation model into a code and a model of the component (Figure 3). Second, the model of the component is analysed and, by using predefined rules, a BRIDE model of the component is generated. The newly generated model contains information about the Communication and Configuration (component interface). The Coordination and Computation of the component are only encoded in the component implementation as BRIDE does not offer primitives to describe these concepts.

5 Use Case Application

5.1 BRICkS Stacking Application

The use case is motion control for the KUKA youBot [4,11] mobile manipulator. The resulting set of software components is termed as *motion stack*. These are implemented using the methodology described above. Orocos RTT was used as target component-based software framework.

Based on Figure 2 some of the steps can be performed in parallel. The Physical System Modeling step can be preformed in parallel to the Functional Architecture Design step. The Component Architecture Design, Run-time Architecture Design and Algorithm Design steps have circular dependency, and therefore require an iterative approach. The Software Implementation and Realization steps are preformed sequentially.

The Physical System Modeling step is presented in detail in [25], and a detailed description of the algorithms is presented in [12,10]. The focus of this paper is the architectural design (Section 5.2), the effect of architecture on algorithm (Section 5.3) and automated transformation of the models into components (Section 5.4).

5.2 Architecture Design

The architecture design is done in three steps, each increase the level of detail at which the system is described.

A generic *functional architecture* of a motion stack [12] is presented in Figure 4. The proposed organization of the software specifies the granularity of the algorithms and provides a standard functional decomposition into components. This standardisation of component functions enables further harmonization of the component interfaces, which is a requirement for introducing variation points [17]. As a result, an exchange of a component for one with a different behavior should not require modifying the motion control structure nor any component internals.

Figure 5 depicts the *component-level architecture* of the motion stack in the form of a data-flow diagram, based on its functional decomposition. The presented architecture is an example of a variability solution.

A *run-time architecture* is required to define how components are executed with their time and concurrency relations. A run-time architecture can introduce various effects that will affect the algorithm performance. For example, message passing between components can introduce time delays, or message losses. To verify the algorithm's robustness and tolerance of such effects, the robot model has to include them. Time delay elements (z^{-1}) indicate possible time delays in the signal exchange, that has been used in algorithm verification.

5.3 Algorithm Design

Algorithm Design (Modeling the Computation) goes in parallel with the design of the architecture. The effects that are introduced by the architecture are used to model and identify unexpected behaviours. Knowledge about the algorithm

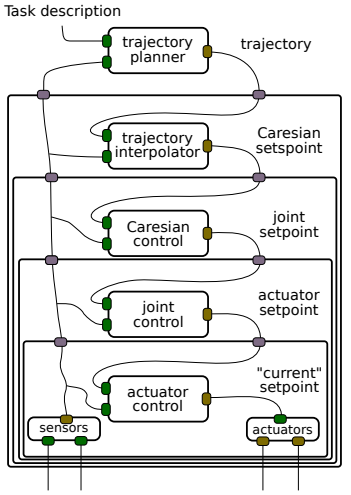

Fig. 4. Conceptual architecture

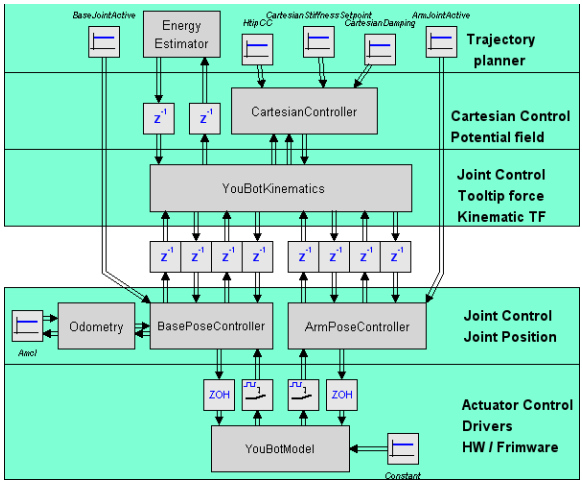

Fig. 5. Architecture of implementation

performance is used to modify the component architecture and generate deployment constraints.

The algorithm design requires a competent model of the system that will be controlled. In the use case, a model developed in [25] was used to verify performance and robustness of the algorithms in a simulation.

The data-flow diagram, shown in Figure 5, has resulted from the architecture design route. Each block of this diagram has to become a software component in the real system and has been filled in with required algorithm according to its task, details of the algorithms are reported in [10].

When the responses of the system meet the task requirements, such model is ready for the Software Implementation step.

5.4 Software Implementation

The parts of the motion stack were implemented as components in the Orocos RTT framework. The data-flow diagram (Figure 5) illustrates the component-level architecture of the motion stack, where blocks in the diagram stand for software components, except z^{-1} which model the effects of communication (latency).

The block at the firmware and drivers level (denoted YouBotModel in Figure 5) implements the communication to the firmware of the youBot actuators and sensors. This component contains operating-system specific and hardware-specific code, and is written in a general purpose language (C++).

The blocks at the "trajectory planner" level of the given application are the Coordination-type components. The Coordination-type components are most efficiently expressed in a FSM, therefore, a DSL for a FSM was used to design the actual trajectory planner components [28]. The existing life-cicle state machine

of an Orocos-RTT component has been extended to include states required for the application. The Coordination was modeled using rFSM [28], a textual modeling language for FSM. The component Configuration and Communication were modeled using BRIDE. The model designed with rFSM is part of the working system, it is executed using a run-time interpreter based on Lua.

Other blocks in the data-flow diagram (Figure 5) are 'pure' Computation components. These components are implemented using the model-to-component transformation, as described in Section 4.

The generated component is imported into BRIDE, where Communication and Configuration perspectives can be further refined to connect the generated component into the system. Detailed instructions on using code generation process are presented by [12].

A composition model of the application is finalized using BRIDE. The models of computation are transformed into component models usable by BRIDE. These models are combined with the models components that were developed in BRIDE to obtain composition model.

6 Testing the Methodology

The methodology and the youBot motion stack software, resulting from applying our method, have been tested for reusability and reliability, in three different occasions, namely three BRICS events.

In the first event, the BRICS research Camp 3 [6], the youBot motion stack was used to test re-usability of the components developed using the proposed methodology. The software was provided to students, research camp participants, with only brief explanation of the algorithmic aspects of the system; the Computation and Coordination model were presented. The students have successfully used concepts of modeling of Computation and Coordination to adapt the provided system to their tasks, which demonstrates ease of re-use of the developed components. Two groups of 4 students were using the proposed methodology to modify the provided youBot motion stack using our tool-chain. Moreover, one of the groups modified their own development tools according to our methodology to develop a sequence control for solving their exercises.

In the second event, Automatica 2012 [7], the youBot motion stack was used to test reliability of the components with respect to inter-component communication faults. The algorithms were shown to work with communication over a congested WiFi [12]. The software was subjected to prolonged active use, with frequent interruptions for inspection and parametric changes, which were done for demonstration purposes. The setup was performing flawlessly for the whole week of the trade fair (5 days 8 hours a day). Comparable research software is rarely capable of withstanding such stress testing on first-time use.

At the third event, BRICS research Camp 5 [8] re-usability and reliability of the components were further tested. The software has been given for modification to the BRSU RoboCup@work team, after a brief introduction similar to Research camp 3 student teams. The RoboCup team has successfully modified parts of

the motion stack and uses this software for robot control during completions. This also confirms that components developed using the proposed methodology can be easily re-use for different applications with high level of reliability.

The software integration developed in Section 4 is available as open source [40,41] and is being used in other projects of our lab such as for example [5].

7 Conclusions

The combination of both approaches, the CBSD method (RAP) advocated by BRICS and the ECS design trajectory, results in uniform coverage of modeling perspectives (5C) for a software component, and thus results in more reliable robotic components and applications.

The proposed approach contributes to software quality improvement as follows:

- Automated model-to-code transformations reduce faults during implementation of the algorithm by excluding human factor from the process.
- Modeling enables the designer to focus on the chosen aspects of the system (*e.g.* algorithm or architecture), instead of implementation details, and thereby increasing quality of the resulted software.
- Simulation of the algorithm allows to examine hypothetical/dangerous situations using fault modeling techniques, such as sensor failures, which can be used for development of fault tolerance algorithms.

The methodology tests have shown that algorithms verified in simulation have a high success rate on a real setups, which confirms results reported by [14].

To achieve the uniform coverage of modeling perspectives, two different modeling tools had to be used. Each tool focuses on a different engineering role. Two tools were used as an example of the approach, BRIDE and 20-Sim. BRIDE is used to model architecture of the system, and 20-sim is used to model the algorithm.

The presented model-based tool integration shows that tools can easily be combined on the component level. A model-to-model transformation is used to export the component interface of a 20-Sim sub-model to an Orocos-RTT component model. The Orocos-RTT component model is used in BRIDE to design the deployment of a robotic application. The advantage of directly generating the executable component from 20-Sim is that the target component meta-model is not required to support an equivalent Computation meta-model because the algorithm code is directly generated for the target framework. The software integration developed in Section 4 is available as open source code [40,41] and is being used in other projects such as for example [5]. The integration approach is generic such that other tools can be integrated in a similar way.

Future work is further developed the tool-chain getting beyond prototype stage.

The developed tool chain requires creation of separated components for each modeling language being used. The template of Generic Architecture Component

(GAC) presented by [2] demonstrates the need to combine Coordination and Computation primitives in a single component. The tool-chain can be modified to accommodate that requirement.

References

1. Aviezienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Trans. on Dependable and Secure Computing 1(1), 11–33 (2004)
2. Bezemer, M.M.: Cyber-Physical Systems Software Development. Ph.D. thesis, University of Twente (2013)
3. Bezemer, M.M., Wilterdink, R.J.W., Broenink, J.F.: CSP-Capable Execution Framework. Communicating Process Architectures 68, 157–175 (2011)
4. Bischoff, R., Huggenberger, U., Prassler, E.: KUKA youBot - a mobile manipulator for research and education. In: Proc IEEE Int'l Conf on Robotics and Automation, pp. 1–4 (May 2011)
5. de Boer, H.: Modeling and Control of the Philips Robot Arm. Msc thesis, University of Twente (2012)
6. BRICS: BRICS Research camp 3 (2011), `http://www.best-of-robotics.org/3rd_researchcamp/MainPage`
7. BRICS: BRICS - European Research Project - demonstration booth. 5th International Trade Fair for Automation and Mechatronics (May 2012)
8. BRICS: BRICS Research camp 5 (2012), `http://www.best-of-robotics.org/5th_researchcamp/MainPage`
9. BRICS: BRIDE - the BRIcs Development Environment (January 2013), `http://www.best-of-robotics.org/bride`
10. Brodskiy, Y.: Robust autonomy for interactive robots. University of Twente, Enschede (2014)
11. Brodskiy, Y., Dresscher, D., Stramigioli, S., Broenink, J.F., Yalcin, C.: Design principles, implementation guidelines, evaluation criteria, and use case implementation for robust autonomy. Tech. Rep. D61, The BRICS Project (nr 231940) (January 2011)
12. Brodskiy, Y., Wilterdink, R., Broenink, J.F., Stramigioli, S.: Collection of methods for achieving robust autonomy. Tech. rep. (2013)
13. Broenink, J.F., Fitzgerald, J.F., Gamble, C.J., Ingram, C., Mader, A.H., Marincic, J., Ni, Y., Pierce, K.G., Zhang, X.: D2.3 — Methodological Guidelines 3. Tech. rep., The DESTECS Project (CNECT-ICT-248134) (December 2012)
14. Broenink, J.F., Groothuis, M.A., Visser, P.M., Bezemer, M.M.: Model-Driven Robot-Software Design Using Template-Based Target Descriptions. In: ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications, pp. 73–77. IEEE (May 2010)
15. Broenink, J.F., Groothuis, M.A., Visser, P.M., Orlic, B.: A model-driven approach to embedded control system implementation. Control, 137–144 (January 2007)
16. Broenink, J.F., Ni, Y.: Model-driven robot-software design using integrated models and co-simulation. In: Int'l Conf. Embedded Computer Systems, pp. 339–344 (2012)
17. Brugali, D., Gherardi, L., Biziak, A., Luzzana, A., Zakharov, A.: A Reuse-Oriented Development Process for Component-Based Robotic Systems. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) SIMPAR 2012. LNCS, vol. 7628, pp. 361–374. Springer, Heidelberg (2012)

18. Brugali, D., Scandurra, P.: Component-based Robotic Engineering Part I: Reusable building blocks. Robotics Automation Mag. 16(4), 84–96 (2009)
19. Brugali, D., Shakhimardanov, A.: Component-Based Robotic Engineering (Part II): Systems and models. Robotics Automation Mag. 17(1), 100–112 (2010)
20. Bruyninckx, H.: Open Robot Control Software: the OROCOS project. In: Proc IEEE Int'l Conf on Robotics and Automation, pp. 2523–2528. IEEE (2001)
21. Bruyninckx, H., Hochgeschwender, N., Klotzbucher, M., Soetens, P., Kraetzschmar, G., Brugali, D., Garcia, H., Shakhimardanov, A., Paulus, J., Reckhaus, M., Gherardi, L., Faconti, D.: The BRICS Component Model: a Model-Based Development paradigm for complex robotics software systems. In: Proc of Annual ACM Symposium on Applied Computing, vol. 28, pp. 1758–1764. ACM (2013)
22. Bruyninckx, H., Soetens, P., Koninckx, B.: The real-time motion control core of the Orocos project. In: Proc IEEE Int'l Conf on Robotics and Automation, vol. 2, pp. 2766–2771. IEEE (September 2003)
23. Controllab Products, B.V.: 20-sim (2013), http://www.20sim.com
24. Dassault Systemes AB: Dymola (2013)
25. Dresscher, D., Brodskiy, Y., Breedveld, P., Broenink, J.F.: Modeling of the youBot in a serial link structure using twists and wrenches in a bond graph. In: Proc SIMPAR 2010 Workshop, Darmstadt, pp. 385–400 (2010)
26. Hochgeschwender, N., Gherardi, L., Shakhimardanov, A., Kraetzschmar, G., Brugali, D., Bruyninckx, H.: A Model-based Approach to Software Deployment in Robotics. In: Proc IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems. IEEE/RJS (November 2013)
27. Klotzbucher, M., Biggs, G., Bruyninckx, H.: Pure Coordination using the Coordinator – Configurator Pattern. CoRR abs/1303.0 (2013)
28. Klotzbucher, M., Bruyninckx, H.: A Lightweight Real-Time Executable Finite State Machine Model for Coordination in Robotic Systems. Tech. rep. (2007)
29. Kraetzschmar, G., Shakhimardanov, A., Paulus, J., Hochgeschwender, N., Reckhaus, M.: Deliverable D-2.2: Specifications of Architectures, Modules, Modularity, and Interfaces for the BROCRE Software Platform and Robot Control Architecture Workbench. Tech. rep., BRICS FP7 project deliverable (2010)
30. MathWorks: MatLAB (2013), http://www.mathworks.com
31. National Instruments: LabView (2013)
32. Open Source Open Modelica Consortium: OpenModelica (2013)
33. OpenRTM Project: OpenRTM Project (2013)
34. Orocos Project: Smarter control in robotics & automation (January 2013), http://www.orocos.org
35. Proteus Project: Proteus (2013)
36. Schlegel, C.: SmartSoft: Components and toolchain for robotics (January 2013), http://smart-robotics.sourceforge.net
37. Schlegel, C., Worz, R.: The software framework {SmartSoft} for implementing sensorimotor systems. In: Proc IEEE/RSJ Int'l Conf on Intelligent Robots and Systems, vol. 3, pp. 1610–1616 (1999)
38. Schmidt, D.C.: Model-driven engineering. Computer, 25–31 (2006)
39. Willow Garage: ROS project (January 2013)
40. Wilterdink, R.J.W., Brodskiy, Y., Broenink, J.F.: Eclipse 20-sim update site (February 2013), http://www.ce.utwente.nl/20sim/updates/
41. Wilterdink, R.J.W., Brodskiy, Y., Tadele, T.S., Broenink, J.F.: 20-Sim C-code generation templates and model-to-model transformations (2013), https://git.ce.utwente.nl/20sim