Formal Model-based Development in Industrial Automation with Reactive Blocks

Peter Herrmann[1] and Jan Olaf Blech[2]

[1] NTNU, Trondheim, Norway, herrmann@item.ntnu.no
[2] RMIT University, Melbourne, Australia, janolaf.blech@rmit.edu.au

Abstract. The use of standard IT equipment to control machines is becoming increasingly popular mostly due to lower costs. Further, trends and initiatives such as Industry 4.0 and smart factories accelerate the use of standard IT components by demanding interconnected controllers and factory equipment communicating with internet services. This development offers new possibilities to use existing software frameworks and software architectural approaches as well as development standards in industrial automation. The formal methods-based support that already exists for standard IT platforms can now be applied to industrial control devices as well. In this paper, we look into the application of our Reactive Blocks framework for industrial automation. Reactive Blocks comes with a well established semantics and verification approaches tied to it. We demonstrate the advantages of our methodology with an example.

1 Introduction

Industrial automation devices have traditionally been programmed by engineers using standards such as IEC 61131–3 [20] and its derivatives. We see, however, novel trends according to which this well established procedure will change in the near future. One trend is the recent convergence of PC hardware and Programmable Logic Controllers (PLC) with respect to software development. In the past, industrial automation devices mostly relied on techniques and standards that where developed independently from PC hardware and IT technologies. Examples include the IEC 61131 standard for PLC and PROFIBUS [2] on the network technology side. In recent years, some PLC vendors started to integrate standard PC processors. Moreover, smart single-board computers like the Raspberry Pi [38] came into the market which offer operating systems close to those of ordinary PCs. These boards are cheap but powerful enough to carry out control functions. For instance, we use Raspberry Pi-based devices to drive a bottling plant deployed in the RMIT's advanced manufacturing precinct [16]. On the network technology side, the Ethernet has gained entry into the world of industrial automation.

Another trend is the growing interconnectivity of controllers. PLCs are now communicating with each other and with other external devices and services, not just for synchronization and basic control via the Supervisory Control and Data

Acquisition (SCADA) level, but also to support maintenance and new production processes making a higher degree of customization possible. The growing interconnectivity also allows for the integration with more traditional IT systems as well as for the utilization of novel technologies like cloud computing. For example, services analyzing data streams to determine maintenance intervals are already in place (see, e.g., ABB ServicePort [7]). Initiatives like Industry 4.0 [21] foster these trends as they propose interconnected plants run by controllers coordinating itself using internet-based services.

In our opinion, these trends in industrial automation will have growing relevance also with respect to the application of human-oriented formal methods. In particular, based on the more extended use of standard IT and PC technology, development paradigms from software engineering and computer science can be applied in this area. This includes the use of model-based development as well as formal specification and verification technologies. Since many technically-oriented engineers have no in-depth experience with the application of the formal methods used in software development, we have to find a way lessening the burden of applying the formalisms in practice. One promising idea is Rushby's concept of "Disappearing Formal Methods" [33] that proposes to wrap formal techniques into tools in a way making them easy to use.

Our model-based engineering technique Reactive Blocks [26] supports Rushby's concept. In this article, we propose its use for the development of control software in industrial automation. We introduce Reactive Blocks in Sec. 2 arguing that it incorporates characteristics that fit well into the industrial automation domain. In Sec. 3, we clarify the use of this technique by means of a toy example which, however, is sufficient to show some of the mentioned advantages. The paper is completed by a section discussing related work as well as a conclusion.

2 Reactive Blocks in Industrial Automation

Reactive Blocks [3, 26] is a model-driven engineering technique for reactive Java-based systems. One of its features is that sub-functionality can be specified separately from each other in so-called *building blocks*. That enables us to create models of recurring sub-functionality once and to reuse them in several engineering projects. The reuse is further facilitated by providing each building block with an *External State Machine (ESM)* [23]. This is a behavioral interface allowing us to combine a building block correctly with its environment without having to completely understand its functionality.

The behavior of a building block is specified using UML activities [30]. An example of such a UML activity is depicted in Fig. 1. It contains three inner building blocks of type *Button*, *Toggle* and *LEDoperation* that all embed certain sub-functionality used[3]. The semantics of activities resembles Petri nets and corresponds to the flow of tokens via the edges towards the nodes. In this way, control and data flows are nicely visualized and can also be animated by the

[3] The content of the building block *LEDoperation* will be sketched in Sec. 3.

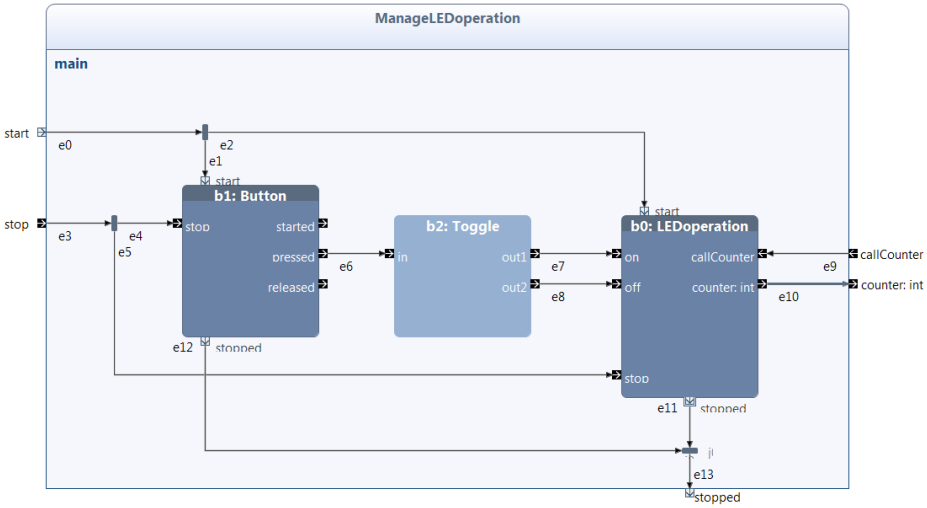

Fig. 1. The UML activity of building block *ManageLEDoperation*.

tool-set. Further, activities may contain operations that represent Java methods executed when a token passes the corresponding node. The flows are run-to-completion. That means, a flow passes all nodes on its way in the same atomic step until it reaches one that models the need to wait for a certain stimulus (e.g., a timeout or an external event).

To connect the flows of an activity containing an inner block and the one specifying the behavior of this block, we use so-called parameter nodes and pins. Parameter nodes are the little arrows at the outer edge of the activity. In the node representing an inner building block in an activity, the parameter nodes are shown as pins. For instance, the pins of the inner building block *LEDoperation* in Fig. 1 are identical to the parameter nodes in its activity (see Fig. 3). A flow reaching a pin of an inner building block will continue in the activity of this block from the corresponding parameter node and vice versa in the same run-to-completion step.

We provided the UML activities and state machines with formal semantics [24]. This allowed us to build a model checker into the tool-set [26] enabling the verification that the UML models fulfill various correctness properties (e.g., the preservation of ESMs by the activities and deadlock freedom). Following the "Disappearing Formal Methods" concept [33] mentioned in the introduction, the formal issues of the verification process are hidden to the user of the tool, while traces towards erroneous states are animated directly on the UML activity graphs. The verification runs scale thanks to the separation of functionality into different building blocks. Moreover, the formal semantics was used to verify that the automatic transformation of the models into executable Java code [25] is correct (see [22]).

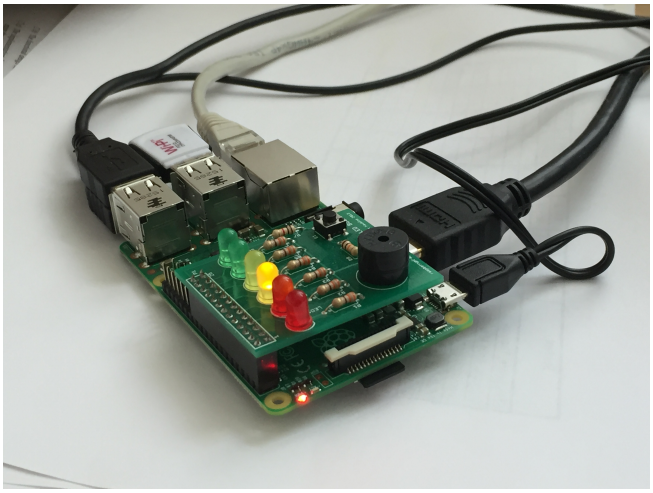

Fig. 2. Raspberry Pi-based toy example.

In our opinion, the features of Reactive Blocks makes it highly suited for the development of control software in industrial automation. For instance, the building block concept fits well to the technical engineering disciplines, in which the same physical components are often used in different applications. So, when a particular pump or valve is reused in a certain chemical plant, the building blocks realizing the control of this component may be reused in the software model of the plant as well.

Also the fact that the UML activities visualize control and data flows, is helpful for the industrial automation domain since a typical property of control software is the large number of threads running in parallel. While the coordination of the threads is difficult in classical programming languages, the run-to-completion semantics together with the clearly arranged modelling of the control and data flows facilitates the coordination of the various threads significantly.

Applying the built-in model checker leads to less errors in the generated control software. Moreover, one can couple Reactive Blocks with other analysis tools. Of particular interest for industrial automation is the composition of the tool-set with BeSpaceD [5], a tool suited to verify spatiotemporal properties (see [14, 17]). That allows us to check already on the modelling level that control software guarantees certain cyber-physical properties [19]. BeSpaceD was already used for decision support allowing to guide humans in taking high-level decisions, e.g., treating system fires (see [6]).

Another advantage of the building blocks and the ESMs is that the development of sub-functionality by various teams of experts can be nicely coordinated by embedding the sub-tasks in separate building blocks. Furthermore, the rich set of building block libraries supports the development of technical systems. For instance, the tool-set contains libraries containing various communication

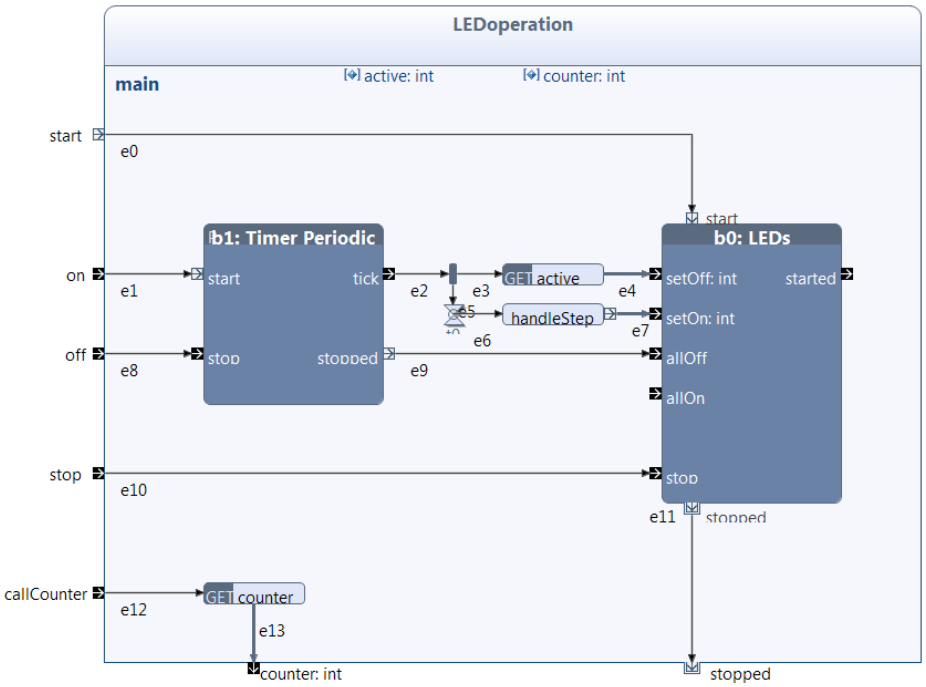

Fig. 3. The UML activity of building block *LEDoperation*.

protocols as well as blocks supporting the design of Internet of Things applications [3] that play an important role in industrial automation. We show in Sec. 3 that building blocks for control and for communication can be easily combined (see also [15]). This fits nicely with the goals of Industry 4.0 [21].

3 Example

To exemplify our approach, we use a Raspberry Pi equipped with a Berry Clip (see Fig. 2). A Berry Clip is a board provided with six colored LEDs, a buzzer, and a switch. In our toy example, a lucent LED represents a certain production sub-process and, to determine the strain of the "plant", the number of changes between the LEDs shall be sent periodically to a remote control center.

We developed the control and communication software for the example by creating three building blocks in Reactive Blocks. Figure 3 depicts the UML activity describing the behavior of the building block *LEDoperation* that realizes the operation of the LEDs on the Berry Clip. The inner block of type *LEDs* contains the functionality to switch on and off the LEDs of the Berry Clip while *TimerPeriodic* realizes a recurring timer that sends flows in even intervals (three seconds in our example).

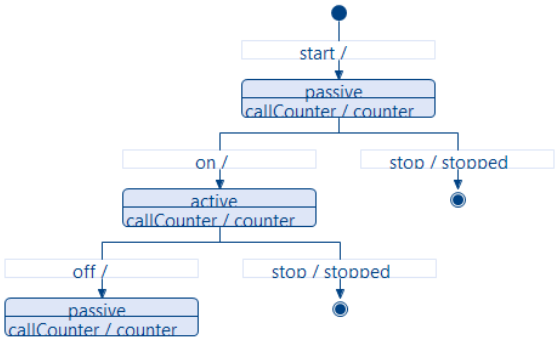

Fig. 4. The ESM of building block *LEDoperation*.

The ESM of building block *LEDoperation* is shown in Fig. 4. The block is started by a flow through parameter node `start` which is forwarded to the pin of the same name at the inner block *LEDs*. Thereafter, the ESM is in state *passive*. In this state, a flow through the parameter nodes `callCounter` and `counter` is allowed that can be used to retrieve the number of LED changes that are stored in the variable `counter`.

The lighting of the LEDs is started by a flow through the parameter node `on` bringing the ESM into state *active*. As shown in the activity, the flow starts the periodic timer. A timeout leads to a flow through pin `tick` of block *Timer-Periodic*. This flow is forked into two flows. One flow retrieves the value of the LED currently switched on, that is stored in variable `active`, and forwards it to pin `setOff` of building block *LEDs*. Thus, the currently lucent LED is switched off. The other flow reaches a flow breaker. That is a special timer without a dedicated duration used to separate a flow into different run-to-completion steps (see [24]). In our case, we use the flow breaker since the ESM of block *LEDs* does not accept flows through its pins `setOff` and `setOn` in the same run-to-completion step. The flow leaving the flow breaker reaches operation *handleStep* that represents a Java method determining the next LED to switch on, sets the selected value in variable `active` and increments the counter. After terminating the method, the flow forwards to pin `setOn` of building block *LEDs* such that the selected LED is switched on. A flow through parameter node `off` stops the lighting of the LEDs by terminating the periodic timer and switching all LEDs off. The building block can be terminated by a flow passing the parameter nodes `stop` and `stopped`.

Figure 1 shows the building block *ManageLEDoperation* modeling that the LEDs can be switched on and off by pushing the button of the Berry Clip. Here, *LEDoperation* is represented by an inner building block. Further, we use building block *Button* handling the access to the button of the Berry Clip and *Toggle* that allows us to lead button pushes mutually to the `on` and `off` pins of *LEDoperation*.

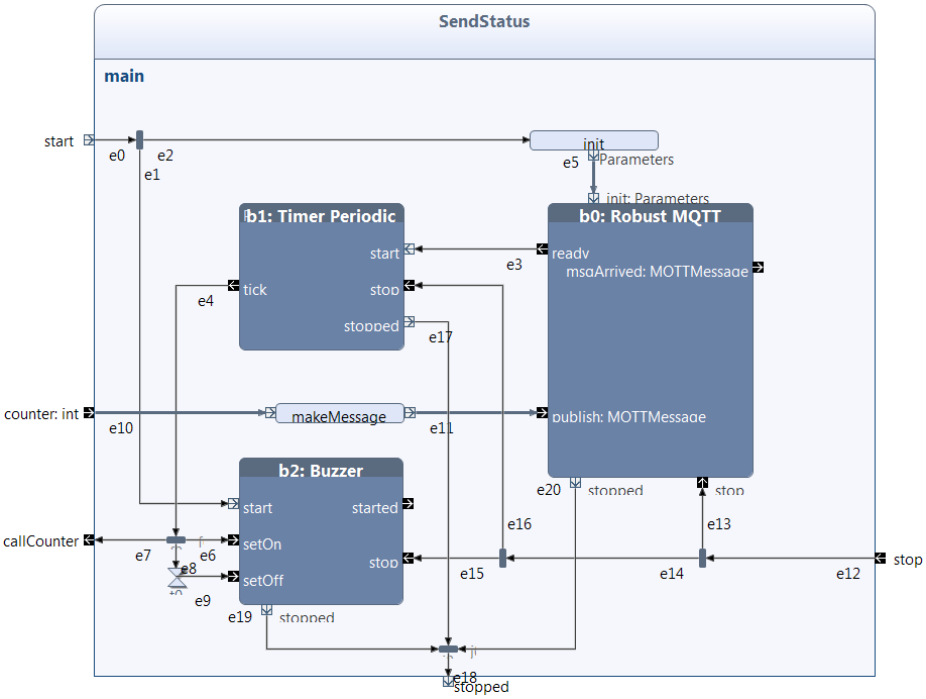

Fig. 5. The UML activity of building block *SendStatus*.

The transmission of the number of LED changes is realized by building block *SendStatus* depicted in Fig. 5. We use the popular MQTT protocol [29], the functionality of which is handled by the inner block *RobustMQTT*. Further, *SendStatus* uses another periodic timer initiating a transmission every 30 seconds. A timeout leads to a retrieval of the current counter value by a flow through parameter node `callCounter`. The value is received via parameter node `counter` that is forwarded to operation *makeMessage*. The corresponding Java method creates an object containing the MQTT message format that is forwarded to the pin `publish` of block *RobustMQTT* triggering the transmission of the counter value. Moreover, the building block contains the inner block *Buzzer* that is used to give a short audio signal using the buzzer of the Berry Clip in order to show that the status value was sent.

The activity of the overall system model is shown in Fig. 6. It consists of instances of building blocks *ManageLEDoperation* and *SendStatus*, initial triggers for these blocks, and edges connecting them to retrieve the value of the counter for the LED changes. We automatically transformed this system description into a runnable JAR file that can be directly executed on the Raspberry Pi. Moreover, we created another simple system model enabling us to receive and print out MQTT messages at a remote control station.

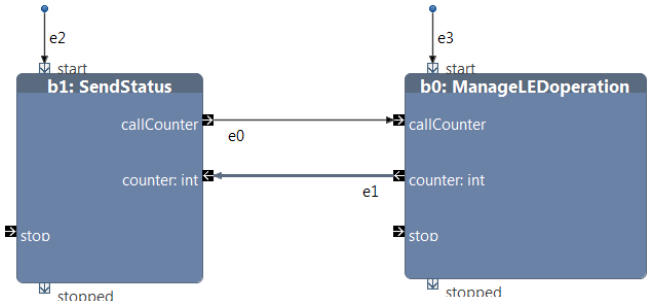

Fig. 6. The UML activity of the system.

The toy example substantiates two of the advantages named in Sec. 2. One is reusability. The complex functionality, i.e., the activation of the various units of the Berry Clip as well as the transmission via MQTT had not to be programmed manually but could be reused by simply adding already existing building blocks. Thus, the only creative task was the link of the various building blocks. Therefore the models for the Berry Clip controller and the remote station could be created by one of the authors within less than an hour. The undertaking was supported by the model checker built into Reactive Blocks since we could easily find out if all the blocks were indeed correctly coupled preserving their ESMs.

The other advantage affirmed by the example is the coordination of development teams since one can hand the creation of the building blocks *LEDoperation* and *ManageLEDoperation* over to a team of control software experts and *SendStatus* to people with in-depth knowledge about communication. Also here the model checker is of great help since it guarantees that the teams realize the ESM-based behavioral interface descriptions of the particular blocks correctly such that the results of their work can be seamlessly coupled.

4 Related Work

Formal specification of Programmable Logic Controllers (PLC) is not new but most work is based on PLC specific programming and specification techniques (see, e.g., [32, 37]). Summaries of earlier approaches to use formal methods for the specification and verification of PLC programs is given in [1, 11].

A popular application of formal methods in industrial engineering is hazard analysis of technical systems. The well-established "Hazard and Operability Studies" approach (HazOp) [28] is supplemented by formal approaches in order to verify safety properties using qualitative equation models [9, 41], Petri net models [35], and temporal logic [31]. A more general approach to specify and verify safety properties supporting the hazard analysis of chemical plants was developed by ourselves [18] based on a derivative of Lamport's Temporal Logic of Actions (TLA) [27]. There, somehow similar to Reactive Blocks, one models hybrid systems by composing existing framework modules. We further defined

and verified a set of lemmata for each framework module that facilitate safety property proofs significantly since they can be directly used as proof steps.

One of the main disadvantages of the IEC standard 61131 [20] is that it leaves some implementation and semantics aspects open to the PLC vendors. This makes formal specification and verification work difficult, but it also hinders cross platform development efforts. Some approaches such as the UNICOS toolset [12] were developed to address these shortcomings. A comprehensive model checking approach for IEC 61131–3 programs in connection with UNICOS can be found in [10]. A transformation from UML into IEC 61131 has been studied in [39]. In [13], UML is used to model control software and analysis patterns together with TLA to verify their correctness. We established Coq descriptions of IEC 61131–3 programs (see [4]) to facilitate human directed verification of PLC programs (see also [8]). Moreover, we studied the runtime monitoring of IEC 61499-based programs based on formal properties (regular expression-based) in [42]. Another formal approach based on IEC 61499 was proposed in [40]. Formal methods are also used to analyze Ethernet-based real-time communication [36].

While the application of human-oriented formal methods is new in industrial automation, the necessity for user friendly approaches is seen for the adjacent area of cyber-physical systems. For instance, [34] discusses suitable ways to ease the formal specification and verification of such systems.

5 Conclusion

In this paper, we motivated that systems bridging control automation with the classical IT world will become more mainstream in the close future. That opens the door for the application of model-based and formal methods in this application domain as well. In particular, we propose the use of Reactive Blocks for control applications in the industrial automation domain. We believe that, due to the easy use of the UML diagrams for modeling and the model checker for analysis, it facilitates the application of formal methods in the practical development of control system software also by users that are not experts in formal techniques. We exemplified our approach by discussing the development of a small Raspberry Pi-based system that, in spite of its size, is sufficient to point out some of the expected advantages.

References

1. Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M., Stursberg, O.: Verification of PLC Programs given as Sequential Function Charts. In: Integration of Software Specification Techniques for Applications in Engineering, pp. 517–540. Springer (2004)
2. Bender, K., Katz, M.: PROFIBUS: der Feldbus für die Automation. Hanser (1990)
3. Bitreactive AS: Reactive Blocks. `www.bitreactive.com` (2016), accessed: 2016-01-28
4. Blech, J.O., Biha, S.O.: Verification of PLC Properties based on Formal Semantics in Coq. In: Software Engineering and Formal Methods, pp. 58–73. Springer (2011)

5. Blech, J.O., Schmidt, H.: BeSpaceD: Towards a Tool Framework and Methodology for the Specification and Verification of Spatial Behavior of Distributed Software Component Systems. Tech. Rep. 1404.3537, arXiv.org (2014)
6. Blech, J., Peake, I., Schmidt, H., Kande, M., Rahman, A., Ramaswamy, S., Sudarsan, S., Narayanan, V.: Efficient Incident Handling in Industrial Automation through Collaborative Engineering. In: IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA). IEEE Computer (Sept 2015)
7. Boo, P.: A Service Tool grows up - ABB ServicePort. In: ABB Review (2015)
8. Canet, G., Couffin, S., Lesage, J.J., Petit, A., Schnoebelen, P.: Towards the Automatic Verification of PLC Programs written in Instruction List. In: Systems, Man, and Cybernetics. vol. 4, pp. 2449–2454. IEEE (2000)
9. Catino, C.A., Ungar, L.H.: A Model-based Approach to Automated Hazard Identification of Chemical Plants. AIChE Journal 41(3), 97–109 (1995)
10. Fernandez Adiego, B., Darvas, D., Vinuela, E.B., Tournier, J.C., Bliudze, S., Blech, J.O., Gonzalez Suarez, V.M.: Applying Model Checking to Industrial-sized PLC Programs. Industrial Informatics, IEEE Transactions on 11(6), 1400–1410 (2015)
11. Frey, G., Litz, L.: Formal Methods in PLC Programming. In: Systems, Man, and Cybernetics. vol. 4, pp. 2431–2436. IEEE (2000)
12. Gayet, P., Barillere, R.: UNICOS a Framework to Build Industry like Control Systems: Principles and methodology. In: International Conference on Accelerator and Large Experimental Physics Control Systems, Genève, Suisse (2005)
13. Graw, G.: Korrekte Steuerungssoftware. Dissertation, Technische Universität Dortmund (2010), in German
14. Han, F., Blech, J.O., Herrmann, P., Schmidt, H.: Towards Verifying Safety Properties of Real-Time Probability Systems. In: 11th International Workshop on Formal Engineering approaches to Software Components and Architectures (FESCA). EPTCS (2014)
15. Han, F., Blech, J.O., Herrmann, P., Schmidt, H.: Model-based Engineering and Analysis of Space-aware Systems Communicating via IEEE 802.11. In: 39th Annual International Computers, Software & Applications Conference (COMPSAC). pp. 638–646. IEEE Computer (2015)
16. Harland, J., Blech, J.O., Peake, I., Trodd, L.: Formal Behavioural Models to Facilitate Distributed Development and Commissioning in Industrial Automation. In: Evaluation of Novel Approaches to Software Engineering, COLAFORM Track (2016)
17. Herrmann, P., Blech, J.O., Han, F., Schmidt, H.: A Model-based Toolchain to Verify Spatial Behavior of Cyber-Physical Systems. International Journal of Web Services Research (IJWSR) 13(1), 40–52 (2016)
18. Herrmann, P., Krumm, H.: A Framework for the Hazard Analysis of Chemical Plants. In: 11th IEEE International Symposium on Computer-Aided Control System Design (CACSD2000). pp. 35–41. IEEE CSS, Anchorage (Sep 2000)
19. Hordvik, S., Øseth, K., Blech, J.O., Herrmann, P.: A Methodology for Model-based Development and Safety Analysis of Transport Systems. In: 11th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE) (2016)
20. IEC: IEC Standard IEC 61161-3. Programmable Controllers — Programming Languages, edition 2.0 edn. (01 2003)
21. Kagermann, H., Wahlster, W., Helbig, J.: Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0. Abschlussbericht des Arbeitskreises Industrie 4, 5 (2013), in German

22. Kraemer, F.A.: Engineering Reactive Systems: A Compositional and Model-Driven Method Based on Collaborative Building Blocks. Ph.D. thesis, Norwegian University of Science and Technology (2008)
23. Kraemer, F.A., Herrmann, P.: Automated Encapsulation of UML Activities for Incremental Development and Verification. In: Model Driven Engineering Languages and Systems (MoDELS). pp. 571–585. LNCS 5795, Springer-Verlag (2009)
24. Kraemer, F.A., Herrmann, P.: Reactive Semantics for Distributed UML Activities. In: Joint WG6.1 International Conference (FMOODS) and WG6.1 International Conference (FORTE). pp. 17–31. LNCS 6117, Springer-Verlag (2010)
25. Kraemer, F.A., Herrmann, P., Bræk, R.: Aligning UML 2.0 State Machines and Temporal Logic for the Efficient Execution of Services. In: 8th International Symposium on Distributed Objects and Applications (DOA06). pp. 1614–1632. LNCS 4276, Springer-Verlag (2006)
26. Kraemer, F.A., Slåtten, V., Herrmann, P.: Tool Support for the Rapid Composition, Analysis and Implementation of Reactive Services. Journal of Systems and Software 82(12), 2068–2080 (2009)
27. Lamport, L.: Specifying Systems: The TLA$^+$ Language and Tools for Hardware and Software Engineers. Pearson Education, Inc (2002)
28. Lawley, H.G.: Operability Studies and Hazard Analysis. Chemical Engineering Progress 70(4), 45–56 (1974)
29. MQTT.org: Message Queuing Telemetry Transport (MQTT). `mqtt.org/`, accessed: 2015-08-14
30. Object Management Group: OMG Unified Modeling LanguageTM (OMG UML), Superstructure — Version 2.4.1. `www.omg.org/spec/UML/2.4.1/Superstructure/PDF/` (2011), accessed: 2016-01-28
31. Probst, S.T.: Chemical Process Safety and Operability Analysis using Symbolic Model Checking. Ph.D. thesis, Carnegie Mellon University, Pittsburgh (1996)
32. Rausch, M., Krogh, B.H.: Formal verification of PLC programs. In: American Control Conference. vol. 1, pp. 234–238. IEEE (1998)
33. Rushby, J.: Disappearing Formal Methods. In: High-Assurance Systems Engineering Symposium. pp. 95–96. ACM, Albuquerque, ACM (2000)
34. Spichkova, M., Zamansky, A., Farchi, E.: Towards a Human-Centred Approach in Modelling and Testing of Cyber-Physical Systems. Tech. Rep. 1601.06222, arXiv.org (2016)
35. Srinivasan, R., Venkatasubramanian, V.: Petri Net-Digraph Models for Automating HAZOP Analysis of Batch Process Plants. Computers Chemical Engineering 20, 719–725 (1996)
36. Steiner, W., Dutertre, B.: SMT-based Formal Verification of a TTEthernet Synchronization Function. In: Formal Methods for Industrial Critical Systems, pp. 148–163. Springer (2010)
37. Stursberg, O., Kowalewski, S., Hoffmann, I., Preußig, J.: Comparing Timed and Hybrid Automata as Approximations of Continuous Systems. In: Hybrid Systems IV, pp. 361–377. Springer (1996)
38. Upton, E., Halfacree, G.: Raspberry Pi User Guide. Wiley (2014)
39. Vogel-Heuser, B., Witsch, D., Katzke, U.: Automatic Code Generation from a UML Model to IEC 61131-3 and System Configuration Tools. In: International Conference on Control and Automation (ICCA). vol. 2, pp. 1034–1039. IEEE (2005)
40. Vyatkin, V., Hanisch, H.M.: Formal Modeling and Verification in the Software Engineering Framework of IEC 61499: A Way to Self-verifying Systems. In: Emerging Technologies and Factory Automation (ETFA). vol. 2. IEEE Computer (2001)

41. Waters, A., Ponton, J.W.: Qualitative Simulation and Fault Propagation in Process Plants. Chemical Engineering Research Descriptions 67, 407–422 (1989)
42. Wenger, M., Blech, J.O., Zoitl, A.: Behavioral Type-based Monitoring for IEC 61499. In: Emerging Technologies and Factory Automation (ETFA). IEEE Computer (2015)