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 formal 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 [17] 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 were 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 [30] came into the market offering 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 [13]. 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 communicate now with each other and with other external devices and services, not

P. Milazzo et al. (Eds.): STAF 2016, LNCS 9946, pp. 253–261, 2016.
DOI: 10.1007/978-3-319-50230-4_19

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 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 [6]). Initiatives like Industry 4.0 [18] 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 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 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" [27] that proposes the wrapping of formal techniques into tools such that they are easy to use. Our model-based engineering technique Reactive Blocks [22] supports Rushby's concept. In this article, we propose its use for the development of control software in industrial automation.

2 Reactive Blocks in Industrial Automation

Reactive Blocks [3,22] is a model-driven engineering technique for reactive Java-based systems. It uses UML activity and state machine diagrams [25] to model systems. Since these diagram types are innately not provided with formal semantics, we defined one ourselves. In [23], we defined an initial formal semantics for

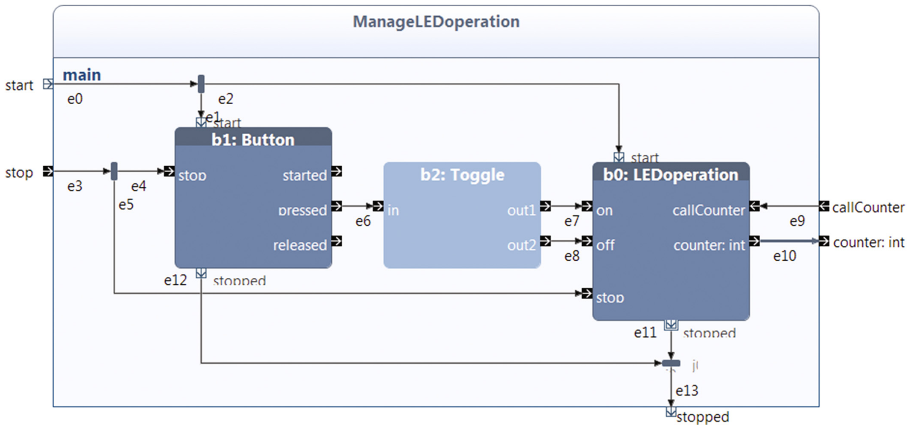

Fig. 1. The UML activity of building block *ManageLEDoperation*.

an early version of the tool based on cTLA [15], a variant of Lamport's Temporal Logic of Actions (TLA) [24]. Becoming more experienced with the tool, we later defined the so-called *reactive semantics* [20]. Since UML activities are basically graphs, we based it on rules in traditional graph theory.

One of the features of Reactive Blocks 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)* [19]. This is a behavioral interface allowing us to combine a building block correctly with its environment without having to completely understand its functionality.

A UML activity is used to model the behavior of a building block. The activity depicted in Fig. 1 contains three inner building blocks of type *Button*, *Toggle* and *LEDoperation* that all embed certain sub-functionality. The reactive semantics of the 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 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 (see [20]). 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 (i.e., 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. 2). 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.

Thanks to the formal reactive semantics, we could build a model checker into the tool-set [22] 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 [27] mentioned in the introduction, the formal issues of the verification process are hidden to the user of the tool, and 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 UML models can be automatically transformed into executable Java code [21].

In our opinion, the features of Reactive Blocks makes it highly suited for the development of control software in industrial automation. 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]). That allows us to check already on the modelling level that control software guarantees certain cyber-physical properties [16].

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 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 Sect. 3 that building blocks for control and for communication can be easily combined (see also [12]). This fits nicely with the goals of Industry 4.0 [18].

3 Example

We demonstrate our approach by using a Raspberry Pi equipped with a Berry Clip, i.e., 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. 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 2 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).

The ESM of building block *LEDoperation* is shown in Fig. 3. 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. It can be used by the environment of the building block to retrieve the number of LED changes that are stored in the variable `counter`.

The rotative 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

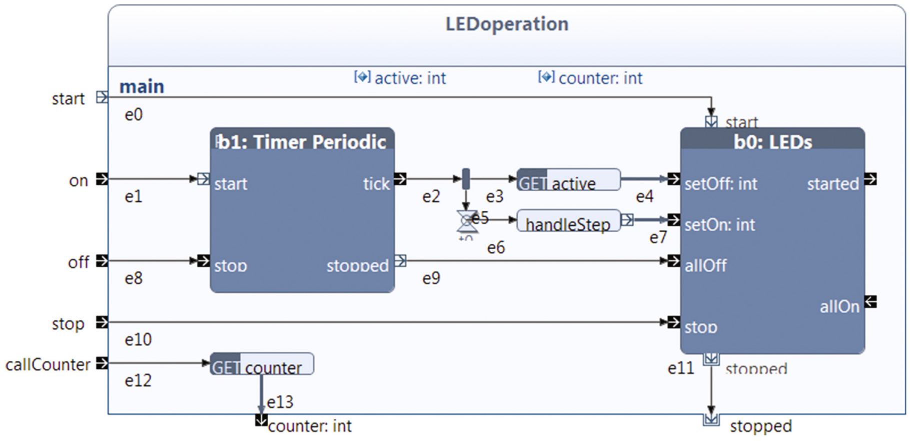

Fig. 2. The UML activity of building block *LEDoperation*.

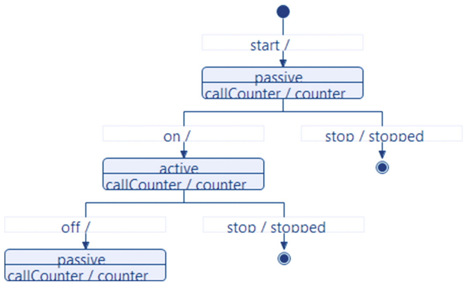

Fig. 3. The ESM of building block *LEDoperation*.

starts the periodic timer. A timeout leads to a flow through pin `tick` of block *TimerPeriodic*. 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. 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

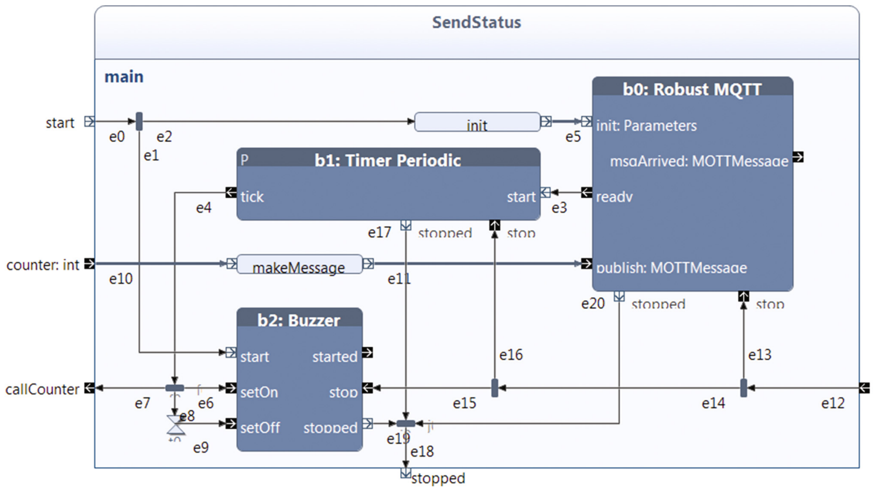

Fig. 4. The UML activity of building block *SendStatus*.

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*.

The transmission of the number of LED changes is realized by building block *SendStatus* depicted in Fig. 4. We use the popular MQTT protocol, the functionality of which is handled by the inner block *RobustMQTT*. Further, *SendStatus* uses another periodic timer initiating a transmission every 30 s. 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 quite simple. It consists of instances of building blocks *ManageLEDoperation* and *SendStatus*, initial triggers for these blocks, and edges connecting their pins `callCounter` resp. `counter`. 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.

The toy example substantiates two of the advantages named in Sect. 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., [26,29]). Summaries of earlier approaches to use formal methods for the specification and verification of PLC programs is given in [1,9].

One of the main disadvantages of the IEC standard 61131 [17] 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 [10] were developed to address these shortcomings. A comprehensive model checking approach for IEC 61131–3 programs in connection with UNICOS can be found in [8]. A transformation from UML into IEC 61131 has been studied in [31]. In [11], UML is used to model control software and analysis patterns together with cTLA (see [15]) 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 [7]). Another formal approach based on IEC 61499 was proposed in [32]. Formal methods are also used to analyze Ethernet-based real-time communication [28].

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: Ehrig, H., Damm, W., Desel, J., Große-Rhode, M., Reif, W., Schnieder, E., Westkämper, E. (eds.) Integration of Software Specification Techniques for Applications in Engineering. LNCS, vol. 3147, pp. 517–540. Springer, Heidelberg (2004). doi:10.1007/978-3-540-27863-4_28
2. Bender, K., Katz, M.: PROFIBUS: der Feldbus für die Automation. Hanser (1990)
3. Bitreactive, A.S.: Reactive Blocks. www.bitreactive.com. Accessed 28 Jan 2016
4. Blech, J.O., Ould Biha, S.: Verification of PLC properties based on formal semantics in Coq. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM 2011. LNCS, vol. 7041, pp. 58–73. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24690-6_6
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. Technical report 1404.3537. arXiv.org (2014)
6. Boo, P.: A service tool grows up - ABB ServicePort. In: ABB Review (2015)
7. 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)
8. 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. IEEE Trans. Ind. Inform. **11**(6), 1400–1410 (2015)
9. Frey, G., Litz, L.: Formal methods in PLC programming. In: Systems, Man, and Cybernetics, vol. 4, pp. 2431–2436. IEEE (2000)
10. 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)
11. Graw, G.: Korrekte Steuerungssoftware. Dissertation, Technische Universität Dortmund (2010) (in German)
12. 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 and Applications Conference (COMPSAC), pp. 638–646. IEEE Computer (2015)
13. 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)
14. Herrmann, P., Blech, J.O., Han, F., Schmidt, H.: A model-based toolchain to verify spatial behavior of cyber-physical systems. Int. J. Web Serv. Res. (IJWSR) **13**(1), 40–52 (2016)
15. Herrmann, P., Krumm, H.: A framework for modeling transfer protocols. Comput. Netw. **34**(2), 317–337 (2000)

16. 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)
17. IEC: IEC Standard IEC 61161–3. Programmable Controllers – Programming Languages, 2.0 edn. (01 2003)
18. Kagermann, H., Wahlster, W., Helbig, J.: Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0. Abschlussbericht des Arbeitskreises Industrie 4, 5 (2013) (in German)
19. Kraemer, F.A., Herrmann, P.: Automated encapsulation of UML activities for incremental development and verification. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 571–585. Springer, Heidelberg (2009). doi:10.1007/978-3-642-04425-0_44
20. Kraemer, F.A., Herrmann, P.: Reactive semantics for distributed UML activities. In: Hatcliff, J., Zucca, E. (eds.) FMOODS/FORTE -2010. LNCS, vol. 6117, pp. 17–31. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13464-7_3
21. Kraemer, F.A., Herrmann, P., Bræk, R.: Aligning UML 2.0 state machines and temporal logic for the efficient execution of services. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4276, pp. 1613–1632. Springer, Heidelberg (2006). doi:10.1007/11914952_41
22. Kraemer, F.A., Slåtten, V., Herrmann, P.: Tool support for the rapid composition, analysis and implementation of reactive services. J. Syst. Softw. **82**(12), 2068–2080 (2009)
23. Kraemer, F.A., Herrmann, P.: formalizing collaboration-oriented service specifications using temporal logic. In: Networking and Electronic Commerce Research Conference (NAEC), pp. 194–220. ATSMA, Riva del Garda, October 2007
24. Lamport, L.: Specifying Systems: The TLA$^+$ Language and Tools for Hardware and Software Engineers. Pearson Education Inc, London (2002)
25. Object Management Group: OMG Unified Modeling LanguageTM (OMG UML), Superstructure – Version 2.4.1 (2011). www.omg.org/spec/UML/2.4.1/Superstructure/PDF/. Accessed 28 Jan 2016
26. Rausch, M., Krogh, B.H.: Formal verification of PLC programs. In: American Control Conference, vol. 1, pp. 234–238. IEEE (1998)
27. Rushby, J.: Disappearing formal methods. In: High-Assurance Systems Engineering Symposium, pp. 95–96. ACM. Albuquerque (2000)
28. Steiner, W., Dutertre, B.: SMT-Based formal verification of a *TTEthernet* synchronization function. In: Kowalewski, S., Roveri, M. (eds.) FMICS 2010. LNCS, vol. 6371, pp. 148–163. Springer, Heidelberg (2010). doi:10.1007/978-3-642-15898-8_10
29. Stursberg, O., Kowalewski, S., Hoffmann, I., Preußig, J.: Comparing timed and hybrid automata as approximations of continuous systems. In: Antsaklis, P., Kohn, W., Nerode, A., Sastry, S. (eds.) HS 1996. LNCS, vol. 1273, pp. 361–377. Springer, Heidelberg (1997). doi:10.1007/BFb0031569
30. Upton, E., Halfacree, G.: Raspberry Pi User Guide. Wiley, Cambridge (2014)
31. 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)
32. 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)