Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 00 (2017) 000–000

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

28th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2018), June 11-14, 2018, Columbus, OH, USA

Composition of Modular Models for Verification of Distributed Automation Systems

Andreas Zeller, Michael Weyrich

Institute of Industrial Automation and Software Engineering, University of Stuttgart, 70569 Stuttgart, Germany

Abstract

The increasing complexity of distributed automation systems requires new methods to verify the correct functionality. Model-based verification is an established approach to test the behavior of the system under test, before going into operation. To apply model-based techniques the overall system model of the automation system is needed. Due to the high complexity of the overall system and the changing dependencies caused by reconfigurations or software modifications an overall system model is seldom available or maintained. In this paper, we propose a modeling technique to manage the complexity of the overall system by modularization which is dedicated to distributed systems. This is presented in a formal way. Thereby, the modeling techniques regard the requirements of a service-oriented-architecture and the properties of automation systems, like interfaces to the technical process and parallelism caused by the distribution. In addition, we present calculation rules how to build up the overall system model automatically which can be used to verify system requirements.

Keywords: Verification; Testing; Automation; Industry 4.0; Model Composition, Software Modification;

1. Introduction

Due to low-cost micro-controllers and sophisticated software tool chains, complex functionality of automation systems are increasingly realized by software [5]. In Smart Factories and Industry 4.0 areas, there's a demand for highly flexible and modifiable production systems. Like it is already today relevant in the automotive sector, it's to be expected that software modifications will occur more frequently within the operation phase of an industrial automation system. To realize this modifiability, automation systems require distributed control solutions [6]. To hide

* Andreas Zeller. Tel.: +49-711-685-67291.
E-mail address: andreas.zeller@ias.uni-stuttgart.de

the complexity of the components of the system, their functionality is abstracted by services. These services expose their functionality by well-defined interfaces. This abstraction allows to use this service without the need to know about its internal structure. Such distributed systems can for example be realized by service-orientation or agent-based. In the following, service-oriented systems are considered. This can also be transferred to other distributed systems. These approaches enable ad-hoc networking by coupling the components on runtime (loose coupling) and semantically described interfaces. The encapsulation of functions in services allows easy software modifications of the components.

Apart from the high flexibility, a service-oriented structure leads to new challenges to ensure the correct functionality of the automation system [10]. System functionality is distributed among different components. This leads to dependencies between the system components. The effort to analyze the dependencies of distributed systems will rise because of a higher number of communication channels [5]. Due to software modifications and ad-hoc networking these dependencies change over time. Nevertheless to ensure the correct functionality of the system, systematic and automated test methods are required.

Model-based Verification is an established approach to test the model of the system before going into operation. To that the system model is verified against its requirements. When modifying the logic of a component the system model has to be adapted as system models of distributed automation systems can become pretty large and complex. This isn't an easy task. The result is that system models are seldom maintained.

In this paper a methodology is proposed on how the composition of a system model can be automated. Therefore a suitable modeling technique for verification of service-oriented systems is presented. Thereby properties of service-oriented architectures are used to simplify the composition process.

2. State of the art

Due to the rising complexity of industrial automation systems, methodologies and techniques for a comprehensive and efficient test gain in importance. In the field of industrial automation there are several approaches regarding model-based testing to automate and structure the test process. For instance the test architecture, test environment, test data, test cases or the system under test can be modeled [9].

Model-based verification is an aspect of model-based-testing and regards the testing of a system against its requirements. The correct logical behavior of a system model, according to its requirements, can be verified with formal methods, theorem-proving or simulatively by executing test cases. Verification of the model against its requirements instead of the real system leads to several advantages. The risk of physical damage to the system when testing is eliminated. The verification process can be done before modifying the real system. This reduces the downtime of the production systems. Model-based verification techniques have in common, that a formal system model is needed. To model the systems different modeling languages originated for different areas of applications. In informatics, model-based verification techniques are very popular and well established. There are similar approaches in the field of industrial automation, some of which will be described in the following.

The DYMOLA environment uses the Modellica language to describe and simulate complex systems as well as their interaction. With the help of this powerful language, it's possible to describe, apart from the control systems, the mechanical, electrical, thermodynamic, hydraulic, pneumatic and thermal behavior of a system. It can be used to validate a control system closed loop with an integrated plant model [11]. In paper [2] is described how PLC-Code, written in the programming language IL (IEC 61131-3), is verified with the help of formal methods. Therefore the code has to be transformed into a model. There are also other approaches transforming IL into formal models listed in [8]. Another approach presents a framework for modeling logic controllers and the technical process. Thereby the logic is modeled with untimed state machines and embedded in a time-dependent signal space, which interacts with the state machine via input and output signals [1]. Upaal is an integrated environment to model and verify real-time systems. For modeling, timed automata which can be extended are used. The system model can be verified against requirements formalized in CTL [12]. NCES (Net Conditions / Event Systems) are designed to model modular control software. The basis is a typical petri net which is extended by event signals and condition signal to build the interface between different components. These new introduced arcs don't correspond with the typical petri net arcs which lead from a place to a transition and reverse. NCES are used in different scenarios, like HiL-Testing, formal verification against CTL requirements and others [3]. [4] presents a modeling method to describe the time behavior of

networked automation systems for simulation as well as for verification via model checking. In informatics, service-orientation and verification techniques are long established. ”Open nets” originate from this domain and describe the interaction of services with the help of extended petri nets. This approach introduces input- and output-places to describe the interface of the components which are orchestrated to a more complex service. By using interface places, the definition of these open nets doesn’t have to be extended with new arcs. So the behavior of the composed petri still follows the petri-net switching rules. Most modeling techniques base on state machines or petri nets. Petri nets offer better abilities to model parallel behavior, an indispensable requirement in distributed systems. NCES use petri nets to model distributed systems. Due to the extension with other new introduced arcs the modeling technique is getting more powerful but also more complex and unintuitive. Open nets, another petri net based approach, cover the requirements of asynchronous communication which are necessary for service-oriented networks without the need of new arcs. As open nets originate from the field of informatics, they don’t have an interface description to the technical process.

In this paper a concept for building up system models is described. The focus is on the suitability for distributed, service-oriented automation systems, an intuitive usability and a good tool support by using established modeling techniques. For this, the presented approaches were regarded and a suitable modeling technique was designed. Thereby just the modeling of the automation system and the interfaces to the technical process shall be regarded. The model of the technical process is assumed to be given.

3. Modeling concept

Due to the fact, that automation systems are getting more complex, an important requirement for the modeling concept is to keep the complexity of the models on a manageable level. For this purpose, the modular structure of a service-oriented system is transferred to the modeling technique. That allows for an easy modifiability of the single components. In the following, the modeling technique for the internal structure of the components and the calculation rules for an automated composition of the components to a system model are introduced. The resulting model is suitable as an input for verification.

3.1. Modularization

The modular structure of the service-oriented network is transferred to a modular modeling approach as illustrated in Fig. 1 b). Each component is represented as an element in the system model and is connected to other elements by its interfaces. Interfaces are differentiated between IT-interfaces to other components and interfaces to or from the technical process. IT-Interfaces are described by the message-description they use. Thereby just the messages are taken into account, which affect the logic behavior of the component. The input message of component 1 is represented by ”message A” and the output message by ”message B”. ”Message B” can be the input message of another component. ”Action X” depicts the interference of the technical process by the components. ”Event Y” depicts the interference of the component by the technical process. Actions are present in actuators, events in sensors. Interfaces from and to the technical process are depicted by underlined action descriptions. By parsing the interfaces of the components of the automation systems, it’s possible to analyze the dependencies between the components and compose the system model.

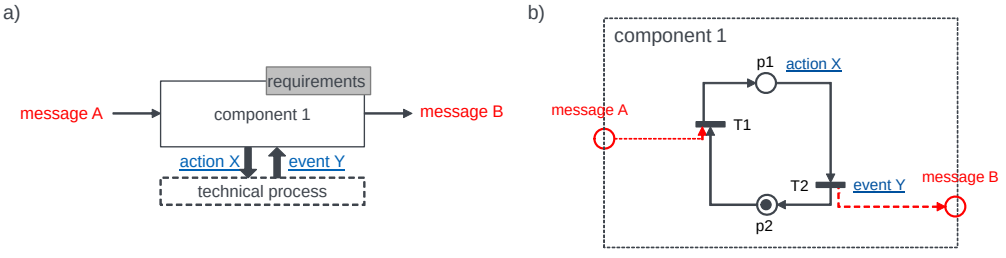

Fig. 1. a) Abstracted view of an automation component. b) Graphical representation of a minimal open net

As well as the functional system is modularized, the requirements for the system are modularized (see Fig. 1). Each component has its functional requirements mapped. The requirements are extracted from the component specification and described in a formal way like CTL. The requirements reference on the functionality of the respective component. If a component is dependent from other components, its requirements can't be verified without including the interaction with these components (integration test). If there are several components offering the same dependent service, the verification has to be done with every possible constellation. That's important because components can offer the same service, but differ in the internal structure. A suitable modeling technique for the internal structure of a component is introduced in the following.

3.2. Modeling of a component

Open nets have been introduced in the state of the art section. These nets were first mentioned in 2005 as "open workflow nets" and are designed to analyze the interaction of different services. A big advantage of this approach is that the composition of the open nets still has the same type of elements and arcs as a typical petri net. The authors extended this approach with further attributes to adapt the modeling techniques to the requirements for industrial automation systems. Thereby attributes of signal interpreted petri nets (SIPN), which are popular in PLC programming, are appended. The model is defined by following 9-tuple. $\mathbf{N} =< \mathbf{P}, \mathbf{T}, \mathbf{F}, \mathbf{M_0}, \mathbf{\Omega}, \mathbf{I}, \mathbf{A}, \mathbf{S}, \mathbf{L} >$

- P: finite set of spaces.
- T: finite set of transitions.
- F: finite set of of flow relations between transitions and places and reverse $F \subseteq (P \times T) \cup (T \times P)$.
  - set of predecessor places of transitions described via $\cdot t = \{p|(p,t) \in F\}$
  - set of successor places of transitions described via $t\cdot = \{p|(t,p) \in F\}$.
  - set of predecessor transitions of places described via $\cdot p = \{t|(t,p) \in F\}$
  - set of successor places of transitions described via $p\cdot = \{t|(p,t) \in F\}$.
- M_0: initial marking, represented via a vector with the cardinality $|P|$.
- Ω: finite set of markings where N is allowed to terminate.
- I: interface places $I \in P$. Places to exchange messages with other open nets. The description of the places corresponds with the messages being exchanged. For minimal open nets apply:
  - input places I_{in} which receive messages don't have predecessor transitions: $\cdot p = \emptyset$.
  - output places I_{out} which send messages don't have successor transitions: $p\cdot = \emptyset$.
  - an input place can't also be an output place: $I_{in} \cap I_{out} = \emptyset$
- A: interface to the technical process. Actions which are executed to the technical process when a place is occupied. A vector with the cardinality: $|A| \equiv |P|$.
- S: interface from the technical process. Events from the technical process trigger transitions. A vector with the cardinality: $|S| \equiv |T|$.
- L: latency of the arcs. A vector with the cardinality: $|L| \equiv |F|$.

The graphical representation of a minimal open net is illustrated in Fig. 1 b). The internal structure of the pictured minimal open net resembles a typical petri net with two places "p1" and "p2". The dashed surrounding represents the component boarder. On the left side the input place(s) and on the right side the output place(s) are located. The output places act as input places for open nets of other components which consume the token. The interface places correspond with the messages of the abstracted presentation of the components in Fig. 1. As well as in Fig. 1 the interface to and from the technical process are illustrated by an underlined description. The actions A are mapped to places and events S are mapped to transitions. This is in conformity to SIPNs. In the initial marking p2 contains a token. When component 1 receives "message A" a token is generated in the respective input place and transition T1 becomes switchable. Analogous to SIPN a transition switches as soon as it is switchable. Tokens from the predecessor places are deducted and generated in the successor places p1. "Action X" is executed as long as the token remains in p1. When "event Y" occurs the guard activates T2. The token of p1 is deducted and tokens in "message B" and p2 are generated. This means that the system sends "message B" and passes into its initial state.

3.3. Composition of subsystems to an overall system

Many requirements refer to a functionality which is distributed on several components. To verify these requirements model-based, an interconnected net of these components is needed. The mathematical rules to compose these minimal open nets to an interconnected net are defined as:

$\mathbf{N_{all}} =< \mathbf{P_{all}}, \mathbf{T_{all}}, \mathbf{F_{all}}, \mathbf{M_{0,all}}, \mathbf{\Omega_{all}}, \mathbf{I_{all}}, \mathbf{A_{all}}, \mathbf{S_{all}}, \mathbf{L_{all}} >$

- $P_{all} = P_{Comp.1} \cup P_{Comp.2} \cup ... \cup P_{Comp.n}$
- $T_{all} = T_{Comp.1} \cup T_{Comp.2} \cup ... \cup T_{Comp.n}$
- $F_{all}(p,t) = F_{Comp.1}(p,t) \cup F_{Comp.2}(p,t) \cup ... \cup F_{Comp.n}(p,t)$
- $F_{all}(t,p) = F_{Comp.1}(t,p) \cup F_{Comp.2}(t,p) \cup ... \cup F_{Comp.n}(t,p)$
- $M_{0,all} = M_{0,Comp.1} \oplus M_{0,Comp.2} \oplus ... \oplus M_{0,Comp.n}$
- $\Omega_{all} = \Omega_{Comp.1} \oplus \Omega_{Comp.2} \oplus ... \oplus \Omega_{Comp.n}$
- $I_{all} = I_{Comp.1} \cup I_{Comp.2} \cup ... \cup I_{Comp.n}$
- $A_{all} = A_{Comp.1} \cup A_{Comp.2} \cup ... \cup A_{Comp.n}$
- $S_{all} = S_{Comp.1} \cup S_{Comp.2} \cup ... \cup S_{Comp.n}$
- $L_{all} = L_{Comp.1} \cup L_{Comp.2} \cup ... \cup L_{Comp.n}$

The components are connected via their interface places. If there are several components offering the same service, they all have the same predecessor place. This reflects the redundancy of the real system. The other way round, when there are several components calling the same services, they have the same successor place. The composed net isn't a minimal open net anymore, because an interface place can now be an input place as well as an output place at the same time. But it still has the behavior of a typical petri net. That makes it suitable for conventional verification tools. The visual representation of the composition is pictured in the following use case.

4. Use-case

The concept presented is illustrated with the help of the use case of a discrete process. As shown in Fig. 2 the technical process consists of a conveyor belt which transports the workpiece to a defined position. The conveyor belt has a modular design and consists of a motor as well as a proximity sensor. The technical process is controlled by a distributed control network. The control of the conveyor belt accesses the motor of the conveyor belt as well as the proximity sensor to fulfill the transportation task. The actuators and sensors possess a process near control. The modifiability of the distributed control system is guaranteed by service-orientation. This results in an event-oriented asynchronous communication. Each entity offers a service which is well-defined. That means the service consists of a semantic description. That is, as well as the coupling on runtime, a prerequisite to realize adhoc networking. The control of the conveyor belt offers the service "'move_workpiece"', the motor of the conveyor belt offers the service "'move_conveyor"' and the proximity sensor offers the service "'detect_workpiece"'.

The semantically described services allow the usage of the functionality without the need to know the internal behavior of its component. Thus, the complexity is hidden by the abstraction of the functionality by the service. The decentralized communication architecture and the semantically described interfaces allow software-modification of the components as well as the integration of new components on runtime. But what effects does software-modification of a component or ad-hoc integration have on other system components? As described in the introduction it's hard to get the dependencies between the software components due to the loose coupling as well as the changing relations between the components due to software modifications. The applicability of the proposed concept is elucidated by

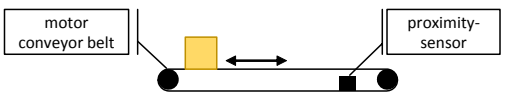

Fig. 2. Physical build up of a discrete production facility

means of this use case. The abstract view of the modules of the use case is exemplarily depicted by the conveyor belt, by the motor of the conveyor belt as well as by the proximity sensor (see Fig. 3).

The conveyor belt has two input message which trigger the internal behavior of the system as well as four output messages which are sent to other components. The motor of the conveyor belt has two actions ”motor ON” as well as ”motor OFF”. The proximity sensor is influenced by the event ”workpiece_detected” from the technical process. The internal behavior of the components can be modeled by open nets. The graphical representations of the components ”control conveyor belt”, ”motor conveyor belt”, ”proximity sensor” are illustrated in Fig. 4.

The control of the conveyor belt contains two places ”convey ON” and ”convey OFF”. In the initial marking a token is in the state ”convey OFF”. The token alternates between these two places via the transitions T1 and T2. The input places I_{in} and output places I_{out} are respective to the input and output messages illustrated in Fig. 3. When the service ”move_workpiece” is called, the tokens from the predecessor places are deducted and in the place ”convey ON” as well as in the output places ”call_detect_workpiece” and ”call_move_conveyor(On)” tokens are generated. In this case, the service of the proximity sensor as well as the service of the motor are called. When the service of the proximity sensor is accomplished, a token appears in the input place ”ack_detect_workpiece”, the Transition T2 becomes switchable and the tokens of its predecessor places are deducted and appended to its successor. By the output place ”call_move_conveyor(Off)” the motor is switched off again and by the output place ”ack_move_workpieces” the completion of the service ”move_workpieces” is reported to the client who called the service. The motor of the conveyor belt (pictured at the bottom of Fig. 4) has a physical interface to the technical process. This is activated as long as a token is in the respective place. The proximity sensor (pictured in the middle of Fig. 4) contains the event ”workpiece_detected”. The event ”workpiece_detected” activates the transition between the places ”wait_for_workpieces” and ”idle”. To verify requirements which concern several components, the interconnected net of these components is needed. The requirements of the control of the conveyor belt are dependent on other components. To verify its requirements the interaction with the motor of the conveyor belt as well as with the proximity sensor has to be considered. To verify this requirements model-based the open nets of the components have to be composed. Composing them according to the calculation rules presented leads to the net illustrated in Fig. 5.

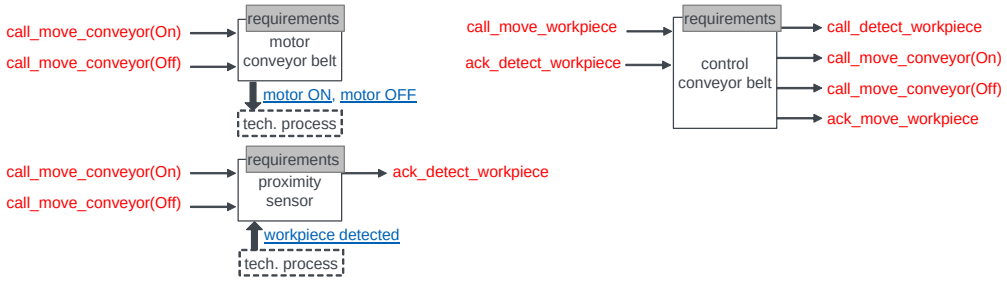

Fig. 3. Modularized view of part of the automation system. The internal structure of the components is abstracted. The components are interconnected via their interfaces.

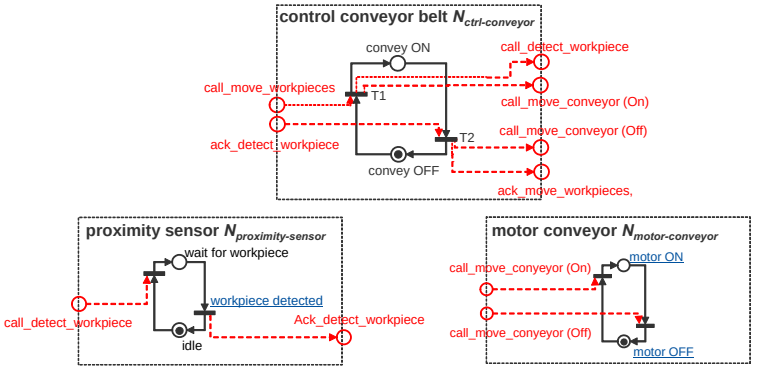

Fig. 4. Open nets of the components control conveyor belt, motor conveyor belt and proximity sensor

The next chapter describes how these algorithms can be used to improve the verification process by automation of the creation of the system model.

5. Realization and implementation

The concept described is implemented and evaluated with a distributed automation system and a Test-Box which have been implemented. The Test-Box which is picture in Fig. 6 a) has two interfaces. By a wrapper, the Test-Box can interact with a distributed automation system. In this case the automation system is realized by an OPC-UA network that contains 153 services distributed on six OPC-UA servers. A virtual reality simulation, depicted in Fig. 6 b) allows the technical process to scale with ease. The Test-Box functions as a client and integrates ad-hoc into the network. The wrapper represents an adapter to the automation system. This allows a technology independent implementation of the algorithms of the test box. The other interface leads to data bases. These data bases are model storages. For each component a model has to exist in the database. The models of a mass product, for example a sensor or actuator, have to be created once and can be reused for different use cases afterwards. The model of individual components, like controller programs, have to be created by the plant manufacturer. Since model-based engineering is gaining importance, it can be assumed that these models will arise with few additional effort. Additionally, the requirements of these components have to be mapped to the models. Because of the modular build up of such a system, the engineer who builds the model and maps the requirements to the model just needs the knowledge about the specific components and not about other components, the system is interacting with. When integrating the Test-Box in an automation system the test box starts the process depicted in the box in Fig. 6. The Test-Box scans the automation system for existing components. This includes their unique ID as well as their version. By means of this information the test box browses the database for the open nets and their requirements describing the identified components. The requirements are described in CTL. In the next step the models are composed into a system model. The composition is necessary to verify the requirements of components which are dependent on other components. In the last step the system model is verified against its requirements and the verification results are displayed. It could be shown that the component models were composed correctly and a verification of given system requirement could be performed succesfully.

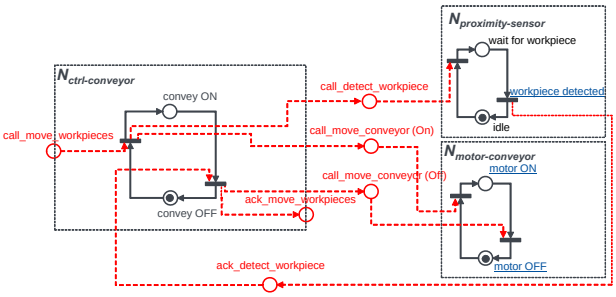

Fig. 5. Composed open net of the minimal open nets of the components control conveyor belt, motor conveyor belt and proximity sensor

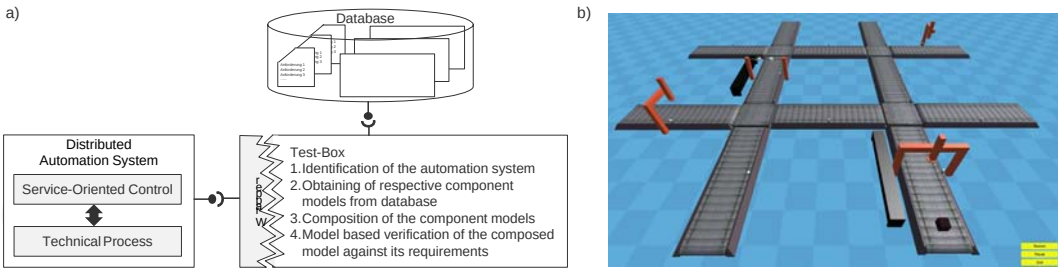

Fig. 6. a) Test-Box: Overview of the structure and the process to build up and verify a system model against its requirements b) Screenshot of the technical process realized by a virtual reality simulation

6. Conclusion

This paper presents a concept on how to ease the creation process of system models by modularization and partial automation of the modeling process. These system models are necessary to verify the behaviour of the overall system with help of model checking. Open nets, a modeling technique based on petri nets which were defined for service-oriented networks in the field of informatics, are extended to adapt to the requirements of industrial automation systems. It is shown that the composed system model still has the same structure as a typical petri net. Summarized the approach has following important properties:

- The complexity of the whole system is reduced by modularization.
- The functionality of a component can be modeled independently from other components.
- High Usability by reducing the modeling technique to attributes which are needed to model the ansynchronous communication of a service-oriented architecture.
- The models of standard components can be reused.
- The well-defined interface description (semantically described) of services is used to automate the composition process.
- The system model can be composed automatically by the component models
- Adaption of the system model when new components are integrated can be automated.
- A typical petri net structure makes it suitable as input for conventional verification tools.
- The models of the components are required for this approach.
- The approach can just verify logical, discrete behavior.
- The model of the technical process is assumed to be given. Just the interfaces to / from the technical process are regarded.

A possible application of the modeling techniques is presented. Thereby a test box integrates into the automation system, reads the system information, collects the deducted models, builds up the system model and verifies it against its requirements. In further research, it shall be shown that the modeling techniques can be extended with further attributes to suit more complex contexts. This just has to be bought at the cost of higher complexity of the model. To show this, the modeling technique shall is applied to a distributed automation system consisting of 153 services. In future research this modeling technique shall be transferred to further automation systems.

References

[1] S. Kowalewski, S. Engell, J. Preussig, O. Stursberg, Verifcation of logic controllers for continuous plants using timed condition/event-system models, Automatica 35 (1999).
[2] B. Schlich, J. Brauer, J. Wernerus, S. Kowalewski, Direct Model Checking of PLC Programs in IL, 2nd IFAC Workshop on Dependable Control of Discrete Systems (DCDS'09), Bari, Italy (2009).
[3] M. Khalgui, NCES-based modelling and CTL-based verification of reconfigurable embedded control systems, Computers in Industry 61 (2010).
[4] B. Vogel-Heuser, J. Folmer, G. Frey, L. Liu, H. Hermanns, A. Hartmanns, Modeling of Networked Automation Systems for Simulation and Model Checking of Time Behavior, 9th International Multi-Conference on Systems Signals and Devices, Chemnitz, Germany (2012)
[5] B. Vogel-Heuser, A. Fay, I. Schaefer, M. Tichy, Evolution of software in automated production systems: Challenges and research directions, The Journal of Systems and Software 110 (2015).
[6] A. Fay, B. Vogel-Heuser, T. Frank, K. Eckert, T. Hadlich, C. Diedrich, Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns, The Journal of Systems and Software 101 (2015).
[7] C. Gerber, S. Preusse, H.M. Hanisch, A complete framework for controller verification in manufacturing, Emerging Technologies and Factory Automation (ETFA) (2010).
[8] S. Roesch, S. Ulewicz, J. Provost, B. Vogel-Heuser, Review of Model-Based Testing Approaches in Production Automation and Adjacent DomainsCurrent Challenges and Research Gaps, Journal of Software Engineering and Applications, (08/2015) 499-519.
[9] J. Krause, Testfallgenerierung aus modellbasierten Systemspezifikationen auf Basis von Petrinetzentfaltungen, Dissertation, Shaker Verlag Aachen (2012).
[10] A. Zeller, M. Weyrich, Challenges for Functional Testing of reconfigurable Production Systems, 21st IEEE International Conference on Emerging Technologies and Factory Automation (2016).
[11] https://www.3ds.com/products-services/catia/products/dymola, retrieved on:2017.10.03.
[12] http://www.uppaal.org, retrieved on:2017.10.03.