Verifying Functional and Non-Functional Properties of Manufacturing Control Systems

Sebastian Preuße
University of Halle-Wittenberg
Institute of Computer Science
Chair of Automation Technology
Halle, Germany
Email: sebastian.preusse@informatik.uni-halle.de

Hans-Michael Hanisch
University of Halle-Wittenberg
Institute of Computer Science
Chair of Automation Technology
Halle, Germany
Email: hans-michael.hanisch@informatik.uni-halle.de

***Abstract*—Verification of control software is usually not applied in industrial practice because of additional work expenses and missing theoretical background that is necessary to apply this technique. Therefore, this contribution presents an integrated approach to verify functional and a subset of non-functional properties of manufacturing control systems. To support the user in creating a well-defined but also understandable specification of plant behavior, two approaches are introduced that specify functional requirements with Symbolic Timing Diagrams and non-functional ones with a Safety-Oriented Technical Language. These behavior descriptions are then translated to temporal logic formulas to perform model-checking of the closed-loop system of plant and controller.**

***Index Terms*—Closed-loop systems, Safety, Software verification and validation, Specification languages, Temporal Logic.**

I. Introduction

In the plant engineering domain, a paradigm shift is recognizable, since Model-Driven Engineering (MDE) approaches are increasingly applied to model continuous and discrete processes [1], [2]. For this, new plants are modeled and simulated long before they are constructed, and the control software is not implemented writing innumerable code lines anymore, but it is derived from software models automatically. While this procedure was an exclusive topic for academia a few years ago, it has now found its way into the industrial practice as well [3]. However, plant and controller models offer far more possibilities than just simulating the controlled process. If they are based on formal models, they further facilitate the verification of the control software. While simulating, certain test cases are executed to check for the correct plant operation. This procedure is not complete, since rare but critical scenarios might be overseen, what rather affects the system's safety. Because of this, simulation is a powerful tool to evaluate the correct function of the control software, but it does not provide a final assertion about its correctness. In contrast, verification considers every possible state of the controlled system and for this, it provides a machine-based proof of correctness. To do so, a specification of the requested behavior has to be defined and applied to the whole state space. A technical specification already contains the description of the requested behavior. However, it is usually written in natural text and for this possibly ambiguous and not well-defined. As the specification has mandatory to be formal to perform model-checking, new challenges arise, as description techniques come along with complex theory and formalisms. To apply verification, temporal logics have been approved as a powerful tool. However, the specification with these formulas is not trivial. To support the user in developing a formal specification, this contribution presents two approaches to create temporal logic formulas out of understandable text-based and graphical description methods. It is part of a simulation and verification framework and structured as follows.

In Section II, some related work concerning specification approaches is listed. Then, the framework of the contribution is described in Section III. Afterwards, the formal specification of plant behavior is described in Section IV. Section V gives an overview, how the specification is applied to perform verification. In Section VI, a case study shows the application of the approach and finally, the contribution is concluded in Section VII.

II. Related Work

The approach presented in this contribution is part of a framework for simulation and verification of manufacturing control systems that is presented in Section III. For this, the authors would like to refer to [4] and [5] for more information on Software- (SIL) and Hardware-In-The-Loop (HIL) Simulation and Verification. This contribution is focused on the specification of plant behavior to check for functional and non-functional requirements. The specification is developed with intuitive description methods and translated to temporal logic formulas automatically. For this, in the following some related approaches are listed that are concerned with the formal specification of plant behavior. Since the amount of specification techniques would fill several books, the survey shall rather give an idea about existing possibilities.

Specification methods can be grouped into text-based and graphical ones. Because of their structure, text-based methods are suitable for non-functional requirements, whereas graphical ones are usable for functional requirements. First of all, text-based approaches shall be considered. The authors of [6] use a subset of the natural English language and grammar. The user develops a specification by using a software tool,

which translates the properties into temporal logic formulas. Unambiguous expressions are detected and corrected in dialog with this tool. However, an expression in natural language has many possibilities of interpretation, so a lot of iteration steps could be necessary to come to a well-defined specification.
A further example for using natural English language, is introduced in [7]. The approach provides text-blocks, which are linked together to expressions. These expressions are translated to formulas of the Clocked Computation Tree Logic (CCTL). The method produces rather long expressions, the more complex a property gets, and this complicates the readability.
A Safety-Oriented Technical Language (SOTL) is introduced in [8]. The authors specify requirements on control software of Programmable Logic Controllers (PLCs) by using 18 sentences in the form of fixed frames. These frames are completed with specific phrases, which correspond to variables and values. This specific view on PLC software points out a disadvantage when specifying plant behavior because the execution behavior of the applied controller shall not be considered. For this reason, the original SOTL is modified in [9] by the authors of this contribution to describe the required behavior. In Section IV, more information is provided.
A graphical approach is presented in [10]. Here, Timing Diagrams (TD) are used to specify production sequences. A Timing Diagram Editor is implemented and presented in [11] to translate the diagrams to Timed Computation Tree Logic (TCTL) formulas. This temporal logic allows combining the qualitative temporal assertions together with real-time constraints, what is important for a specification of dynamic behavior.
In [12], the author introduces Symbolic Timing Diagrams (STDs). They are basically used in hardware development to describe reactive systems [13]. Due to their structure, STD are applied in [14] by the authors of this contribution to be used for specifying plant behavior as well. More information on this topic is provided in Section IV.
All the approaches mentioned in this section support the user in developing a meaningful specification in a temporal logic. Although the idea of transcribing an intuitive specification to a formal description of behavior is common, the applied techniques differ very much from each other. Especially the term ”intuitive” might be confusing, since this evaluation is related to the personal experience of the user. In Section IV, the authors of this contribution present two specification approaches, which are considered to be understandable for engineers, who specify the behavior of a technical plant. Beforehand, the framework of this contribution is described in the next section.

III. Framework

This contribution presents an approach to create a formal specification of plant behavior. It factors out the complex theory of temporal logics and provides a means to specify well-defined on the one hand but also understandable on the other hand. It is part of an integrated framework for simulation

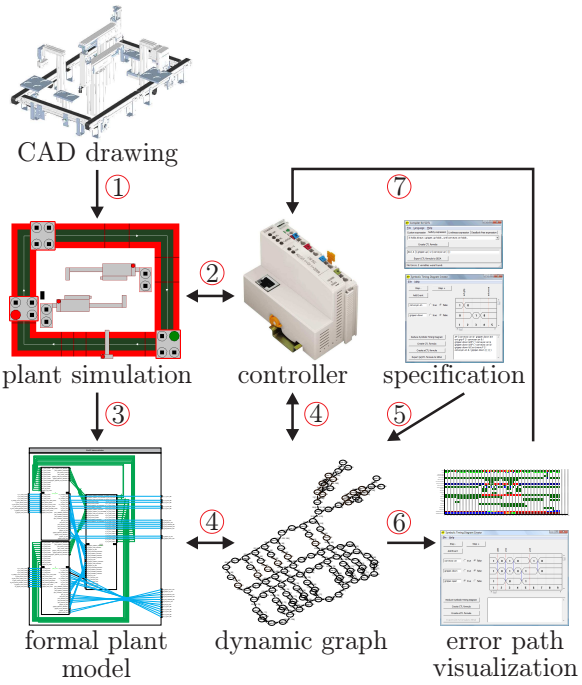

Fig. 1. Simulation and verification framework.

and verification of manufacturing control systems [4]. To give an overview, a survey of the framework is described in this section. A more comprehensive description is provided in [5]. Based on the technical specification, a new plant is designed using a Computer Aided Design (CAD) tool to plan the necessary construction steps and material requisitions. This CAD drawing, which is shown for the EnAS demonstrator (see Section VI) at the upper left side of Figure 1, is the starting point for the presented framework, since it is further usable to simulate the plant. Naturally, a CAD drawing contains the static plant structure but not its dynamic. In the scope of this work, the commercial simulation software *Incontrol Enterprise Dynamcis* is applied. The software imports the structural CAD data ① and assist in defining the moving plant parts to develop the uncontrolled plant simulation. This simulation is suitable to check the physical plant behavior to exclude collisions of moving parts or to check whether the specified production scenarios can be executed. To do the latter, the controller is interconnected with it ② to run a HIL or SIL Simulation. For the HIL Simulation the control code is executed on the target controller. In contrast, the software runs on a PC, when performing the SIL Simulation.
A remarkable disadvantage of simulation is the restriction to certain test cases. Even though they cover many failure scenarios, they do not consider all possible plant states. This is possible only by performing verification. For this, the plant simulation is translated to a formal plant model ③. In this work, Timed Net Condition/Event Systems (TNCES) [15] are applied as modeling formalism. The translation process is widely automated and supported by a software tool [5]. This tool analyses the CAD data and maps the different plant modules of the simulation to formal plant TNCE Modules,

which have to be manually created for each plant part. Once generated, they are reusable for other plants and for this, a library of frequently used plant modules can be established to support the translation process. Like for simulation, there are two possibilities to verify the controlled system. For HIL Verification, the controller is connected to the formal plant model, and a reachability analysis is performed ④. In contrast, for SIL Verification, the control code is translated to a formal controller model [4]. This controller model is composed with the formal plant model to a closed-loop system model, and again a reachability analysis is performed ④.
The computed dynamic graph contains every possible state of the closed loop of plant model and controller (model). If it is completed or a certain final state is reached, a formal specification of behavior will be applied to it ⑤ to do model-checking. The specification is provided in form of temporal logic formulas and checks functional and non-functional properties. If a property is not fulfilled, a counterexample is generated that shows the trajectory through the state space to the error state. This error path is visualized ⑥, so that the control code can be adapted ⑦ to run the next iteration step.
The framework is tailored to verify the correctness of the control software according to a specification of behavior. The formal plant models are generated semi-automatically, so that the engineer, who applies the framework, does not have to be familiar with formal modeling in detail. However, the approach will be applicable only if meaningful plant models are provided, so it is mandatory that the CAD data (i.e. the static model) and consequently the plant simulation (i.e. the dynamic model) are correct. The framework provides information, where errors in the control code can be found, but it does not generate control code. For this, it is independent from the hardware vendor, and it is applicable for control applications following the IEC 61131-3 as well as for those ones following the IEC 61499-1.

IV. Specification

A specification in general is a description of a product, a system or a service. Its aim is to define characteristics and attributes and to explain functionalities. A specification of a technical system shall be well-defined and formal. This mathematical basis allows analyzing the system with well-established methods and provides an unambiguous description of the system. However, a specification is non-executable because it describes what the system does but not how necessarily.
There are mainly two methods to specify requirements, namely text-based and graphical ones. Text-based techniques are suitable especially to describe non-functional requirements, i.e. properties of a plant such as safety, liveliness or absence of deadlocks. In contrast, graphical techniques are capable to describe functional requirements, i.e. production sequences of a plant. In the following, two approaches to specify non-functional as well as functional requirements are presented. Both deal as a front-end interface that supports the user in describing the requested plant behavior formally without having to be familiar with the theory in background. The formal basis of the presented approaches allows translating the specification to temporal logic formulas that are further applied for the verification.

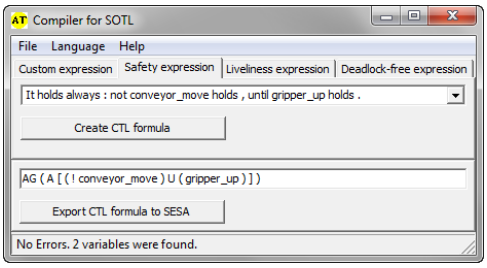

Fig. 2. Compiler for SOTL.

A. Safety-Oriented Technical Language

As described in Section II, there are numerous text-based specification approaches. To be of practical use, the applied approach shall be understandable in the environment of natural language and it should not require additional expertise. On the other hand, it has to be formal to be further applied in the verification framework. The SOTL presented in this contribution claims to fulfill these requirements. It consists of 21 pattern phrases that can be nested arbitrarily. The formal grammar enables parsing and translating the expressions to Computation Tree Logic (CTL) formulas [16]. To do so, a software tool was implemented. This *Compiler for SOTL*, shown in Figure 2, supports the user in describing non-functional properties. Furthermore, the software checks the semantic and helps to correct errors within the SOTL expressions. In Section VI, some examples according to the application are provided.
As a formal back-end, CTL is chosen to keep SOTL as simple as possible. This temporal logic offers enough expressive power to describe most non-functional requirements and it is suitable to specify discrete manufacturing control processes as proposed in this contribution.
Text-based approaches have a specific weakness, since they become hard to understand the more complex the specified requirements get. For this, they are not adequate to describe production sequences or functional requirements, respectively.

B. Symbolic Timing Diagrams

Graphical specification techniques will be applied if a lot of information shall be displayed in a well-arranged way. To describe complex program sequences and functional requirements, respectively, STDs are applied in this contribution. These diagrams do not require further expertise, since they are easy to read and to understand. Timing Diagrams can be used to schedule processes and for this, they are common to engineers, who specify (production) sequences. Beyond this, their formal basis allows deriving temporal logic formulas. The diagrams are created with a software tool, namely the *Symbolic Timing Diagram Editor*, shown in Figure 3. In addition to the version presented in [14], the software now supports the specification of state transitions and discrete time

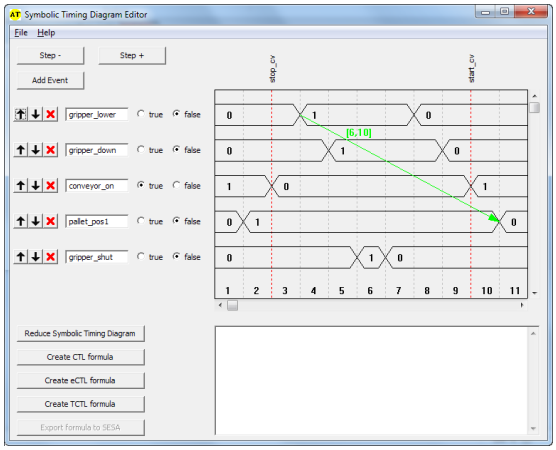

Fig. 3. Symbolic Timing Diagram Editor.

intervals as well. To describe a functional requirement, the user first of all specifies the necessary variables. The sequence is then developed in discrete steps, each one representing one system state. State transitions can be specified with events and interrelated with temporal constraints. The formal basis of STD allows to translate the diagram to a corresponding CTL, extended CTL (eCTL) [16], or Timed CTL (TCTL) formula [16] automatically. The next section demonstrates, how a formal specification is applied to verify a closed-loop system.

V. Verification

The framework presented in this contribution examines the closed-loop verification of manufacturing control systems. In contrast, just considering the controller on its own and drawing conclusions according to the functional operation in connection with a plant would be a doubtful activity. The controller rather needs feedback from the controlled process to verify its functionality. To ease the formal plant model generation, a corresponding TNCE Model is derived from the plant simulation automatically. To verify the control software, there are two possible procedures, namely HIL Verification and SIL Verification.

To apply HIL Verification, the implemented control software runs on the target controller (see ④ in Fig. 1) that is interconnected with the formal plant model running on a PC. To do so, a software tool was implemented that captures the whole dynamic graph of the system under control (cf. Section VI. The computation will be completed if a certain final state is reached or if the dynamic graph is completed, i.e. if all possible states are found. The specification of functional and non-functional requirements is then applied to the dynamic graph to perform model-checking (see ⑤ in Fig. 1). Finally, the user will get the assertion *TRUE* if the specification is fulfilled, or *FALSE* if it is violated. In this case, a counter example is produced that visualizes the trajectory through the graph to the faulty state (see ⑥ in Fig. 1). With this information, the control software can be adapted to start over the procedure. HIL Verification is of advantage if the target hardware is already determined. For the controller, it makes no difference if it controls the plant

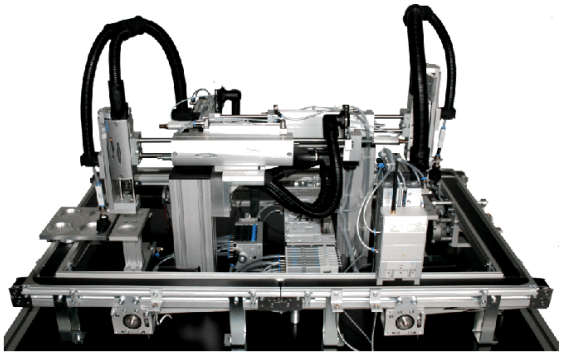

Fig. 4. EnAS demonstrator.

or its model and for this, the software is verified for ”real conditions”.

In contrast, the control software has to be translated to a formal TNCE Model, when performing SIL Verification. This task is automatable as described in [4]. Controller and plant model are composed to a closed-loop system model and for this, the dynamic graph is calculated by determining all possible states. Subsequently, the specification is applied to the graph as already described for the HIL Verification. Although the control software has to be translated to a formal model, the procedure has some benefits. As both models run on a PC, no additional hardware is required to connect controller and plant. Furthermore, the PC does not have to wait for the controller to provide new output information and for this, the computation of the dynamic graph is faster.

Both procedures consider the whole state space of the system under control. For this, the specified requirements are checked for every possible state of the dynamic graph. It is a crucial point that the applied models are correct, otherwise model-checking delivers incorrect assertions. Because of this, the different steps of the framework of this contribution are widely-automated. In the next section, an example shall demonstrate the application.

VI. Case Study

The applicability of the framework was tested using a manufacturing system in lab-scale, namely the EnAS demonstrator shown in Figure 4. Information to the plant is provided in [15]. As depicted in Figure 1, the CAD data of the plant is transferred to a plant simulation ①, and from this simulation, the formal plant model is derived ③. This TNCE Model of the plant forms the basis for HIL and SIL Verification as described in Section V. The model generation is widely automated to prevent overcharging the user with theory and to exclude additional sources of error. To perform model-checking, a specification of the required behavior has to be provided. As demonstrated in Section IV, this work is supported by software tools. In the following, some examples according to the specification of non-functional and functional requirements are shown.

In this work *non-functional requirements* are considered that specify safety, liveliness and the absence of deadlocks. Safety properties specify requested or forbidden behavior, e.g. that

it never occurs that the gripper of the demonstrator and the conveyor are both moving in one state (see Formula 1). Liveliness requirements specify desired behavior, e.g. if the conveyor starts moving, the pallet will reach position 1 finally (see Formula 2). The absence of deadlocks checks, whether a possible action can be executed in at least one successive step, e.g. if the pallet reaches position 2, the gripper will be lifted down (see Formula 3). The examples show that SOTL phrases would be understandable in a technical specification, since they are close to the technical language engineers apply. In addition, they have a formal grammar, so that CTL formulas can be derived. As the SOTL phrases can be nested, there is no restriction according to the length of an expression. However, they get confusing, the more variables and relationships are defined.

For this, *functional requirements* are described with a graphical approach. The STD, shown in Figure 5 specifies the gripper sequence. The corresponding variables are the actuators *gripper_lower* and *conveyor_on* as well as the sensors *conveyor_pos1* and *conveyor_pos2*. The sequence starts with all variables on false. After the event *ev_start* fires, the conveyor starts moving. If the pallet on it reaches the first position, the conveyor will be stopped. After the event *ev_lower* fires, the gripper goes down and up again. Then, the pallet moves to the second position and the gripper is lowered and raised again. Finally, the conveyor transports the pallet to the next procession station and both sensors are deactivated. An STD does not describe a fixed sequence but rather checks, whether each following specified state is fulfilled in at least one future system state. Anything that happens in between two steps is not specified and for this not important for the functional requirement. The user can refine the diagram arbitrarily, and it gets the more detailed, the more information is provided. Formula 4 displays a cutout of the eCTL formula derived from the STD in Figure 5. AF characterizes that the sequence has to occur in a future state, whereas EF connects the specific states, i.e. 1 to 14, of the diagram. Since an eCTL formula is derived, events are considered as a transition from one state to another. This information is inserted between the EF operator, e.g. in E ev_start F. Figure 6 shows the TNCE Module of a cylinder. As mentioned above, the formal modules are once created and stored to a library. The translation software analyses the simulation model and maps the plant components to the corresponding formal modules. The derived plant model

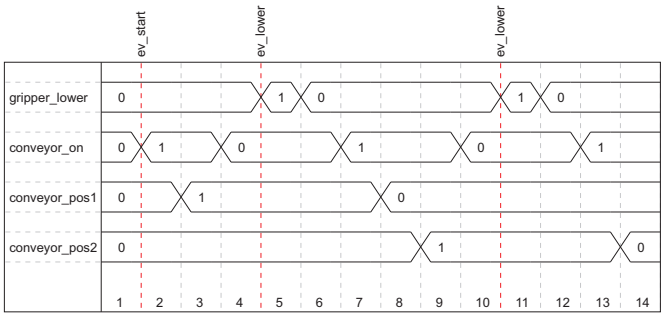

Fig. 5. Gripper sequence specified with an STD.

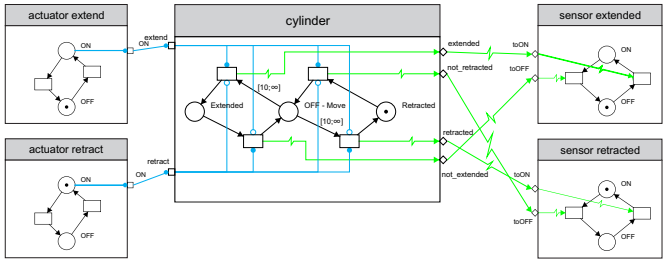

Fig. 6. TNCES plant model example.

has got the same interface as the real plant, i.e. sensors and actuators and for HIL Verification, the hardware controller is connected through its inputs and outputs to this formal plant model, so that the state space of the closed-loop system can be captured. The controller is seen as a black box that accepts input values and provides output values and for this, the controller behavior can be determined even if the source code is not supplied. The procedure start after initialization of the controller. A software reads out the controller output values and transfers them to the corresponding plant input modules of the formal plant model. This marks the initial state for the dynamic graph computation. The calculation will be interrupted if the value of at least one plant output module changes. The software then transfers the new information back to the controller inputs. The controller executes its program and delivers new output values, so that the procedure can start over again.

The computed dynamic graph usually contains branches and loops, since the controlled system runs production cycles. At a branching point, the controller state is recorded and after finishing one possible trajectory, the controller is reset to its initial state, and then driven to the branching point. Afterwards, the other possible trajectory is determined. Considering the controller as a black box is actually a partial verification because the internal states of the controller are not taken into account. However, this information is necessary to get a final assertion about the correctness of the controller because the values of internal variables, counters and flags have to be consistent at the different branching points. Implementing a complete verification is part of future work to improve the framework of this contribution. For this, the commercial tool AutoSPy of GWT-TUD will be applied. In contrast to HIL Verification, the whole state space of the closed-loop is computable, when performing SIL Verication [4].

The specification of non-functional and functional requirements is applied to the computed dynamic graph in a model-checking tool. Actually, the verification step is the crucial one within the framework of this contribution. Complex models lead to large dynamic graphs, since the number of possible states rises exponentially. This state space explosion problem shall not be underestimated, as it might be a time-intensive task even for modern computers. In case of a violated specification, model-checking delivers an error path that represents the sequence of states within the dynamic graph that leads to this failure state. This path is visualized in a Gantt Chart or

in an STD (see ⑥ in Fig. 1). The graphical information is then used to adapt the faulty control software. This procedure is performed iteratively until the specification holds for the closed-loop control system.

SOTL: It holds never: gripper_lower and conveyor_on.

CTL: AG ! (gripper_lower & conveyor_on) (1)

SOTL: It holds always: If conveyor_on holds, then it holds finally: conveyor_pos1.

CTL: AG (conveyor_on $\rightarrow$ AF conveyor_pos1) (2)

SOTL: It holds always: If conveyor_pos2 holds, then it holds next on one path: gripper_lower.

CTL: AG (conveyor_pos2 $\rightarrow$ EX $gripper\_lower$) (3)

CTL: AF (! gripper_lower & ! conveyor_on & ! conveyor_pos1 & ! conveyor_pos2 & E ev_start F (! gripper_lower & conveyor_on & ! conveyor_pos1 & ! conveyor_pos2 & $EF(\ldots)$)) (4)

VII. Conclusion

Verification of industrial controllers is not used in real industrial applications because of additional work expenses and the complex theory that has to be handled when performing formal analysis. However, this procedure can contribute to the safety of control systems, since it delivers reliable assertions about the correctness of control software. Therefore, this contribution presents a framework for the verification of functional and non-functional requirements of manufacturing control systems that is fully integrated into the usual engineering environment to develop manufacturing systems. The work of this contribution is focused especially on the specification of the corresponding plant properties. For this, two approaches are introduced that support engineers in creating a well-defined and formal specification, namely the STDs for characterizing functional requirements, as well as the SOTL for describing non-functional ones. The approaches claim to be understandable in daily engineering practice, whereas they do not overcharge the user with theory. A case study has shown that they can be provided as a technical plant specification and beyond this, they are further applicable for the verification of the control software.

The claim of the verification framework is an easy application, so that it can be integrated into daily practice of plant engineering. Because of this, the different steps of the framework have been automated widely, so that modeling errors are minimized and the user does not need to know all the theory in background. Although it might be extended, the framework does not generate control code automatically. It rather assists to migrate existing controllers and to check manually-developed control code. For this, it is addressed to practical engineers, who do usually not like the idea of automatically-generated control code. The control code itself is not touched, but the framework rather gives information about errors within the code. Beyond this, the framework does not depend on any hardware vendor because it is applicable for any controller.

The development of the framework is still ongoing. In future works timing aspects as well as optimization of the dynamic graph calculation will be investigated. Furthermore, monitoring possibilities of the hardware controller shall be improved to ease the integration of the approach.

Acknowledgment

This work is funded by the Federal Ministry of Economics and Technology (BMWi) under reference 16 IN 0651 on account of a decision of the German Bundestag. The authors are responsible for the contents of this contribution.

References

[1] T. Lukman, G. Godena, J. Gray, and S. Strmcnik, "Model-Driven Engineering of Industrial Process Control Applications," in *15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Bilbao, Spain, 2010, index: MF-001333.

[2] I. Hegny, M. Wenger, and A. Zoitl, "IEC 61499 based Simulation Framework for Model-Driven Production Systems Development," in *15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Bilbao, Spain, 2010, index: MF-003565.

[3] P. Wallner, "Highest Profit: Automatische Code-Generierung," *openautomation REPORT*, pp. 22–25, 2010.

[4] C. Gerber, S. Preuße, and H.-M. Hanisch, "A Complete Framework for Controller Verification in Manufacturing," in *15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Bilbao, Spain, 2010, index: MF-001279.

[5] S. Preuße, C. Gerber, and H.-M. Hanisch, "Virtual Start-Up of Plants using Formal Methods," *Int. J. of Modelling, Identification and Control (IJMIC)*, 2010, submitted for publication.

[6] A. Holt and E. Klein, "A semantically-derived subset of English for hardware verification," in *Meeting of the Association for Computational Linguistics (ACL)*, 1999, pp. 451–456.

[7] S. Flake, W. Müller, and J. Ruf, "Structured English for Model Checking Specification," in *Workshop of the Association for Computer Science (Gesellschaft für Informatik (GI))*. Berlin: VDE Verlag, 2000, pp. 91–100.

[8] M. Heiner, T. Mertke, and P. Deussen, "A Safety-Oriented Technical Language for the Requirement Specification in Control Engineering," in *Computer Science Reports 09/01*, BTU Cottbus, May 2001, p. 65ff.

[9] S. Preuße and H.-M. Hanisch, "Specification of Technical Plant Behavior with a Safety-Oriented Technical Language," in *7th IEEE International Conference on Industrial Informatics (INDIN)*, Cardiff, United Kingdom, June 2009, pp. 632–637.

[10] H.-M. Hanisch and V. Vyatkin, "Application of Visual Specifications for Verification of Distributed Controllers," in *IEEE Conference on Systems, Man and Cybernetics (SMC)*, Tucson, Ariz, USA, October 2001, pp. 646–651.

[11] G. Bouzon, V. Vyatkin, and H.-M. Hanisch, "Timing Diagram Specifications in Modular Modeling of Industrial Automation Systems," in *16th IFAC World Congress*, Prague, Czech Republic, July 2005, pp. 208–221.

[12] R. Schlör, "Symbolic Timing Diagrams: A Visual Formalism for Model Verification," Ph.D. dissertation, Fachbereich Informatik, Carl-von-Ossietzky Universität Oldenburg, 2001.

[13] R. Schlör and W. Damm, "Specification and Verification of System-Level Hardware Designs using Timing Diagrams," in *Proceedings of the European Conference on Design Automation*. IEEE Computer Society Press, 1993, pp. 518–524.

[14] S. Preuße and H.-M. Hanisch, "Specification and Verification of Technical Plant Behavior with Symbolic Timing Diagrams," in *3rd International Design & Test Workshop (IDT)*, Monastir, Tunisia, December 2008, pp. 313–318.

[15] C. Gerber, "Implementation and Verification of Distributed Control Systems," Ph.D. dissertation, Martin-Luther-University, Halle (Saale), Germany, 2010.

[16] P. Starke and S. Roch, "Analysing Signal-Net Systems," *Informatikberichte, Humboldt-Universität zu Berlin*, vol. 162, September 2002.