Specification of Technical Plant Behavior with a Safety-Oriented Technical Language

Sebastian Preuße*, Hans-Michael Hanisch*
*University of Halle-Wittenberg, Institute of Computer Science, Chair for Automation Technology,
Kurt-Mothes-Straße 1, 06120 Halle, Germany
Telephone: ++49 (345) 55-25975, Fax: ++49 (345) 55-27304, Email: sebastian.preusse@informatik.uni-halle.de

***Abstract*—Manual reconfiguration of existing systems is a common task for engineers. Especially, the modernization of plants depends on an unambiguous and reliable documentation. Furthermore, the well-defined specification of plant behavior enables the engineer to verify the control software with a model-checking tool. In order to be able to provide the description correctly and without misunderstandings, it should be done formally. Temporal logics, like the Computation Tree Logic, offer an ideal solution for this purpose. However, they are based on a complex theory. In order to use them anyway in practice, the formal expressions have to be developed by automated procedures out of an intuitively understandable description. This contribution proposes a text-block based method, which enables the user to create such a formal specification by the use of a Safety-Oriented Technical Language.**

I. Introduction

The work described in this contribution is part of a collaborative project that is carried out by a consortium of several companies related to semiconductor manufacturing and two universities. The goal is to speed up and ease the reconfiguration of manufacturing systems by applying model-based design technologies. Information can be found at [1].

It is obvious that any reconfiguration of a manufacturing system results in a reconfiguration of the specification of its behavior. At the current state of the project, this is done manually by an engineer, but the work we present here has the potential to be performed automatically in future.

During reconfiguration process, it is of advantage to test and verify the control software in combination with a plant model. For this, a specification of plant behavior has to be provided. This description should be formal, so that model-checking for the verification of the model is possible. Temporal logics, like the Computation Tree Logic (CTL) [2], offer an ideal tool for this purpose. They provide the descriptive power to describe state properties and state sequences and consequently to describe discrete plant behavior. However, they are based on a complex theory, which is often considered as hard to understand. The specification normally comes from an engineer who is an expert in manufacturing, but not in the field of formal modeling or model-checking. Thus, he has to be supported in specifying plant behavior even if he does not know the theory in detail.

This contribution proposes a text-block-based method for specifying demands on plant behavior as well as test cases for the verification of models. The main objective is to provide an intuitive specification method, which is close to the technical language an engineer would use in practice. It shall be understandable without previous knowledge on the one hand, but also well-defined with a formal basis on the other hand. The specification is developed with a Safety-Oriented Technical Language (SOTL) in a software environment, namely the *Compiler for SOTL*, which automatically derives CTL formulas.

Net-Conditon/Event-System (NCES) [3] has been developed in our group as a tool for modular and compositional modeling of large discrete event systems. It has been successfully applied to modeling of manufacturing plants as well as of controllers during more than one decade. It is tailored to the needs of engineers because it supplies real modularity, object orientation (to some extent) and a concept of signals as usual in control engineering. We therefore use it for modeling of closed-loop behavior.

Specifications are usually describing either forbidden or desired behavior of the plant, not of the controller. The controller is just a means to ensure that specified behavior. So, the motivation is how to combine a graphical method for systems modeling with a text-block-based method for behavioral specification to be used by engineers and at the same time to provide the capabilities of formal model-checking. For this, we want to stay as close as possible to the original model (i.e. NCES). This will significantly support debugging if a specification is not fulfilled. Therefore, the proposed method does not focus on development of formal methods of model-checking, but on the application within a control engineering framework. This marks the very clear difference between other work coming from computer science (e.g. [4]–[6]) and the presented approach for engineering. This issue also justifies the use of the modeling and specification methods.

The contribution is structured as follows. Section II gives an overview about specification requirements and possibilities to develop descriptions of plant behavior. Section III depicts some related work of other work groups concerning the specification with text-block-based methods. Section IV introduces our SOTL and presents the *Compiler for SOTL*. In Section V, a NCES plant model of a cutout of our EnAS demonstrator [1] is created to show the practical application. In Section VI, the specification of process requirements is developed and applied to the model of the closed loop in Section VII. Section VIII concludes our work and lists some future work.

978-1-4244-3760-3/09/$25.00

II. Specification Requirements and Possibilities

Temporal logics are suitable to describe discrete time lapse and they provide the possibility to create a reasonable and well-defined specification. This description of plant behavior is summarized under the synonym of process requirements, which are divided into two categories. On the one hand, safety requirements examine plant behavior, which is absolutely necessary for a failure-free operation. The constructs contain a small number of variables and are formulated precisely. On the other hand, production requirements check liveliness, the absence of deadlocks and complex production sequences. They examine whether the actual production process corresponds to the required one.

There are different possibilities to create specifications for technical plants [7]. Using natural language is the most intuitive variant, though a translation to a non-ambiguous specification is not possible without restrictions. In contrast, a standard language has a clear vocabulary and a formally defined grammar. It can be transferred to a formal specification by translation rules. A semi-formal description possesses a defined syntax, but no mathematical basis. In contrast, the formal specification is based on a well-defined syntax and semantic. In this contribution, a possibility for the text-block-based description of technical plant behavior is presented. The specification is created using the SOTL. The formal grammar of this standard language facilitates the transfer of the text-block-based description to CTL expressions.

III. Related Work

The specification of plant behavior plays an important role especially while designing or modernizing a plant. To exclude misunderstandings because of ambiguous expressions, it is helpful to create this description with well-defined methods. Furthermore, a formal specification is essential for the verification of models of reactive systems. Specifying with natural language would be a very intuitive possibility, but it is rather imprecise because of subjective styles of writing. Standard languages solve this problem by providing a fixed dictionary and a definite grammar. Due to their well-defined basis, they allow the automatic translation of text-blocks into formal constructs like temporal logic formulas.

A lot of work groups deal with the translation of standard languages to formal expressions. In [4], the authors use a subset of the natural English language. The user enters the specification and receives temporal logic formulas. The software helps to find and to remove ambiguous expressions iteratively. The method is very comfortable because it supports the user in creating a well-defined specification. However, a lot of iteration steps could be necessary to receive a result.

A further approach for the translation of natural English language is shown in [5]. The user combines text blocks to a property specification and a software translates it to Clocked Computation Tree Logic (CCTL) [8] formulas. The method produces long expressions the more complex the properties become. This disadvantage is illustrated in [6] and the authors try to solve this problem by advancing the Object Constraint Language (OCL), which is part of the Unified Modeling Language (UML), to the Real Time-OCL (RT-OCL). The specification in RT-OCL can be translated automatically to CCTL formulas. The approach provides more precise expressions, but anyway the user is confronted with an abstract language, which has first to be learnt.

In [9], a SOTL is presented. 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. The language is very suitable to specify the behavior of software of IEC 61131 conform PLCs because it includes the characteristic cyclic execution behavior of a PLC. However, it is too specialized to describe the behavior of technical plants in general because this shall be possible without dealing with certain properties of the applied controller. For this reason, the idea of our approach is to introduce a modified SOTL that describes plant behavior without concerning the details of the controller.

IV. Saftey-Oriented Technical Language

The modified SOTL presented in this contribution specifies plant behavior and does not consider the execution behavior of the applied controller. More precisely, we focus on the description of discrete states, in which the plant shall be or not. The general idea about this proceeding is that an engineer is able to use the language for describing process requirements without having to know the details of software and hardware issues except the physical components of the plant. However, he develops a formal specification of plant behavior, which can be used in following steps to e.g. verify the controller software with model-checking tools. Furthermore, he provides a well-defined and intuitive description of process requirements, which can serve as a technical documentation. This is rather important especially for manual reconfiguration because it is significant to have a solid basis when modernizing a plant. Formulas 1 to 21 show the translations rules to derive a CTL formula from a SOTL expression. The temporal terms are to be interpreted as stated below:

- We will use *simultaneously* if the values of variables are considered within one time step. This is expressed by conjunction ($\wedge$) and disjunction ($\vee$) respectively.
- If a property holds *always*, it will be valid in every state. This is expressed by the linear operator G.
- *Next* refers to the immediately following state. This is marked by the linear operator X.
- We will use *finally* if a requirement is fulfilled in a future state. Information about the concrete state is not supplied. This is expressed by the linear operator F.
- A condition holds *until* another condition is fulfilled. As a demand, the second condition has to be true necessarily in a future step. This is shown by the linear operator U.
- A condition holds *before* another condition is fulfilled. As a demand, the first condition has to be true necessarily in at least one previous step. This is shown by the linear operator B.

The modal parameters are to be interpreted as shown below:

- A condition will hold necessarily if it is fulfilled on every path. This is described by the path quantifier A.
- A condition will hold possibly if it fulfilled on at least one path. This is expressed by the path quantifier E.
- If a sufficient condition holds, the subsequent condition will hold as well. This implication is expressed by the symbol $\rightarrow$.

In the following, the different SOTL constructs are grouped regarding to their temporal parameter. The text-blocks on the left side are converted to the CTL formulas on the right side. If the expressions are nested, the different units within the SOTL constructs will be separated by a coma. Respectively, the units in a nested CTL formula are marked by brackets. This proceeding is demonstrated in an example later on. If A and B as well as φ and ψ represent basic plant variables, it will hold: A $= \varphi$ and B $= \psi$. Otherwise, if they represent nested expressions, they will have to be translated recursively and it will hold: A $\rightarrow \varphi$ and B $\rightarrow \psi$.

Simultaneously:

A and B. $\Longleftrightarrow \varphi \wedge \psi$ (1)

A or B. $\Longleftrightarrow \varphi \vee \psi$ (2)

Always:

It holds always: A. $\Longleftrightarrow AG\ \varphi$ (3)

It holds never: A. $\Longleftrightarrow AG\ !\ \varphi$ (4)

It holds possibly always: A. $\Longleftrightarrow EG\ \varphi$ (5)

Next:

It holds next: A. $\Longleftrightarrow AX\ \varphi$ (6)

It holds possibly next: A. $\Longleftrightarrow EX\ \varphi$ (7)

Finally:

It holds finally: A. $\Longleftrightarrow AF\ \varphi$ (8)

It holds possibly finally: A. $\Longleftrightarrow EF\ \varphi$ (9)

Until:

A holds, until B holds. $\Longleftrightarrow A\ [\varphi\ U\ \psi]$ (10)

A holds possibly, until B holds. $\Longleftrightarrow E\ [\varphi\ U\ \psi]$ (11)

Before:

A holds, before B holds. $\Longleftrightarrow A\ [\varphi\ B\ \psi]$ (12)

A holds possibly, before B holds. $\Longleftrightarrow E\ [\varphi\ B\ \psi]$ (13)

Implication:

If A holds, then it holds simultaneously: B. $\Longleftrightarrow \varphi \rightarrow \psi$ (14)

If A holds, then it holds always: B. $\Longleftrightarrow \varphi \rightarrow AG\ \psi$ (15)

If A holds, then it holds never: B. $\Longleftrightarrow \varphi \rightarrow AG\ !\ \psi$ (16)

If A holds, then it holds next: B. $\Longleftrightarrow \varphi \rightarrow AX\ \psi$ (17)

If A holds, then it holds finally: B. $\Longleftrightarrow \varphi \rightarrow AF\ \psi$ (18)

If A holds, then it holds possibly always: B. $\Longleftrightarrow \varphi \rightarrow EG\ \psi$ (19)

If A holds, then it holds possibly next: B. $\Longleftrightarrow \varphi \rightarrow EX\ \psi$ (20)

If A holds, then it holds possibly finally: B. $\Longleftrightarrow \varphi \rightarrow EF\ \psi$ (21)

Figure 1 shows the graphical user interface of the *Compiler for SOTL*. The software was programmed in C++ with the graphics package wxWidgets. The included compiler was implemented in a C environment with the help of the tools Flex and Bison. The software features four different flags. *Custom expression* includes the 21 possible pattern phrases shown in formula 1 to 21. The other flags contain specific phrases for *safety expressions*, *liveliness expressions* and *deadlock-free expressions*. The Patterns are chosen via a drop down menu and just serve as assistance while developing the specification. The user is free to enter any custom expression. The complier checks whether the input is valid and derives and displays a CTL expression, when clicking on *Create CTL formula*. In case of error detection, the software puts out a message at which position within the SOTL expression the error can be found. Finally, there is the possibility to export the CTL formula to a file format, which is readable by the model-checking tool *Signal/Events Systems Analyzer* (SESA) [2]. The language of the compiler is switchable, so that in the current version the whole functionality is available in English and in German.

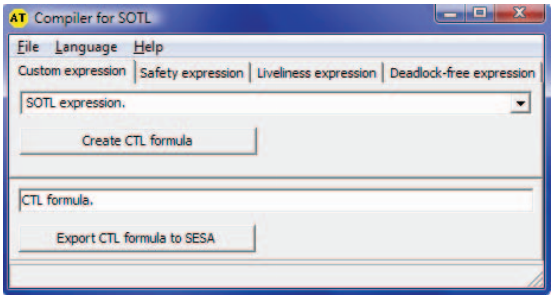

Fig. 1. Compiler for SOTL

V. Plant Model

An example shall demonstrate the application of the compiler. For this, a cutout of the EnAS demonstrator of the Chair for Automation Technology is focused. In figure 2, the vacuum sucker ① and the slide ② of the demonstrator are shown. The sucker can move vertically up and down and possesses a binary sensor for the upper position. The slide moves horizontally to charging position 1 and 2

Fig. 2. Cutout of the EnAS demonstrator

and possesses a binary sensor for charging position 2. It contains slots for two work pieces. The task of the vacuum sucker is to pick up a work piece from the slide and to depose it into a tin. For more information about the functionality in detail, please visit our website [1]. In initial position, the sucker is retracted and the slide is in charging position 1.

Figure 3 shows the NCES, which models the plant cutout. In [10] it is described, how hierarchical controllers for distributed control systems are created and verified. The model was implemented with the *(T)NCES Editor* of the Chair for Automatic Technology. It consists of the control module *control*, the actuator modules *slide_move* and *sucker_lower*, the sensor modules *slide_pos2* and *sucker_up* and the plant modules *up_down* and *pos1_pos2*. The structure of the modules is described briefly below.

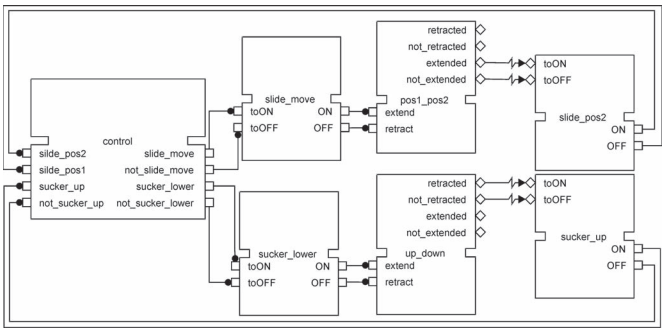

Fig. 3. NCES plant model

A. Actuator Module

Figure 4 shows the net within an actuator module. State information of the control module is transfered through the condition inputs *toON* and *toOFF* to the internal net. Similarly, state information of the internal net is transfered through the condition outputs *ON* and *OFF* to the plant module. An actuator can be in state *ON* or *OFF*. The information about the states of the actuator module is transferred by condition connections to the plant module because the physical component in this example is a pneumatic cylinder. The air pressure, which is necessary for pushing out, must be built up. The necessary time cannot be neglected and so the state transition of the actuator module and the movement of the cylinder do not happen synchronously.

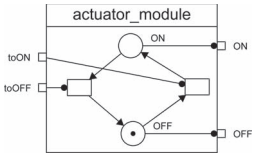

Fig. 4. Actuator module

B. Sensor Module

Figure 5 shows the net within a sensor module. State transition information of the plant model is transfered through the event inputs *toON* and *toOFF* to the internal net. Similarly, state information of the internal net is transfered through the condition outputs *ON* and *OFF* to the control module. A binary sensor can be in state *ON* or *OFF*. The state transition information of the plant module is transferred with event connections to the sensor module because the physical component is a electrical sensors. When a cylinder reaches the end position, the signal will be transferred without relevant time delay.

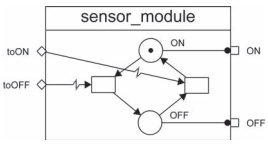

Fig. 5. Sensor module

C. Plant Module

Figure 6 shows the net within the module, which models the physical behavior of the vacuum sucker in connection with sensors and actuators. The cylinder depicted by the model can be in state *Extended*, *Retracted* or in intermediate state *OFF-Move*. State information of the actuator module is transfered through the condition inputs *extend* and *retract* to the internal net. Similarly, state transition information of the internal net is transfered through the event outputs *not_retracted*, *retracted*, *not_extended* and *extended* to the event inputs of sensor module. As the physical component of the slide is a pneumatic cylinder as well, the plant module for the slide looks the same.

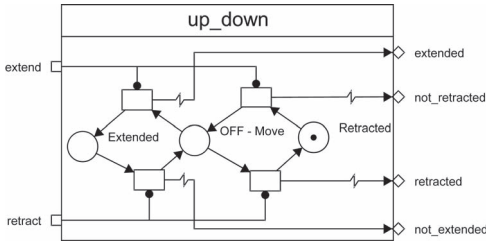

Fig. 6. Plant module

D. Control Module

Figure 7 shows the model of the controller. In initial position, the slide is in charging position 1 and the vacuum

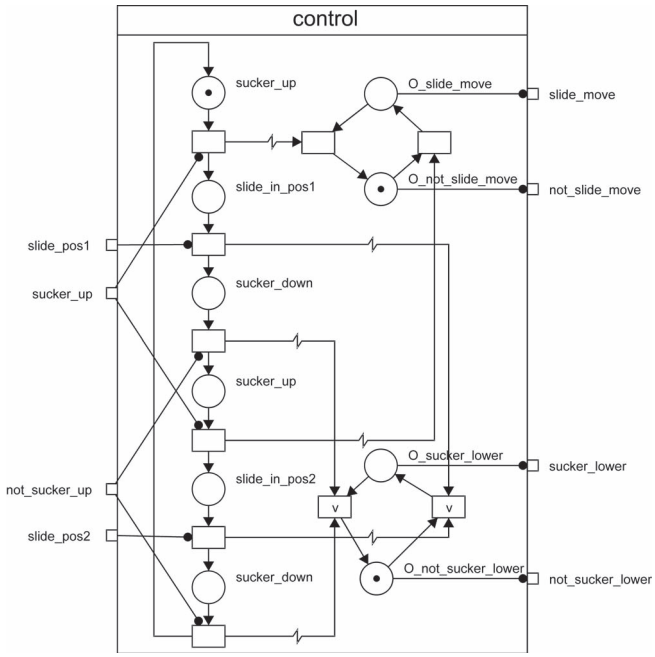

Fig. 7. Control module

sucker is retracted. In the 1^{st} step, the slide will move to charging position 1 if it is not already there. In step 2 the sucker moves down and in step 3 it moves up again. In the 4^{th} step, the slide moves to charging position 2 and in step 5 and 6 the sucker moves down and up again. Afterwards, the sequence starts again from the beginning.

VI. Specification of Plant Behavior

After having implemented the NCES model, it shall be verified with a model-checking tool. For this, a formal specification of plant behavior is necessary, so that formal test cases can be applied to the model. The development of the specification shall be demonstrated in this section.

A formal specification can be described with temporal logic formulas because they offer the expressive power for this task. We focus on CTL, which is suitable for the specification of discrete plant behavior. The use of CTL does not complicate the structure of the text-blocks, what makes it easier to apply them for a technical documentation. For the description of dynamic behavior, some possibilities shall be discussed in the conclusion.

Since temporal logics are based on a complex theory, the user has to be supported during the development process. For this reason, the description of plant behavior is implemented with SOTL. Because of the formal basis, the text-blocks can be translated automatically to CTL formulas by the *Compiler for SOTL*. The procedure is described in the following at the example of the EnAS demonstrator.

A. Safety Requirements

The plant model considers the vacuum sucker and the slide of the demonstrator. We will now apply safety requirements to the model to check that the plant does not execute forbidden behavior. The according CTL formulas only contain the operators AG, AX, or AU. Expressions that contain the path quantifier E or the temporal operator F are not permitted because they do not provide unambiguous assertions concerning the safety. The slide may only move if the vacuum sucker is retracted. The following two expressions in SOTL make sure that the slide stays in one charging position when the sucker is moved, until it is in the upper position again, what is signalized by the position sensor.

SOTL: It holds always: If not slide_move and sucker_lower, holds, then it holds simultaneously: not slide_move holds, until sucker_up holds.

CTL: $AG\ ((!\ \text{slide\_move} \wedge \text{sucker\_lower}) \rightarrow A\ [\text{slide\_move}\ U\ \text{sucker\_up}])$ (22)

SOTL: It holds always: If slide_move and sucker_lower, holds, then it holds simultaneously: not slide_move holds, until sucker_up holds.

CTL: $AG\ ((\text{slide\_move} \wedge \text{sucker\_lower}) \rightarrow A\ [\text{slide\_move}\ U\ \text{sucker\_up}])$ (23)

B. Productuion Requirements

Production requirements are liveliness expressions, deadlock-free expressions and complex production sequences, which have not been checked yet with safety requirements. Liveliness checks, whether desired events occur. The CTL expressions always contain the operator AF. The following expressions check liveliness of the actuators slide_move and sucker_lower and they check, whether the sensors sucker_up and slide_pos2 are activated correctly, when the corresponding actuator performs an action.

SOTL: It holds always: If not slide_move holds, then it holds finally: slive_move.

CTL: $AG\ (!\ \text{slide\_move} \rightarrow AF\ (\text{slide\_move}))$ (24)

SOTL: It holds always: If not sucker_lower holds, then it holds finally: sucker_lower.

CTL: $AG\ (!\ \text{sucker\_lower} \rightarrow AF\ (\text{sucker\_lower}))$ (25)

SOTL: It holds always: If not sucker_lower holds, then it holds finally: sucker_up.

CTL: $AG\ (!\ \text{sucker\_lower} \rightarrow AF\ (\text{sucker\_up}))$ (26)

SOTL: It holds always: If slide_move holds, then it holds finally: slide_pos2.

CTL: $AG\ (\text{slide\_move} \rightarrow AF\ (\text{slide\_pos2}))$ (27)

Deadlock-free expressions check, whether a possible action can be executed in at least one successive step. The CTL formulas always contain the operator EX. Since there are no possible deadlock-free expressions for our example, the principally structure of such an expressions is given below.

SOTL: It holds always: If φ holds, then it holds possibly next: ψ.

CTL: $AG\ (\varphi \rightarrow EX\ \psi)$ (28)

Last but not least, production sequences shall be checked. The structure of the CTL formulas is not subject to any restrictions. The application of text-block-based specification techniques shows some disadvantages because the text-blocks become quite confusing the more complex the properties get. To solve this problem, alternative specification methods like graphical procedures could be applied. This shall be discussion in the conclusion. The sequence shown below checks, whether the vacuum sucker is extended and retracted, after the slide has moved to charging position 2. A second sequence would look similarly for changing position 1.

SOTL: It holds finally: not slide_move and it holds possibly next: sucker_lower and it holds possibly next : not sucker_lower,.

CTL: $AF\ (!\ slide\_move \wedge EX\ (sucker\_lower \wedge EX\ (!\ sucker\_lower)))$ (29)

In this section, the classification of different requirements was explained. It is not the aim to find every possible SOTL

expression for a complete specification of behavior, but to check rare and often critical test cases. Due to the formal basis of SOTL, the implemented compiler produces CTL formulas, which can now be applied to the model.

VII. Model-Checking

Verification of a model provides a mathematical proof of correctness in contrast to simulation. The NCES model is finally to be verified with the model-checking tool *SESA*. For this, the model is transferred to a Signal-Net System (SNS) by vertical and horizontal composition with the tool *MOSAIC*. The application of *SESA* is described in detail in [2]. We use SNS, *MOSAIC* and *SESA* only as tools to test our specification procedure in practice. For more information please refer to the literature. In connection with the specification of behavior in CTL, a proof of correctness of the closed-loop model can be obtained. *SESA* calculates a reachability graph as long as a statement about correctness of the model can be met. If an error is found, the path that leads to the state, which does not fulfill the specification, will be put out so that the model can be adapted manually with this information. For our example, *SESA* supplies the statement that the model fulfills the specification of behavior. Thus, it was verified for the process requirements and consequently for the specification.

VIII. Conclusion

In this contribution, a text-block-based specification procedure was presented. The Safety-Oriented Technical Language can be used as an intuitive interface to create a specification of technical plant behavior. As discussed in a previous section, the formal description supports the manual reconfiguration of manufacturing systems because it serves as a well-defined technical documentation and enables the formal verification. Both is essential while modernizing existing systems. The presented language is suitable to specify safety requirements as well as production requirements. Our approach eases the creation of formal specifications with temporal logics because the user does not have to be familiar with the complex theory of e.g. CTL. To support the user in developing the formal constructs, the *Compiler for SOTL* was implemented. This software tool contains SOTL patterns, which are preformulated for different requirements, but it also allows the creation of custom SOTL expressions. The compiler derives CTL formulas automatically and passes them to the model-checking tool *SESA*.

Our contribution also depicts some disadvantages of our approach. The SOTL expressions could become confusing the more complex the requirements get. This is not a problem while specifying safety requirements, but as shown in Section VI-B, production sequences will produce rather long sentences. To handle this problem, we developed the description of sequences with a graphical specification method, namely the Symbolic Timing Diagrams (STD). The results are summarized in [11]. The user is free to choose the most intuitive one, since both approaches have certain pros and cons. The combination of the two composes a framework for the specification of any process requirement.

The current program version of the *Compiler for SOTL* only supports CTL formulas because they are powerful enough to describe discrete safety and production requirements. However, the use of CCTL, Timed Computation Tree Logic (TCTL) [12] or extended Computation Tree Logic (eCTL) [2] for the specification of dynamic or event-driven behavior is possible as well. To implement these features, we have to think about the basic structure of SOTL because additional information would complicate the expressions. These questions shall be answered in continuative studies.

Acknowledgment

This work was 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] "Institute of Computer Science, Chair for Automation Technology," April 2009. [Online]. Available: http://aut.informatik.uni-halle.de/

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

[3] J. Thieme, "Symbolische Erreichbarkeitsanalyse und automatische Implementierung strukturierter, zeitbewerteter Steuerungsmodelle," Ph.D. dissertation, Mathematisch-Naturwissenschaftlich-Technische Fakultät (Ingenieurwissenschaftlicher Bereich) der Martin-Luther-Universität Halle-Wittenberg, 2002.

[4] 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.

[5] 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.

[6] S. Flake, W. Müller, U. Pape, and J. Ruf, "Specification and Formal Verification of Temporal Properties of Production Automation Systems," in *SoftSpez Final Report*, ser. Lecture Notes in Computer Science, H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, and E. Westkämper, Eds., vol. 3147. Springer, 2004, pp. 206–226.

[7] F. Bitsch, "Verfahren zur Spezifikation funktionaler Sicherheitsanforderungen für Automatisierungssysteme in Temporallogik," Ph.D. dissertation, Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart, Institut für Automatisierungs- und Softwaretechnik, 2006.

[8] J. Ruf and T. Kropf, "Modeling and Checking Networks of Communicating Real-Time Systems," in *Correct Hardware Design and Verification Methods (CHARME)*, IFIP WG 10.5. Springer, September 1999, pp. 265–279.

[9] 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, Mai 2001, p. 65ff.

[10] D. Missal, M. Hirsch, and H.-M. Hanisch, "Hierarchical Distributed Controllers - Design and Verification," in *Conference on Emerging Technologies and Factory Automation (ETFA)*, Patras, Greece, 2007, pp. 657–664.

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

[12] E. Emerson, A. Mok, A. Sistla, and J. Srinivana, "Quantitative temporal reasoning," *Real-Time Systems*, vol. 4, no. 4, pp. 331–352, December 1992.