Virtual Start-Up of Plants using Formal Methods

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

Christian Gerber
University of Halle-Wittenberg, Germany
Institute of Computer Science, Chair of Automation Technology
Kurt-Mothes-Straße 1, 06120 Halle (Saale), Germany
Phone: ++49 (345) 55-25974, Fax: ++49 (345) 55-27304,
Email: christian.gerber@informatik.uni-halle.de

Hans-Michael Hanisch
University of Halle-Wittenberg, Germany
Institute of Computer Science, Chair of Automation Technology
Kurt-Mothes-Straße 1, 06120 Halle (Saale), Germany
Phone: ++49 (345) 55-25970, Fax: ++49 (345) 55-27304,
Email: hans-michael.hanisch@informatik.uni-halle.de

Abstract: Start-up of a manufacturing system is a time-consuming task because control software is usually designed detached from the real plant or a simulation model. Even the most experienced software engineer is not able to consider every possible scenario without having feedback from a plant or its model. So, undesired behavior can occur after transferring software to the controller. At best, the errors are recognized just after starting-up, so that the software can accordingly be corrected. But there can be even worse failures that occur rarely and randomly, but are the more critical and mean danger for human and machine. This problem is of high significance because automated systems get more and more complex, especially if they control parallel operations. Hence, control software bugs are preassigned and should be considered in early project phase. The approach presents an integrated framework that facilitates virtual start-up of a plant. For this, formal methods are applied, so that control software cannot only be simulated but also verified before start-up. This eases control software design and reduces plant downtimes and consequently costs.

Keywords: (Formal) Modeling; Net Condition/Event Systems; Plant Behavior; Specification; (Virtual) Start-Up; Test Case Generation; Verification.


Biographical notes: S. Preuße is a graduate engineer. He has been a member of the scientific staff of the Chair of Automation Technology at the Martin Luther University Halle-Wittenberg in Germany since 2008. His main research topics are specification of plant behavior and verification of control software.

C. Gerber is a graduate engineer. He has been a member of the scientific staff of the Chair of Automation Technology at the Martin Luther University Halle-Wittenberg in Germany since 2006. His main research topics are design and implementation of heterogeneous and distributed control systems.

H.-M. Hanisch is Full Professor and Head of the Chair of Automation Technology at the Martin Luther University Halle-Wittenberg in Germany. His main research topics are (amongst others) methods for modeling, synthesis and verification of discrete and hybrid systems and distributed systems.
1 Introduction

The work described in this contribution is part of a collaborative project that is carried out by a consortium of several companies and two universities. The fields of interest reach from automation technology to systems simulation and systems engineering. The goal is to speed up and ease the start-up of manufacturing systems as well as the migration of already existing plants.

The authors are particularly interested in applying formal methods to make virtual start-up of plants more efficient. Virtual start-up is performed as Hardware in the Loop (HiL) test. For this, the hardware controller is attached to a software model of the plant that has to be controlled. This test allows engineers to check functionalities of their implemented control software without having to interrupt the operation of the real plant and furthermore, it can even be performed before the construction of the manufacturing system is finished.

Simulation and verification especially of control software gain rising interest. The more complex systems get, the more sources of errors can possibly lead to unexpected behavior while implementing control software. These errors are not released by the environment - what marks the difference of this approach to those dealing with fault tolerant systems like for example (Rajakarunakaran et al., 2008; Soliman et al., 2008; Wang and Duan, 2008) - but by faulty control software code. Although, many errors can be corrected by just reviewing the code there can still remain bugs that effect critical states, which are dangerous for the plant and its environment.

To avoid such errors, it is essential to test software during the development process. Since this process is a cooperation between engineers of different disciplines, interfaces have to be clearly defined, and a fundamental procedure should be developed, so that every team member can benefit from the ideas of each other.

In contrast to automobile industry, where HiL as well as Software in the Loop (SiL) simulations are long established, those methods are usually not applied in plant engineering. One particular reason is the higher number of units that are produced in automobile industry, what justifies the development of a model in parallel to the product. The additional work and money expenses will be marginal in contrast to the added costs if a new car type has malfunctions. For this, plant engineers have to be convinced to apply those simulation techniques as well.

This approach is addressed to the specific demands of plant engineering. It proposes a framework, which serves as a guideline to ease virtual start-up of plants using formal methods and is therefore structured as follows.

In Section 2, preliminaries are given to describe some basic theory. Section 3 introduces the framework of the OMSIS project and points out specific tasks of the workgroup. In Section 4, a manufacturing prototype is introduced, which serves as a demonstrator for showing the practical application of the contribution. Beyond this, the development of formal plant models is depicted. Afterwards, Section 5 discusses the specification of test cases. In Section 6, a methodology for deriving formal plant models out of 3D simulation models is presented. These formal models are connected to the control hardware in Section 7 for an applied controller test. In Section 8, controller verification is performed by applying the test case specification to the closed-loop model and finally, the contribution is concluded in Section 9 and an outlook for future work is given.
extent) and a concept of signals as usual in control engineering. Therefore, it is used for modeling of closed-loop behavior. Section 4.2 introduces briefly, how formal models can be created with NCES.

2.2 Temporal Logics

Temporal logics were first introduced in (Pnueli, 1977). Since then, they have been successfully applied in computer science to describe behavior of discrete systems. They are especially used for specification and verification of computer systems. As discrete plant behavior is considered in this approach, temporal logics offer a great tool to create formal and well-defined specifications, which can further be used for formal analysis of systems.

In this contribution, two different subcategories of temporal logics are considered. In the following, they are shortly introduced.

2.2.1 Computation Tree Logic

The Computation Tree Logic (CTL) (Starke and Roch, 2002) is a branching time logic. A CTL formula considers paths as a sequence of states in which a system can be. A valid expression is always a combination of a path quantifier and a linear operator. A path quantifier expresses if the statement holds on every path (A) or at least on one path (E). The linear operators define if the expression holds in all states (G), in a future state (F) or in the subsequent state (X). Furthermore, an expression can hold until a second one holds (U) (under condition that the second condition holds in at least one future step) or it can hold before a second one holds (B) (under condition that the first condition holds in at least one previous step).

Figure 2 visualizes the CTL formula \( AG(\varphi \rightarrow AX \psi) \). It expresses the following. It holds globally in all states if \( \varphi \) is true then \( \psi \) is always true in the very next state. To check a safety requirement with a CTL formula, one can test the following property. If the gripper of a system moves down the conveyor will always be in stop state. The corresponding variables are \( \text{conveyor} \_\text{move} \) and \( \text{gripper} \_\text{lower} \). The CTL formula is shown in Formula 1.

\[
AG(\text{gripper}\_\text{lower} \rightarrow \text{NOT conveyor}\_\text{move}) \tag{1}
\]

2.2.2 Extended Computation Tree Logic

An extension of CTL was introduced in (Roch, 2000). This extended Computation Tree Logic (eCTL) enlarges the expressive capability of CTL. It focuses not only on description of states but also on description of state transitions. This is particularly important, while specifying event-driven behavior. Figure 3 shows an exemplified eCTL formula. It specifies that \( \varphi \) is true in a future state on at least one path. On this specific path, every state transition has to fulfill \( \tau \). The formula holds true for the third and forth path from the left only.

A functional requirement that can be checked with an eCTL formula can look like this. It always holds in the next state that after the start event has been received, the gripper is retracted and the conveyor is moving. In contrast to CTL, it is possible to consider the state transition information, namely the start event with eCTL. The resulting eCTL formula is shown in Formula 2.

\[
A \tau_{\text{start}} X(\text{NOT gripper}\_\text{lower} \& \text{conveyor}\_\text{move}) \tag{2}
\]

3 Framework

The joint research project OMSIS\(^1\) is aimed to ease start-up and migration of a plant. One major point within this project is the elimination of control software errors before starting-up. This means, it is not considered to monitor or to detect faults at run time because formal methods usually require much computation power and it is fairly hard to apply them online. So, the analysis of control software is completed before applying it to the real plant. This might sound as a restriction, but if formal methods prove the software to be correct, it will only have to be adjusted if the plant configuration changes. However, this procedure only works if the plant model is correct and contains as much details as required.

In the following, the framework for virtual start-up of plants is demonstrated.

The state of the art is such that automation solutions are usually tailored according to a specific order of a customer. Because of this, it is necessary to develop the framework to be reusable, so that it can be used for
general application. Nevertheless, simulation techniques are often not applied because they mean investment costs and increase the final price of a project. However, this calculation does not consider the added costs of maintaining and adjusting control software after having started up the plant. So, it is present situation that control software is programmed without having feedback from a model of a system. The problem appears after transferring this software to the controller and starting-up the plant. There can be lots of unexpected reactions, which have to be analyzed and fixed. This procedure is time- and money-consuming, and of course it means danger for human and plant. Especially, rare but critical behavior is not immediately recognized but hours or days after start-up. This scenario can be avoided by testing the control software in connection with a plant model. Unfortunately, this procedure does not cover every malfunction because only manually chosen scenarios are considered.

The authors’ approach tries to go one step further. The simulated plant is translated semi-automatically to a formal model. This model can be analyzed by calculating a reachability graph of all possible states. In combination with a formal specification of functional and non-functional requirements, correct behavior can be proven. The procedure is depicted in Figure 4. The controller is connected to the formal model, which runs on a PC. In this contribution, a Siemens Programmable Logic Controller (PLC) is used. Anyway, there is no restriction to the applied control hardware because a controller just serves as a device to ensure that the implemented behavior will be executed. That means in particular, only sensor and actuator signals, which are transferred from and to the formal model, are of interest and it makes no difference, what kind of controller is applied.

The values of the control outputs are transferred to the model (3). This marks the initial state for computing the reachability graph (2). The computation is executed until the value of at least one control input changes in a state. The state is mapped back to the formal model (3) and the resulting input values are transferred to the controller (4). The control software reacts according to the new information and generates new output information. Then, the procedure starts again.

The calculation of the state space is finished if the whole graph is computed or if a certain final state is reached. This of course belongs to the plant model and the corresponding process, i.e. if it is cyclic or linear. Admitting that the results are trivial if everything works fine, it shall be emphasized that the approach detects errors, which are not obvious by just observing the controlled plant, i.e. unexpected behavior that happens rarely but that is the more critical because it has not been considered while programming.

Those possible error trajectories within the reachability graph can be visualized in a Gantt Chart for example (5). With this information, the control software can be corrected (6). As already stated, this procedure is performed before start-up of the real plant. Bugs that are detected thereby, can be eliminated before leading to danger. Besides the benefits of a simulation for plant planning and construction, it can be used for formal analysis in an early project phase as well. The additional work and expenses will be compensated at the latest when control software acts as specified while launching the plant.

4 Plant

4.1 Plant Example

The framework presented in this contribution shall be demonstrated using the example of a manufacturing prototype. The demonstrator in Figure 5 is part of the EnAS project, which deals with energy self-sufficient actuators and sensors.

The testbed consists of two identical plant modules, which are rotated by 180° to each other. Figure 6 shows a CAD drawing of the plant from the bird’s eye view. Conveyor (1) marks the wait position and has one light barrier. It has no mounted processing station, so it can be used as a pallet buffer. Conveyor (2) delivers pallets
The horizontal middle position can only be reached if the sucker of the Jack Station can move to a high and a low position, the pallet or back onto the tin again. To do this, the Jack Station can open a tin and deposit its lid onto the tin or to the Jack Station and vice versa. Furthermore, it can lift and deposit the pallets to and from the pallets.

To emphasize the effort of using the Energy-Autarkic Actuator and Sensor System, only the right plant module is equipped with it. The left one is classically wired. The conveyors 1, 2, 3 of both modules form a circuit and transport pallets cyclically to each processing station in the clockwise direction.

Each conveyor is moved by a separate drive and all six conveyors transport pallets with tins and workpieces inside from one processing station to another. Equipped with light barriers, the tins on the pallets are correctly positioned in front of the stations. The first conveyor 1 transports a pallet to the first light barrier, i.e., the wait position. After the light barrier is activated, the controller device switches off the drive and the pallet is stopped. If the drive is started again, the pallet will move on and will disable the light barrier after a certain time. The second conveyor 2 brings the pallet by means of two light barriers to the correct position in front of the Jack Station 3. The first photo sensor indicates the first loading position, and the second one marks the second loading position. The third conveyor 3 transfers the pallets to two positions in front of the Gripper Station 5. The first position determines the correct positioning of the first tin. The second tin will be positioned if the second sensor switches on. After activating the drive of the last conveyor 3, the two light barriers will be disabled after a certain time. The pallet will move to the wait position of the following plant module if the drive of its first conveyor 1 is switched on.

The Jack Station puts workpieces from the Slide Station to the tins on the pallet or vice versa. Furthermore, it can open a tin and deposit its lid onto the pallet or back onto the tin again. To do this, the sucker of the Jack Station can move to a high and a low position as well as to three different horizontal positions. The horizontal middle position can only be reached if the Jack Station extends an additional Jack Stick. By means of vacuum, the lid or the workpiece is sucked on and safely lifted when the sucker moves up. Through the combination of these possibilities, the tins can be loaded and unloaded, respectively.

The main function of the Gripper Station is to close the lid of a tin, which was previously loaded by the Jack Station. Furthermore, a tin can be lifted and deposited onto another pallet. Through compressed air, the gripper is moved up and down and activates the sensors up and down thereby.

### 4.2 Formal Plant Modeling

The plant is the dominant part of a manufacturing system and the chosen controllers, either hierarchical, central or distributed, are only a means to realize the desired plant behavior. This also means that the design of a control system without a minimum knowledge of the physical plant behavior is a doubtful activity. Nevertheless, it should be mentioned that designing a model of the plant used for simulation or formal verification is a time-consuming task and therefore a cost factor that must not be underestimated. But having a look to the everyday engineering workflow, this closed loop is the best way to identify implementation errors in advance and therefore the plant will be brought into service faster.

Each manufacturing system consists of several mechatronic components such as conveyors, cylinders, heaters, tanks, storage, etc. as well as of digital and analogue sensors and actuators. Due to the use of each component over and over again in the same or other plants, a methodology of engineering plant models in a systematic way rather than designing them from scratch is helpful. Obviously, a modular or object-oriented approach is required to create formal or informal plant models to be applicable for an engineer. Each object or module should encapsulate the discrete and uncontrolled behavior of the represented mechatronic component. This means that every possible behavior has to be described at each physically possible state under each order of arbitrary assignment of the actuator state. As described in (Hanisch et al., 1999), NCE Modules fulfill these requirements.

During the last decade, a library of frequently used NCE Modules encapsulating the uncontrolled behavior of several mechatronic components was built up. The systematic modeling of the plant as described in (Vyatkin and Hanisch, 2003) is simplified by interconnecting these modules with event and condition arcs. The composition of large models from smaller ones is obvious to any engineer who has ever modeled a system in a block-diagram-oriented way. Nevertheless, this manual work shall be replaced during the ongoing OMSIS project. For this, plant models used for plant simulation are translated semi-automatically to formal NCES plant models. This procedure is shown in Section 6.
4.2.1 Composition of the whole Plant

The physical composition of the EnAS Demonstrator is depicted in Figure 7. The numbers on the arcs represent the set of components within the plant. For example, there is one EnAS Demonstrator that consists of two plant parts. Each plant part has one Jack Station, which consists of one sucker, five sensors of type 2, four valves and three cylinders.

Using a library of basic plant modules (Sec. 4.2.2), it is feasible to derive the hierarchical structure of the composed formal model from the structural description of the mechanical components in Figure 7. Thus, the NCE Module describing the uncontrolled behavior of the Gripper Station (Fig. 15) contains two basic modules of the type cylinder (Fig. 11 and 12), two of the type actuator (Fig. 8) and two of the type sensor (Fig. 9) as well as an additional local workpiece model. The Jack (Fig. 16) and the Slide Station use the same basic module types of a different quantity. The NCE Module describing the uncontrolled behavior of a conveyor uses instances of the actuator module (Fig. 8) to describe the relay and the drive as well as instances of the sensor module (Fig. 9) to model the conveyor and the mounted position sensors.

The composite NCE Modules describing the Gripper, the Jack and the Slide Station as well as the conveyors are combined to the NCE Module of the plant part. The two module instances of the plant part are extended by the workpiece model of the pallet to form the whole plant model.

In the following, the basic plant modules are introduced. The signal interconnection between the applied modules is exemplarily discussed for the Gripper and the Jack Station, afterwards.

4.2.2 Basic Plant Modules

The elementary actuators inside the EnAS testbed are valves, suckers, relays and drives with one rotating direction. All of them can be switched on and off to influence other actuators like cylinders or conveyors as well as several workpieces. The formal model in Figure 8 consists of a place invariant with the places ON and OFF. The condition input toON enables the transition connecting them. The model can be used for every simple actuator, which is not instantly switched on due to a mechanical connection. The condition output ON can be used to connect other plant parts as cylinders or drive models, which are not instantly influenced if the actuator state changes. The event outputs toON and toOFF shall be used to model interconnections to plant parts, which
are instantly influenced by direct mechanical interaction, e.g. a conveyor by a drive or a workpiece by a sucker.

The formal sensor model consists as the actuator model of a place invariant, but the state transition is enforced by input events. The model in Figure 9 cannot only be used for sensors, but also for all plant parts described by a place invariant that are instantly influenced by another module like a conveyor or a local workpiece behavior. The model in Figure 10 may be a sensor, which will only be switched on if two constraints are fulfilled. This is for example the case at the Jack Station of the testbed. Therein, two cylinders, namely the Jack Stick and the horizontal cylinder, have to be in a certain position to activate the sensor.

If the first constraint is fulfilled (\(\rightarrow{toON_1}\)) and the second cylinder reaches its end position (\(\rightarrow{toON_2}\)), the left transition will be enabled and the token goes to place ON. Otherwise, if the second constraint is fulfilled first (\(\rightarrow{toON_2}\)) and the first cylinder reaches its end position (\(\rightarrow{toON_1}\)), the other transition will be enabled.

All cylinders of the testbed are extended or retracted by compressed air. The flow into the cylinder will be enabled if the corresponding valve is opened and if it is closed again, the air will flow out. Thus, the cylinder will not instantly retract or extend if the valve is opened or closed because the change of pressure is time-dependent. Depending on the system pressure, the length, the diameter and the construction of tube and cylinder (i.e. the amount of tube connectors, corners, T and Y-fittings), the duration between opening or closing a valve and cylinder movement will never be instant. Consequently, the cylinder model can only be connected by condition arcs to other modules. In Figure 11, this condition signal enables the state transitions between Retracted, Move and Extended.

Discrete times can also be inserted into the model, so that it is extended to a Timed Net Condition/Event System (TNCES) (Karras, 2009). For considering time to leave the end positions, the model shown in Figure 12 is extended by four places \(*_{\text{reached}}\) and \(*_{\text{leave}}\) as well as two connecting transitions, whereas \* stands for Retracted and Extended, respectively. The places \* and \*_{\text{reached}} get a token after one step. If the condition signal changes, the transition between the place \* and OFF-Move will only be enabled if there is a token at the place \*_{\text{leave}}\*, which gets there from the place \*_{\text{reached}}. The condition that enables the transitions between the places \*_{\text{reached}}\*, OFF-Move and \*_{\text{reached}}, leave_{\text{*}}\* is identical. If now a discrete time is noted at the post arc of leave_{\text{*}}\*, the token will also stay at \*_{\text{reached}}\* for the modeled time even though the transition is condition enabled. An example is found within the UpDown cylinder module of Gripper and Jack Station in the following.

Figure 13 and 14 show the models of double-acting cylinders. In contrast to the single-acting cylinder, there are two chambers, which have to be filled and released.
with compressed air to move the cylinder. In case that both connected valves are opened, both chambers are filled and the cylinder will remain in its position until one valve is closed. It is also possible to introduce discrete times into the formal model as described previously.

4.2.3 Gripper Station

Figure 15 shows the NCE Module of the Gripper Station. Depending on the true and false state of the control output, the connected pneumatic valve switches to ON or OFF. This enables the flow of compressed air into or out of the corresponding cylinder, which extends or retracts. Doing so, the gripper moves up and down or is closed and opened again. According to the duration between the opening of the valve and the start of the cylinder movement, a modeling with condition arcs is performed. The up and down movement will turn the positioning sensors immediately to ON or OFF if the end positions are reached. If a closed gripper starts moving up (→ gripperUpDown_not_extended) or finishes its down moving (→ gripperUpDown_extended), a tin will immediately be taken from or deposited to the pallet. For modeling this synchronous behavior, event arcs are used. Despite this, a tin is only closed correctly after the gripper is closed for a certain period of time, which results in modeling the connection between the closing cylinder and the tin attribute by a condition arc (→ gripper_close_not_extended). The EnAS Demonstrator handles three different pallets transporting zero, one or two tins with different loading states. This workpiece behavior of the pallet is modeled in the same way as a sensor. Only one tin can be gripped at the same time, so it will be lost if the close cylinder starts opening (→ gripper_close_not_extended). It will be gripped if a pallet with a tin is positioned right and the cylinder UpDown starts retracting in combination with an extended close cylinder, i.e. if the gripper is closed. This decision is modeled within the global workpiece model of the pallets and will be true if an event is passed to the input wp_toON.

4.2.4 Jack Station

The Jack Station consists of 5 sensors, 4 valves, 3 cylinders and one sucker. Figure 16 shows the corresponding module. According to the explanations of the Gripper Station, the connections between the valves and cylinders are modeled by condition arcs and between the cylinders and sensors by event arcs. The sucker is modeled in the same way as a valve, but it will immediately suck in a workpiece and lose it if it is switched on (→ jack_sucker_toON) or off (→ jack_sucker_toOFF). Accordingly, the interconnection to the local workpiece behavior is modeled with event arcs.

4.2.5 Workpiece Behavior (Pallet)

Each workpiece has several properties that are modeled globally or locally. Since pallets are moving through the whole plant, the position modeling is done in a global module interacting with all plant parts. Each station model is extended by a local workpiece model to identify if something is sucked in, gripped or stored. The synchronization between local and global models is done by event arcs because if something is stored to the global workpiece model, it will no longer be available within the
local model. Each workpiece attribute is modeled with a separate NCE Module, describing all possible states. For the pallets, the actual position as well as the loading state of the two tins on it is considered.

5 Test Case Specification

Test cases describe functional and non-functional properties of the plant. They have to be seen detached from the controller because they specify the desired behavior without considering the control environment. Anyway, they can be used to test the control software as well if they are applied to the closed loop of controller and plant. In this section, methods to define test cases are introduced. The application is shown in the following ones.

A specification in general is a description of a product, a system or a service. Its aim is to define characteristics and attributes 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 necessarily how. The automatic transformation of a specification together with a plant model to an implementation is called synthesis. For more information to this topic, please refer to (Schewe, 2008; Missal and Hanisch, 2009).

The specification of plant behavior is the very first step when designing a technical plant. Based on this description, the plant itself is built and the control software is implemented. Usually, the design process is team work between engineers with different technical background. So, it is natural that the specification has to be well-defined on the one hand, but also understandable without restrictions on the other hand. This is essential for the exchange of ideas, the documentation, the implementation of control software, the subsequent modernization and migration and last, not least for the analysis of plant and control models.

Especially the last point, namely the verification of models, is very important according to safety because in contrast to simulation, verification considers every possible scenario and delivers a mathematical proof of correctness.

However, these formal methods require expertise, which is not self-evident in daily practice. So, there is a need for kind of front ends that enable engineers to create a formal specification of plant behavior without necessarily having to know all details about the underlying theory. This section lists specification techniques and gives an overview about different methods, which are applicable in practice to be formal on the one hand, but also understandable on the other hand.

5.1 Specification Requirements and Possibilities

The description of plant behavior is summarized under the synonym of process requirements, which are divided into two categories. On the one hand, safety or nonfunctional 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 or functional 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 in general four different possibilities to create specifications for technical plants (Bitsch, 2007). 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 translated 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 the following, specification techniques are presented. For this, some related work is listed and afterwards, two approaches, which were adapted for this work, are explained in detail.

5.2 Specification Techniques

5.2.1 Related Work

There are two main possibilities to specify plant behavior formally, namely text-based methods on the one hand and graphical ones on the other hand. While text-based approaches are applicable to describe short and precisely formulated properties, such as safety requirements, graphical techniques are suitable for describing even complex program sequences and functional requirements, respectively.

The approach presented in (Holt and Klein, 1999) utilizes 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, this procedure has a certain weakness. 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 the natural English language, was introduced in (Flake et al., 2000). 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 provided software tool Raven applies the specification to the model in the verification step.

The authors discussed limits of their approach in
Modelica is an object-oriented description language for physical models. It was standardized in 1997 and is available in version 3.1. Different graphical development environments are available, e.g. Dymola or MathModelica. The physical models are translated to mathematical models, so that they could be analyzed with a model-checking tool. The specification of model behavior and the subsequent verification is described in (Bunus et al., 2004). Since the tools around Modelica are integrated, there is no clear borderline between modeling and specification. However, the concept is suitable for having everything in one while developing and analyzing physical plant models.

A formal method for modeling and implementing cooperative multi-agent behavior is presented in (Risler and Stryk, 2008). The Extensible Agent Behavior Specification Language (XABSL) is an agent programming language, which is based on hierarchical finite state machines. The authors apply their approach to specify the execution behavior of robots. For this, the XABSL models are translated to meta code, which is interpreted by the XABSL execution engine. Both, the compiler for generating meta code and the C++ implementation of the interpreter are available for free at the project website. Although, the control software is synthesized from the execution behavior model, the specification is not used for formal model verification because the authors do not consider the closed loop but only the controller. Anyway, they investigate how behavior programming can benefit from the further use of formal methods, e.g. in order to prove certain properties of implemented behaviors.

An example for specifying graphically, was presented in (Vyatkin et al., 2000). The authors describe, how a specification of plant behavior could be formulated with the Timed Computation Tree Logic (TCTL). This temporal logic allows combining the qualitative temporal assertions together with real-time constraints, what is important for a specification of dynamic behavior. To support the user in creating such a description of plant behavior, a graphical specification method was proposed in (Hanisch and Vyatkin, 2001a). Here, Timing Diagrams (TD) are used to specify production sequences. A Timing Diagram Editor (TDE) was implemented and presented in (Bouzon et al., 2005) to translate the diagrams to TCTL formulas. This editor is part of the Verification Environment for Distributed Applications (VEDA) described in (Hanisch and Vyatkin, 2001b).

The UML offers a further graphical specification method. In (Missal et al., 2007a) activity diagrams are used to specify production sequences. The Systems Modeling Language (SysML) is an extension based on UML and provides description methods that are important for modeling of a system. There are essential diagram types available for the specification of plant behavior, namely activity diagrams, sequence diagrams, state machine diagrams and use case diagrams. For static descriptions, SysML offers diagram types like block diagrams, internal block diagrams and package diagrams. An outstanding difference between UML and SysML is the fact that SysML offers requirement diagrams. In these diagrams, textually formulated requirements of a system are represented graphically and their relationships among each other are shown. The clue is to assign each requirement to one or more according diagram/diagram component. By simulating the system, one automatically gets a simulation of the requirements. In (Hirsch and Hanisch, 2008) the application of the different diagram types and facts on simulation, which are provided by SysML, are described.

Message Sequence Charts (MSC) are related to sequence diagrams of UML. They are applied to describe communication sequences between interacting objects. In (Canver, 1999), MSCs are extended to MSC\textsubscript{CTL} because the application of classic MSCs is limited when specifying system properties. It is not clearly expressible on which condition a sequence diagram holds. For this, the authors added textual expressions to clarify the general requirement, when a sequence has to be interpreted. The CTL path quantors $A$ and $E$ define if a sequence holds at any time or if it is just one possible path. Furthermore, negation, conjunction and disjunction of MSC expressions are possible with the extension.

A further approach, extending MSCs was presented in (Damm and Harel, 2001). The authors added expressions that describe the changeability of a sequence. That means that for example it can be expressed if a sequence describes a possible scenario (existential) or if a sequence describes a scenario, which must hold (mandatory). These Life Sequence Charts (LSC) specify intra-object behavior, i.e. the time flow of an interaction or communication between objects can be illustrated. The properties can be defined for individual messages and in contrast to timing diagrams, one can express that a property holds at any time in a future state.

Hierarchical Hybrid Automata (HHA) have been proposed as a combination of UML state machine diagrams and hybrid automata to model complex and particular multi-agent systems. For specification and verification of HHA, Constraint Logic Programming (CLP) is used. For this, the user has to write a CLP program to specify properties of the HHA. As this
procedure is error-prone, a tool environment, namely the HieroMate, was presented in (Mohammed and Schwarz, 2009) to simplify this work. Within its environment, the user enters the specification graphically. The tool translates it to CLP code and subsequently verifies the model while applying the specification to it. Afterwards, HieroMate displays the result to the user. The tool reduces faults because it supports the user in creating the specification. It is limited to be applied to HHA, but closes the gap between intuitive specification on the one hand and formal verification on the other hand.

5.2.2 Previous Work

The previous subsection gives an overview about approaches of other workgroups. In this subsection, two specification techniques are presented that have been developed in the context of the OMSIS project.

**Safety-Oriented Technical Language.** Text-block-based specification techniques belong to the workaday life of an engineer. They are used for documentation and communication during design as well as for later modernization and migration. If they are verbalized in natural language, there are some weaknesses regarding clearness and completeness. Everybody has its own style of writing and sometimes it might be a problem to express a fact in an unambiguous form without misunderstandings. A standard language tries to solve this problem by limiting the valid expressions. It has a well-defined syntax and semantic and can be translated to other formal languages because of its well-defined grammar. A user-friendly standard language has to be formal on the one hand, but also understandable on the other hand. The user shall not be confronted with theory, so the expressions have to be close to the language an engineer uses in practice.

A Safety-Oriented Technical Language (SOTL) was introduced in (Heiner et al., 2001) in the framework of the DFG project “Certifiable Logic Programmable Controllers”. The authors specify requirements on control software of 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 for specifying the behavior of software of IEC 61131 conform PLCs because it considers the characteristic cyclic execution behavior of a PLC.

This specific view 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 had to be modified for the OMSIS project to describe desired plant behavior. The structure of the adapted SOTL is depicted in (Preuße and Hanisch, 2009).

As the language shall be applicable in practice, a compiling tool was implemented, namely the Compiler for SOTL. The tool contains all grammar rules and translates SOTL expressions automatically into CTL formulas. Figure 17 shows the graphical user interface. The software features four different flags. Custom expression includes the 21 possible pattern phrases. 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 compiler checks whether the input is valid according to the grammar rules 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) (Starke and Roch, 2002). The language of the compiler is switchable, so that in the current version the whole functionality is available in English and in German.

**Symbolic Timing Diagrams.** Graphical specification techniques are applied when a lot of information shall be displayed in a well-arranged way. This figures especially when complex program sequences are visualized. A possibility to describe those sequences is supplied by the Symbolic Timing Diagrams (STD), which are defined in (Schlöer, 2001). They are basically used in hardware development to describe reactive systems as for example in (Korf and Schlör, 1995; Schlör and Damm, 1993; Schlör et al., 1998). Within the “ForMaT” project (Kloos, 2001), they have been used as an input language for automated verification tools. In (Feyerabend, 1996), an extension of STD, namely the Real-Time Symbolic Timing Diagram (RT-STD), is used to describe dynamic aspects of systems.

Due to their structure, STD can be used for specifying plant behavior as well (Preuße and Hanisch, 2008) because they can represent state sequences of a large number of variables in a clearly arranged way. Thus, STD can describe even complex production sequences in an understandable and compact way in contrast to e.g. text-block-based description forms.

The automatic translation of a STD to a CTL expression was implemented in an editor tool. Figure 18 shows the
1. **Generating Formal Models**

After having presented the framework of the approach in Section 3, this section is now going into detail. First of all, it shall be clarified how formal models are generated. The idea is to use an existing CAD/CAE plant model, which can be created by using commercial tools as SolidEdge or ProEngineer. These models are imported into a simulation tool. In the following, some preconditions are listed, which limit the number of possible candidates.

### 6.1 Simulation Environment

As formal models shall be derived from an existing plant simulation, the simulation tool has to meet several demands. It is essential that discrete event models are applied because formal analysis methods cannot be used to analyze dynamic temporal behavior. Because of this, simulation models on the one hand and formal ones on the other hand shall be discrete as well. Furthermore, the data exchange format has to be open, i.e. the generated files shall be readable by any editor and they must not be encrypted. As NCES is module-based, the simulation tool has to model component-based as well, so that elements of both environments can be mapped to each other. For this, a library of components is developed once to define the connection between both worlds. This enables automatic mapping from simulated to formal models and allows recognizing signal connections between modules. Last not least, uncontrolled behavior is considered. For this, the simulation tool shall only model the uncontrolled behavior because the control functionality is provided by the controller. Hence, the simulation tool needs to have an interface to the applied controller to enable HiL testing.

Within the framework of the OMSIS project, the software Enterprise Dynamics of Incontrol Simulation Solutions is chosen to examine whether the procedure of translating a simulated plant to a formal plant model is applicable in practice. Figure 19 shows the 2D simulation of the EnAS testbed, which was created with Enterprise Dynamics. According to Figure 6, the conveyors (1, 2, 3), the Jack Station 4, the Slide Station 5 and the Gripper Station 6 are displayed. As described in Section 3, this model is connected to the hardware controller, so that the control software can be tested in a simulation environment. In the following, it is described, how the simulation plant model is converted to a formal NCE model.

### 6.2 Model Transformation

As the framework is focused on controller simulation and verification, a formal model of the uncontrolled plant has to be provided. This model is connected to the controller, so that a closed loop is established. In (Missal and Hanisch, 2007b) it is described, how plant models can be developed in a modular way using NCES. Another approach, describing modeling and verification in a more comprehensive way, is presented in (Hanisch et al., 2009). Since it is essential for going on further, the idea of connecting the controller to the formal plant model is introduced briefly in the following. Figure 20 shows the NCE Module, which models the control output behavior. It combines 8 binary outputs in one module, since this structure is usually used to build control input.
and output devices. The internal NCE structure is not correct per definition because the set of transitions is empty. Anyway, the net is composed to a flat NCES for analysis. A flat model does not contain any module frames or hierarchies, so the plain net includes a non-empty set of transitions and it will be correct. When performing the HiL test, the information of the hardware control output is passed to the module, i.e. the place will be marked if the control output signal is true and it will be unmarked if it is false, respectively. The state of the described module is transferred to the actuator modules within the plant module via condition signals.

The module describing the control input behavior, is shown in Figure 21. It again combines 8 binary inputs in one module because of hardware structure. The module is connected to the sensor modules within the plant module via condition arcs. Depending on the input state of the module, the tokens will flow to ON if the condition signal is true and to OFF if the signal is false, respectively. Beyond this, the digital input module has a Sample event input and a SampleO event output. These event signals trigger the cyclic input scan of the connected PLC, since an IEC 61131 compliant controller reads its inputs periodically. When performing the HiL test, the information of the module is transferred to the control input, i.e. if place ON is marked, the input signal will be true and if place OFF is marked, the input signal will be false.

Both output and input modules are connected to a plant model, e.g. to a conveyor. The exemplified NCE System is displayed in Figure 22. Due to clear arrangement, only a cutout of the converted simulation model shown in Figure 19 is displayed. The plant module contains nested NCE Modules, which model a relay, an engine, a conveyor and a sensor. Like in a real plant, the relay gets an order from the controller, namely from the control output. A drive is started, which moves a conveyor. If there is a workpiece that reaches a specific position, a sensor will be activated and transfers its information to the control input.

As described above, the condition signal from the output module is passed to the actuator relay. Analogically, the condition signal of the sensor is passed to the input module. Modules within a plant module can communicate via condition and event arcs. However, the plant module is usually not connected via event arcs to a control module to avoid event loops. Event outputs and inputs of a plant module can be connected to a workpiece module that is described in 4.2.5. The application of the whole model in a HiL test is described in a further section.

Translating the simulation model to a formal model shall happen semi-automatically. I.e., the implemented tool must not be more complicated than model creation by hand. Otherwise, it won’t be applied in practice. In Figure 23, the user interface of a prototypic software tool, namely the JED to NCES Converter, which handles the transformation task, is presented.

The tool manages an xml library, which contains all NCE Modules required for mapping. This library has standard functionality such as adding, deleting or resetting and it administrates path information of the modules.

Transformation starts opening a simulation model, which is created with Enterprise Dynamics. The input file is parsed and the status bar displays information about the process. Afterwards, all recognized simulation modules are mapped to formal modules. This is manually done in a link list dialog. Since a library of NCE Modules that represent Enterprise Dynamics modules shall be developed, the linking process can be automated, but at present, this information is not yet stored to the Enterprise Dynamics save file. Nevertheless, the
simulation tool can be adapted, so that it contains the required information about module types and enables automatic transformation. Anyway, the user is able to decide on his own, which modules shall be considered and he can omit certain insignificant ones.

At this point, enough information is supplied to generate a NCE System, which can be exported to a file. The exchange format is again xml, so that it is readable by standard editors. It is important to mention that mapping procedure does not touch the underlying behavior model within a NCE Module because it was already modeled and proven to be correct in previous steps. Because of this, the modules do not need to be verified again.

However, the NCE System not yet contains information about signal connections. Usually, plant modules are connected via condition arcs to the control model to avoid event loops. In case of HiL, the control hardware exchanges signals through input and output NCE Modules (Fig. 22). This is done with condition signals as well. Since plant modules normally do not share information among each other, the converting tool only has to find signal connections between plant modules on the one hand and input and output modules on the other hand. This is done in a Connection List Editor (Fig. 23). The tool is able to find corresponding signal connections, while analyzing the Enterprise Dynamics save file. The user is only prompted for decision if there are multiple choices, e.g. if a NCE Module has more than one condition input or output. Beyond this, the connections can be edited by hand within the Connection List Editor.

The mapping process is finished after signal recognition because any suitable information is extracted from the simulation model. The NCE System can be exported to a file and the user can proceed to the next step within the framework.

7 Applied Controller Test

The formal model generated in the previous section shall now be connected to the control hardware. This is done using an interface unit, which emulates Profibus or Profinet input and output devices, namely the SIMBA PNIO. For more information, please refer to the manufacturer website.

The control outputs are connected to the model, the plant reaction is computed and the result are transferred back to the control inputs. For this, the TNCES-Workbench, which was developed in the authors’ workgroup to formally analyze NCE Systems, is extended in its functionality.

Figure 24 shows a screenshot of the extension module. The user defines a SimbaPro Project Path, which includes information about the hardware configuration of the controller, i.e. the addresses of input and output devices and Profinet or Profibus IDs. After establishing the connection to the PLC, the software recognizes the input and output slides depending on the loaded NCE model. Beyond that, the user can change the configuration manually.

At the bottom, there are different buttons for task handling. First of all, the control output values are imported from hardware and visualized. As already described briefly in Section 3, this information is transferred to the formal model by pressing the button Insert new state. Since analysis is done in the background, the user does not have to care about the formal model because once generated, it can be applied. The inserted state is the starting point for reachability analysis. The graph is computed step by step and the marking of the calculated state can be transferred back to the model. If the value of at least one control input changes, i.e. if at least one token of a place within the input module moves, the visualization will display
the new information. Finally, the values of a certain interesting state are exported to the controller, the control software executes the next cycle and new output values are generated, so that the procedure can start again from the beginning. The test will be finished if the calculated graph is complete or if a specific final state is reached.

Currently, the HiL test is done manually, but it is intended to automate this task. Objective of this procedure is to detect malfunctions of control software. As already stated, the TNCES-Workbench calculates the reachability graph of the plant model under control, what is illustrated in Figure 25. One can see that the trajectories through the state space are not linear. Several branches can exist, which have to be analyzed. For this, the software passes through each trajectory of every branch and checks the correct behavior of the control software.

There are two possibilities to handle this task. On the one hand, the reachability graph is calculated based on the initial state and at the branching point, the state of the plant model as well as the state of the controller are recorded. After computation of a trajectory is finished, the recorded states of a branch are transferred back to the plant model and the controller to compute the other possible trajectory. On the other hand, the plant model and the controller have to be reset to their initial states after having finished computation of one possible path. Afterwards, the software passes through the already calculated trajectory to the branching point and takes the alternative path. Although the first possibility is the better way to be sure that the image of the controller state - including variables, flags, timers, counters etc. - is exactly the same at the branching point, it is rather complicated to be implemented. Reading and writing internal variables of a controller at run time is not trivial and only possible with additional software tools. For example the tool AutoSPy implemented by GWT-TUD can monitor the internal information of PLCs. Anyway, manipulating a running PLC program is much more complicated and usually not intended by the hardware manufacturer.

For this, the second possibility is more practicable because resetting a controller to its initial state is a standard function. The approach uses the plant model that is connected to models of the control inputs and outputs. The controller itself is seen as a black box that processes data inputs and provides information to data outputs. The reachability graph, which is calculated based on this model, reflects the state space of the controlled plant model. However, it does not represent the state space of the closed loop of plant and controller model and consequently the procedure does not provide verification.

HiL test is based on the idea that it is much better to perform automatic testing of a control implementation rather than doing it by a human being (who would simulate every scenario of plant behavior manually).

Obviously, the state of the closed-loop system is given by the state of the plant and the state of the controller. Since we only know the inputs and outputs of the controller, the method for testing is strongly limited. Having a plant model goes beyond most approaches for controller testing since this allows studying any plant behavior that will occur during executing a control program on the plant.

At the moment, testing is done using a formal model of plant behavior that comes from the simulation model (see Sec. 6). In future, the state of the controller (internal variables, counters etc.) will be included in the simulation. This requires access to the internal variable of the controller. It can be performed using, just for example, the tool AutoSPy.

Using that approach, it is even more realistic to perform validation of controller behavior than doing it by modeling and formal verification.

If errors occur while calculation - such as infinite loops or deadlocks - or the specification is violated, a so called error trajectory will be put out. It can be visualized in a Gantt Chart (Fig. 4) and with this information the control software can be adjusted. This marks the benefit of the approach very clearly because software errors are recognized before start-up of the real plant. Therefore, the procedure can save time and money and last but not least it can avoid dangerous situations effected by faulty software.

The approach of applying the hardware controller to the formal model provides another advantage. Since every controller has a specific interface and is programmed with software that is incompatible with
controllers of other manufacturers (especially IEC 61131 PLCs), the application of analysis tools is limited to specific controllers. However, this contribution proposes a method that can be applied independent of the manufacturer because the formal model only requests the information of control outputs and supplies information to the control inputs. For this, it is not important, what kind of controller is actually connected.

8 Controller Verification

The applied test has some restrictions concerning hardware. The PLC has a cyclic execution behavior and therefore needs specific time delays until signals can be transferred. This is usually not a problem because calculation of state space is more time-consuming than running one cycle of a control program. Anyway, it shall be kept in the back of one’s mind. Beyond this, the controller is seen as a black box in a HiL test. Internal variables, timers or counters are not considered, so that reachability analysis is not performed for the closed loop of controller and plant model but “only” for the plant model under control.

A more elegant way to verify the control software is shown in this section. The test case specification can be applied to the formal model of the closed loop of controller and plant. For this, a formal model of the controller has to be provided. This means, the control software has either to be translated to a formal model or the engineering process is done with a formal controller model, which is compiled to executable control code.

If the formal controller model is available, controller verification will be a practicable solution. Otherwise, translation means additional work and the approach presented in Section 7 should be applied.

The formal analysis of a closed-loop model applying a formal specification is called model-checking. This procedure provides several advantages (Hanisch et al., 2009). It delivers a machine-based assertion of correctness because every reachable state is computed. As stated in Section 7, classic simulation only tests a cutout of possible scenarios. Rare but often critical cases are not considered, so that erratic behavior can happen suddenly. For this reason, verification is usually the only possibility to exclude these worst case scenarios.

However, the specification of test cases has to cover these scenarios because they cannot be generated automatically, since every plant has its own characteristics. As a demand, the engineer who creates the specification has to know the plant details to handle critical situations and to exclude faulty behavior.

Model-checking is performed calculating the reachability graph of all possible states of the closed loop. This is a challenge even for modern computers because the complexity rises exponentially the more complex models become. Since the problem of state space explosion is well-known, there are several contributions, which consider the topic. For example, the approach in (Missal et al., 2007a) proposes to do hierarchical level-by-level verification to verify the system efficiently. The method basically is to first verify the tasks of a module (separate and application-independent). Afterwards, the pre-verified task controllers can be arranged in various combinations. Then, one can verify the coordination of the system, without considering the detailed behavior models of the task controllers of the modules.

As the OMSIS project is still ongoing, further work will be to facilitate automatic translation of control software to formal models. For this, IEC 61131 conform control software is considered. This is not to be seen as a restriction because controller verification is independent from control software implementation. Translation on the one hand and code generation on the other hand are performed using transformation rules, but this does not affect the model itself. Because of this, IEC 61499 conform control software shall also be considered, going one step ahead.

Regarding transformation of control programs written in IEC 61131, it is even simpler than transformation of IEC 61499 applications due to the sequential execution model of IEC 61131 programs. The methodology presented in (Hanisch et al., 1997) is clear and concise and covers all the features that are needed for such a transformation. It just needs to be implemented.

9 Conclusion

In this contribution, virtual start-up of plants using formal methods is presented. The work is part of the ongoing joint project OMSIS, which intends to speed up and ease the start-up of manufacturing systems as well as the migration of already existing plants.

The approach describes everything that is necessary (but not yet completed in detail) to perform controller test in a systematic, formal way. The crucial point is to have a detailed plant model to interact with the hardware controller in closed loop. Although this is rather trivial, it is, however, not so common in control engineering. If once such a model - designed using commercial simulation software - exists, this paves the way to apply formal specification techniques and formal methods to controller design and testing. Also this is not very new, but the fact is, that nowadays industry - at least the industrial partners within the OMSIS project - feels the need to go a step further onto the field of model-based technologies for controller design. This has not been caused by promises from academia, but by the desire of companies.

That gives some hope for further development of the issues described in this contribution that would eventually bring model-based technologies one step closer to daily practice. As the work is ongoing, some open questions remain that should be answered in future. The major goal is to provide an integrated tool chain that does not complicate the practical work of an engineer. For this, some case studies may follow...
that evaluate the user-friendliness because every idea is just as good as the implementation that is provided to the user. Furthermore, the complexity of the whole framework has to be evaluated to answer the question how long it would take to test more complex systems.

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


Note


