Approaches to the Formal Verification of Hybrid Systems

S. Kowalewski, Frankfurt; P. Herrmann, S. Engell, H. Krumm, and H. Treseler,

Dortmund; Y. Lakhnech, Grenoble; R. Huuck and B. Lukoschus, Kiel

This paper presents two different approaches to the problem of formally verifying the correctness of control systems which consist of a logic controller and a continuous plant and, thus, constitute a hybrid system. One approach aims at algorithmic verification and combines Condition/Event Systems with Timed Automata. The first framework is used to model the controller and the plant in a block-diagram representation, which is then translated into the latter model for analysis by available tools. A second approach is presented which is based on deductive verification. It allows for a structured analysis of compositional specifications formulated in a temporal logic called cTLA. This logic is a compositional style of the Temporal Logic of Actions established in Computer Science by Lamport. Both approaches are introduced using a common example and the results of their application are discussed. As an outlook, a possible strategy for integrating algorithmic and deductive verification of hybrid systems is sketched at the end of the paper.

1 Introduction

Hybrid systems often arise from the interaction of logic controllers or other discrete automation systems with a continuous plant. Usually, the controller part of such constellations is designed from a specification of the closed-loop behaviour. To check whether a given logic controller meets this specification, it is therefore necessary to consider the closed loop system and, thus, to analyse a hybrid system. One way to do this is to build formal models of the system and of the requirements from the specification, and to apply formal methods (i.e., rigorous analysis in the mathematical or logical sense) to prove that the system model has the required properties. This general procedure is called *Formal Verification* and has been a research topic in Computer Science for more than three decades. While the focus originally was on purely discrete systems (general computer programs or computer systems), in recent years new models and methods were developed which are capable of handling hybrid systems. Some of these approaches have been investigated in the German focussed research program *Analysis and Synthesis of Technical Systems*

Dr.-Ing. Stefan Kowalewski is with the Corporate Research and Development of Robert Bosch GmbH where he is concerned with software technology for car control systems. The results presented in this paper were developed while he was a Senior Scientist in the Process Control Laboratory at the University of Dortmund. His research interests are discrete event and hybrid systems, formal methods for the design and analysis of control software, and software technology.

Address: Robert Bosch GmbH, FV/SLD, Postfach 90 01 69, D-60441 Frankfurt am Main, Germany, Tel. +49 69 7909-530, Email: stefan.kowalewski@de.bosch.com

Dr. Peter Hermann is a Senior Scientist in the Postgraduate Programme *Modeling and Model-based Design of Complex Technical Systems* at the University of Dortmund. His main research interests are formal-based development of distributed and hybrid systems and security aspects of component-structured distributed software.

Address: Lehrstuhl IV, Fachbereich Informatik, Universität Dortmund, D-44221 Dortmund, Germany, Tel. +49 231 755 4836, Email: herrmann@ls4.cs.uni-dortmund.de

Prof. Dr.-Ing. Sebastian Engell is Professor of Process Control in the Chemical Engineering Department at the University of Dortmund. His main areas of interest are controller design, hybrid systems, and optimization of batch processes.

Address: Lehrstuhl für Anlagensteuerungstechnik (CT-AST), Fachbereich Chemietechnik, Universität Dortmund, D-44221 Dortmund, Germany, Tel. +49 231 755 5126, Email: s.engell@ct.uni-dortmund.de

Prof. Dr. Heiko Krumm is Professor of Computer Networks and Distributed Systems in the Computer Science Department at the University of Dortmund. His main research interests are formal specification, implementation, and verification of communication systems and distributed applications.

Address: see P. Herrmann, Tel. +49 231 755 4674, Email: krumm@ls4.cs.uni-dortmund.de

Dipl.-Ing. Heinz Treseler is Research Assistent and PhD student at the Process Control Laboratory in the Department of Chemical Engineering of the University of Dortmund. He is developing a computer tool for the verification of logic controllers.

Address: see S. Engell, Tel. +49 231 755 5171, Email: h.treseler@ct.uni-dortmund.de

Prof. Dr. Yassine Lakhnech is Professor of Computer Science at the Université Joseph Fourier, Grenoble, France.

Address: Laboratoire VERIMAG, Centre Equation, 2 av. de Vignate, F-38160 Gières, France, Tel. +33 476 6348-31, Email: yassine.lakhnech@imag.fr

Dipl.-Inform. Ralf Huuck is Research Assistant and PhD student at the Department of Computer Science and Applied Mathematics at the University of Kiel. His research areas are formal methods in real-time and hybrid systems, programmable logic controllers as well as distributed and parallel systems.

Address: Lehrstuhl für Softwaretechnologie, Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität Kiel, Preußerstr. 1-9, D-24105 Kiel, Germany, Tel. +49 431 5604-97, Email: rhu@informatik.uni-kiel.de

Dipl.-Inform. Ben Lukoschus is Research Assistant and PhD student at the Department of Computer Science and Applied Mathematics at the University of Kiel. His research areas are formal methods in discrete-event, real-time and hybrid systems, and specification and verification of hierarchical control programs.

Address: see R. Huuck, Tel. +49 431 5604-22, Email: bls@informatik.uni-kiel.de

with Continuous/Discrete Dynamics (KONDISK). In this program a key question is whether methods originating from computer science are applicable to technical control systems, and if and to what extent they can be integrated with approaches from the control community. The paper reports some of these activities.

In Computer Science, two different approaches to formal verification are distinguished, *algorithmic* and *deductive* verification. Algorithmic verification, often called *model-checking*, means that a computer algorithm is used which receives a model of a system and a specification of its required behaviour as input, and then checks whether the requirements hold for all possible behaviors of the system model. This is done basically using search techniques, very often by computing the reachable states of the system. Each verification algorithm is applicable to a particular class of systems (e.g., finite transition systems). In general, algorithmic verification is only possible for such classes for which it is guaranteed that the search procedure terminates. In the case of hybrid systems, this is only true for very restricted classes. It is therefore often necessary to find a finite abstraction of a hybrid system before algorithmic verification can be applied. On the other hand, the advantage of algorithmic verification is that the user only has to provide the system model and the requirements and can leave the rest to the algorithm. No further expertise and knowledge of the analysis technique is needed.

When applying deductive verification, also called *theorem proving*, the question whether a system has certain desired properties is answered by creating a proof. For this purpose, the user not only has to specify the system behavior and the requirements in an appropriate logic, but also has to find a suitable sequence of arguments. Although this is often supported by a set of proof-rules which can be applied in a schematic way, in the end, the verification success depends much more on the intuition, creativity and experience of the user than it does for algorithmic verification. However, one main advantage of deductive verification is that the application domain is not restricted to systems with finite search spaces or finite approximations. If a suitable theory is available, infinite systems can be handled also. This is important in the case of hybrid systems which are infinite by definition.

In *KONDISK*, both lines of verification have been followed in different projects. In this contribution, these projects are described and the results are discussed. As an outlook, an approach is proposed which represents an attempt to overcome restrictions which were experienced for the algorithmic verification by integrating algorithmic and deductive techniques. The paper is organized as follows. In the next section, the work concerned with algorithmic verification is presented. Section 3 is devoted to the deductive and structured verification of compositional specifications in cTLA. In Sec. 4, we summarize and discuss the results, and sketch the possible integration of both basic techniques for the verification of hybrid systems.

2 Algorithmic Verification of Hybrid Systems using Condition/Event Systems and Hybrid Automata

There are two basic pre-requisites for the algorithmic verification of hybrid systems: a modeling paradigm which makes it possible to express hybrid dynamic phenomena like switching of the continuous dynamics or resets of the continuous state, and an algorithm for computing properties of a system represented in this modeling framework. One suggestion for a hybrid systems modeling paradigm are *Condition/Event Systems (CESs)* [21] which are based on a block-diagram and signal-flow presentation and make it possible to separate the discrete and continuous parts of the dynamics in a modular fashion. Since CESs utilize notations which are familiar to systems engineers, they seem to be well suited for the verification of hybrid control systems in industrial practice. However, they are lacking the second pre-requisite: There is currently no computer tool available for the direct analysis of CESs. This is different for *Hybrid Automata (HA)* [3], a model for hybrid systems, for which effective algorithms were developed in computer science over the last decade. On the other hand side, the semantics of HA, as we experienced, are less intuitive for engineers who often are familiar only with causal, signal-based models as they are used for instance in block-diagram oriented simulators for continuous systems as e.g. Matlab/Simulink. One approach to overcome this situation is to provide a transformation from CESs to HA which makes it possible to use CESs for building models and HA tools for their analysis.

2.1 Condition/Event Systems and Hybrid Automata

CESs were first introduced by Sreenivas and Krogh in [21] and presented in this journal in [16]. In this section, we recall only some basic properties which are needed for the following discussion. The interaction between subsystems or blocks in a CES model is based on the intuitive concepts of *enabling/disabling* and *forcing* of transitions. A transparent distinction between these two mechanisms is achieved by the introduction of two classes of continuous time signals which can be both input and output signals of one systems: *condition signals* and *event signals*. Condition signals are piecewise constant and their values correspond to discrete states. Event signals are pointwise non-zero and carry information about currently occurring discrete state transitions. A CES has a *condition input signal* $u(t)$, an *event input signal* $v(t)$, a *condition output signal* $y(t)$, and an *event output signal* $z(t)$. The corresponding sets of conditions or events are U, V, Y, and Z, respectively. The basic definition in [21] defines a CES as an input/output mapping which gives the set of admissible output signal trajectories for each possible input signal trajectory. As one possible model for the internal behavior of a CES, Sreenivas and Krogh introduced *Discrete State CES (DCES)*. They are characterized by the introduction of a set Q of discrete states and three functions, the *state transition function* f, the *condition output function* g, and the

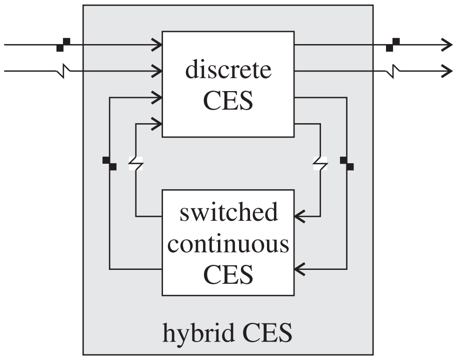

Bild 1: Generic structure of a hybrid CES

event output function h. The latter specify the valid state trajectories of a DCES, also called *runs*, in the following way: A right-continuous and finite-valued function $r : \Re_{\geq 0} \to Q$ is a run over a DCES, if it satisfies the following conditions for each $t \in \Re_{\geq 0}$:

- $r(t) \in f(r(t^-), u(t^-), v(t))$,
- $y(t) = g(r(t), u(t))$,
- $z(t) = h(r(t^-), r(t), v(t))$, and
- $r(0) = q_0$,

where $r(t^-)$ and $u(t^-)$ are abbreviations of $\lim_{\varepsilon \to 0} r(t-\varepsilon)$ and $\lim_{\varepsilon \to 0} u(t-\varepsilon)$, respectively. The set of all signal trajectories $(u(\cdot), v(\cdot), y(\cdot), z(\cdot))$ for which there exists a run are called the *behavior* of the DCES.

A DCES can be regarded as an untimed finite state machine which is embedded in a time-dependent signal space formed by the condition and event input and output signals. The condition input signal constitutes conditions for changes of the state of the system (hence can *disable* or *enable* certain transitions) whereas the event input signals can *force* transitions. Event outputs can be generated if and only if the state of the system makes a transitions, condition outputs provide information about the actual state of the system. Transitions can be forced (only by event signals) or occur spontaneously, and can be non-deterministic.

A straightforward way for the integration of hybrid dynamics into the CES framework is illustrated in Fig. 1. The flow lines with the block-shaped symbol are condition signals, the arrow symbolizes event signals. The main idea is to separate the continuous dynamics from the discrete dynamics and then to encapsulate both blocks in one *hybrid* CES. Thus, the continuous subsystem as well as the complete hybrid system interact with their environment by discrete conditions and events. In the case of the *switched continuous CES*, the condition input can switch the continuous dynamics (i.e., the right hand side of the differential equations) and the event input can be used to reset the continuous state. The condition output indicates the region where the continuous state is, and the event output indicates a crossing of a switching manifold, usually a boundary of a region. Slightly different concepts for hybrid models based on CES can be found in [17] and [4].

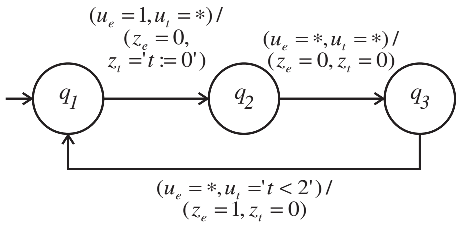

Bild 2: Graphic illustration of a TCES (Example)

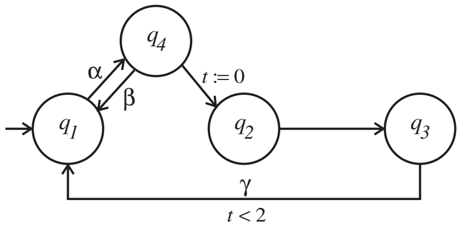

Bild 3: Graphic illustration of a TA (Example)

In the following we will restrict our interest to the most simple class of hybrid systems, namely timed systems which have only one kind of continuous variable, i.e., the time. The basic ideas and results, however, can be extended to more complex hybrid systems. For the structure in Fig. 1, this restriction means we have a *Timed CES (TCES)* in which the switched continuous CES is a set of *CE timers*. Each timer has one continuous variable, a *clock*, which can be reset to zero by an input event and for which an output event indicates that a predefined period since the last reset has just elapsed. To illustrate the behavior of a TCES, it is convenient to take the state graph of the DCES part and assign the external input and output as well as the input and output from and to the timers to the corresponding transitions and states. Fig. 2 shows a small example with one timer t and only condition input and event output signals. The subscript e stands for external, i.e., from and to the environment of the TCES, and t identifies the signals from and to the timer. The star symbolizes that any value of the respective signal is valid. The system will move from the initial state q_1 to q_2 only if the condition $u_e = 1$ holds. It will then reset the timer t. The transition from q_2 to q_3 is not depending on any condition and can happen at any time. In state q_3, the system can only go back to q_1 as long as $t < 2$ holds for the timer. All transitions in this example are spontaneous, that is, not triggered by an event. Note that, in this case, remaining in a state is always possible and is not explicitly symbolized by self-loops.

The counterpart to TCES in the world of Hybrid Automata are *Timed Automata (TA)* [2]. They are a generalization of finite automata over infinite strings by continuous variables, here again called *clocks*. Clocks increase with the rate 1, they can be reset, and clock values can be conditions for transitions. If a TA model is composed from multiple parts, the modular TAs are synchronized using event labels. Fig. 3 shows an example of a TA. Its behavior is close to the TCES example in Fig. 2. A discrete state q_4 is introduced to model the condition signal of the TCES. The transitions from q_1 to q_4 and back are synchronized with events α and β in the environment which make the condition for the transition from q_4 to q_2 true or false, respectively. Again, the clock t is reset by the transition to q_2, and the system can move from q_2 to q_3 unconditionally. The transition back to q_1 is only possible as long as the clock value is less than 2. This transition is synchronized with the environment by the label γ.

2.2 From Timed Condition/Event Systems to Timed Automata

From the two examples in Figs. 2 and 3 it seems that TCES and TA are very similar models, and the conjecture that they have the same expressive power is intuitive. However, there are subtile differences. The main difference is the following. As mentioned before, TCESs are based on an input/output system view with free input signals. Since the timer values are input signals to the discrete part of the TCES, in each discrete state the possible transitions (including staying in the state) must be specified for all possible timer values. In computer science terms, TCES (and CES in general) are *input-enabled*. This means that any finite state trajectory which is possible according to the TCES model is extendable to an infinite one. For TA, the validity of a state trajectory is decided by an acceptance criterion, usually that some states are visited infinitely often. This concept does not require input-enabledness. It is possible that a model allows finite state trajectories which cannot be extended to an accepting (and, thus, infinite) run of the automaton. For example in Fig. 3, there is no *local* condition that we cannot stay in q_2 for 2 or more time units because then we are stuck in q_3 and cannot complete an accepting run. Instead, the transition from q_2 to q_3 must be taken less than 2 time units after the clock reset. As a matter of fact, this particular example is not expressible by TCES. However, since we are interested in a transformation from TCESs to TA, we can expect the resulting TA to be input-enabled and that the problem of non-input-enabled TA does not occur.

An effective transformation algorithm which maps TCESs into TA with an equivalent behavior is described in detail in [11]. The main result which is proved in [11] is that every system modeled by a TCES can also be modeled by a TA such that any reachability problem for the TCES can transformed into a reachability problem for the corresponding TA. The basis for this result is an equivalence notion which relates input/output condition/event signal sequences (or behaviors) to timed words, which are used to define the semantics of TA. The main idea of the transformation is to map the discrete states of the TCES into discrete states of the TA, and then to transform the dependency of transitions on signal values into transition constraints in the TA. In the following, some particular aspects are sketched to provide a feeling for the problems which had to be solved. Readers who are interested in the technical details are refered to [11].

- As illustrated in the examples in Figs. 2 and 3, the conditioning of transitions by condition signals has to be represented by additional states in a TA. In general, the resulting state space is $(Q \times U) \cup \{q_{ini}\}$. It is necessary to introduce the dummy initial state q_{ini} because the initial value $u(0)$ of the condition input is not known a priori. From q_{ini} there are spontaneous transition to all states in $\{q_0\} \times U$. Of course, the blow-up of the state space due to the Cartesian product of the original state space of the TCES and its condition input alphabet is a disadvantage of the transformation. However, it is often possible to reduce the state space by collapsing all states (q, u^i) and (q, u^j) for which u^i and u^j are equivalent in the sense that $f(q, u^i, *) = f(q, u^j, *)$ and $g(q, u^i) = g(q, u^j)$.
- In CES, transitions can be forced by incoming events which means that they have to take place synchronized with the peak in the event signal. This concept does not correspond completely to the (non-causal) synchronization of events in TA because it is based on directed signals. For example in Fig. 2, the transition from q_3 to q_1 generates the event $z_e = 1$, but it can happen independent of whether some other system will do a simultaneous transition upon receiving this event. In Fig. 3, this is different because the corresponding transition can only take place if another TA can and will also make a transition with the label γ. The directed forcing can be represented with the help of additional transitions and the concept of *urgent* transitions which are transitions that have to take place as soon as they are enabled.
- A very subtle difference between TCES and TA is that clock values as conditions for transitions are evaluated at the time t of the transition in TA, but at time t^- in TCES (see the definition of the state transition function in the previous section). Thus, the clock constraints have to be adapted, e.g. from $t \geq 2$ in TCES to $t > 2$ in the corresponding TA.

2.3 An Example

To illustrate the transformation from TCES to TA and to provide an impression of the kind of problems which can be solved by this approach, we use a small laboratory batch plant example from [13]. The system is shown in Fig. 4. The following production sequence takes place: “A batch of sodium chloride solution is filled into tank 1 T1 via valve V12 and then evaporated until a desired concentration is reached. During evaporation, the condenser C1 is in operation and condenses the steam coming from T1. When the desired concentration is reached, the material is drained from T1 into tank 2 T2 via valve V15 as soon as T2 becomes empty. A post-processing step takes place in T2, before the material can be pumped from T2 to a subsequent part of the plant via valve V18.” We suppose that in the undisturbed case the operation sequence described above is ensured by the controller and direct our interest to the problem of appropriate control reaction to disturbances. In particular, we look at the consequences of a cooling breakdown in the condenser. This failure can lead to a dangerously high pressure in the condenser tube, if the evaporation is

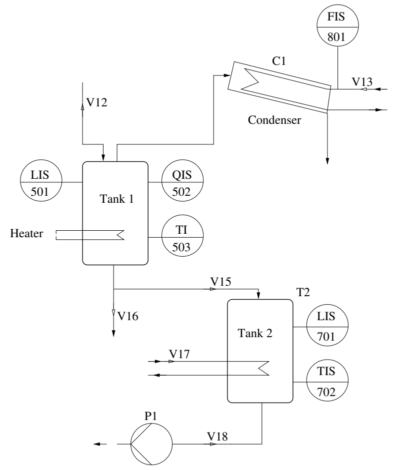

Bild 4: Flow chart of the example batch system

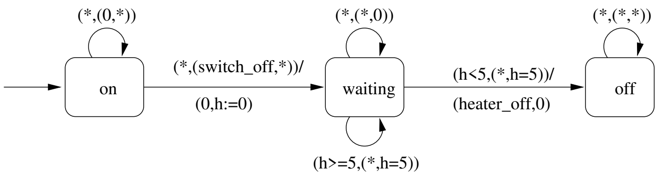

Bild 5: TCES model for the module *heater* of the evaporator example

continued. On the other hand, when the heater in T1 is switched off, the pressure in C1 will not rise anymore. We assume that the material in T1 becomes solid after a certain time when it cools down and cannot be drained into T2 via V15. This situation is of course not desired. As a consequence, the controller should not switch off the heater immediately after a cooling breakdown. But, if T2 is still full, switching off the heater must be delayed to ensure that T1 can be drained via V15 before the material becomes solid. However, the controller should start draining T2 via V18 as soon as a cooling breakdown occurs, so that no time is lost.

For a given controller, its correctness can be checked by modeling the controller and the plant with TCES, transforming the model into TA, and then performing reachability analysis to find out whether the undesired states "material solid" or "pressure too high" are reachable. The complete model can be found in [11]. Here, we present one module, the heater. The TCES model is shown in Fig. 5.

It has the input event alphabet $Switch = \{switch\_off\}$, output event alphabet $Heateroff = \{heater\_off\}$ and the timer h with threshold value (5). Assuming that the heater is switched off with a delay of 5 time units, the module Heater exactly describes this process. Here, after receiving *switch_off* the module delays the shutdown of the heater by 5 time units. The heater is switched off by the transition from *waiting* to *off*. The event *heater_off* is sent.

The TA resulting from the transformation procedure is shown in Fig. 6. In this case only q_{ini} has to be added as an additional state. Urgent transitions are shown as dashed arrows. Note that since both transitions from

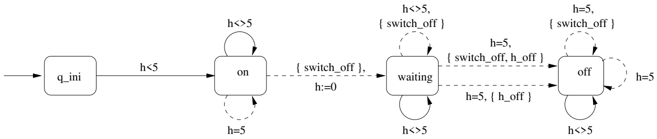

Bild 6: TA model for the module *heater* of the evaporator example

state *waiting* to state *off* are urgent the automaton cannot stay in *waiting* beyond 5 which is the desired behavior.

2.4 A Tool Environment for Algorithmic Verification

The modeling and analysis framework presented above is the necessary core of a verification tool for hybrid systems. However, it covers only a part of what is needed when algorithmic verification techniques shall be applied in industrial practice where control engineers and not computer theorists are involved in the software development. It is desirable to support the verification process by the following additional functionality:

- Interfaces for modeling with familiar notations (like block diagrams).
- Model libraries.
- Automatic translation of control code into formal models.
- Interfaces to analysis tools.
- Automatic abstraction of complex hybrid systems by simpler models which can be analyzed by the available tools.

In another KONDISK project, a tool environment has been developed which aims at providing this functionality [12]. It is also based on TCES as the main modeling paradigm, and it takes advantage of the transformation presented before. But it complements this approach by a suite of interfaces, either as front-ends, as e.g. hierarchical block diagram and state graph editors, translation from control code to TCES, or approximation of switched continuous CES by TCES, or as back-ends, e.g. interfaces to TA analysis tools or discrete model-checkers.

3 Deductive and Structured Verification based on cTLA

Due to their complexity, formal specification and deductive verification of real-life hybrid systems is a rather time- and cost-consuming task. The compositional specification technique cTLA [7], however, facilitates the development of system specifications as well as the formal proof of interesting system properties. Since system specifications are composed of smaller specification blocks each modeling only a single system component, the system structure can be modeled in a quite direct and open manner. Moreover, cTLA guarantees that properties fulfilled by a single component are also properties of each system containing this component [9]. Therefore, one can reduce the verification effort of system properties as well. Instead of the complex system model, proofs of desired properties are based on subsystem models only. If a system is composed from modules in a suitable way, these subsystem specifications are quite simple and the proof can be performed easily.

Since certain properties of hybrid systems are influenced by many cooperating system components, however, the subsystem specifications used for proofs tend to be large and structured verification is not supported well. Therefore, cTLA supports not only resource-oriented specifications where each physical component of a system is modeled by a separate specification, but also supports constraint-oriented specifications. Here, system components are not specified monolithically but are composed from constraint specifications each describing only a certain aspect of the component. For instance, a vessel may be described by these different aspects: volume of liquid in the vessel, temperature of the liquid, and pressure in the vessel. These aspects can be specified by separate specification blocks, the composition of which forms the entire vessel specification. In a proof of a system property, the used subsystem model consists only of those component constraint specifications which realize the system property focused on. For example, to prove that the pressure in a vessel does not exceed a certain maximum, one only needs to look at the specification block modeling the pressure in this vessel.

cTLA is based on Leslie Lamport's specification technique "Temporal Logic of Actions" (TLA) [20], which was supplemented by a compositional process concept (cf. [18]). Processes encapsulate private system state variables. State transitions are specified by actions each describing a class of transitions. As in the formal description language LOTOS, interactions between processes are modeled by joint system actions. These are coupled local process actions which have to occur simultaneously. Data transfer between processes is described by means of data parameters of actions. The formal semantics of cTLA specifications is defined by a mapping from cTLA specifications to equivalent TLA formulas. cTLA was successfully used to model and prove computer science applications, in particular communication protocols. In order to model realtime assumptions of discrete system controllers as well as continuous flows of technical systems, it was supplemented by further syntactical constructs [8] which also correspond to similar TLA concepts [1, 19]. To model realtime constructs one can define that an action must not be enabled for a certain period of time without being executed. Furthermore, a special action facilitates the description of continuous flows by means of difference equations.

This cTLA extension was applied to prove that hybrid systems fulfill certain safety properties [6] and to verify functional properties of hybrid system control [10]. Moreover, cTLA also supports the reuse of specifications and proofs. So-called specification frameworks adopt the well-known notion of "frameworks" from software engineering and transfer it from the construction of software systems to the construction of formal models and proofs [9]. A specification framework contains libraries of cTLA module types and of theorems. A specification is developed by instantiating and composing cTLA module types. Formal proofs that a more detailed system specification provides for a more abstract one are facilitated by the theorems which have already been proven by the developers. The user can easily reduce a proof into proof steps which correspond directly to these theorems. Also, a tool facilitates the selection of suitable theorems and the necessary consistency checks [5]. Below, we introduce a specification framework which is devoted to the formal hazard analysis of hybrid systems. Moreover, we outline the use of framework theorems to prove that hybrid systems fulfill certain safety properties.

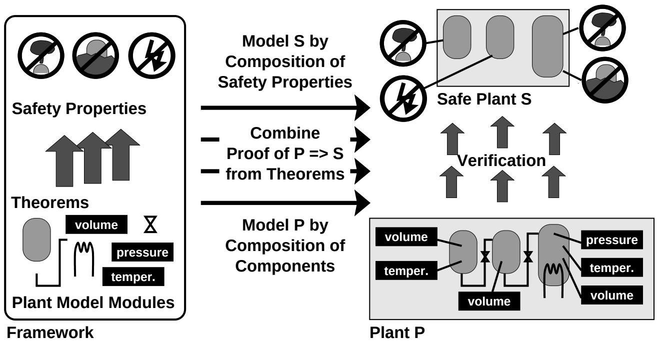

Bild 7: The framework for the formal hazard analysis of chemical plants

3.1 The Hazard Analysis Framework

As depicted in Fig. 7, the hazard analysis framework [8] consists of two collections of specification modules, safety property modules and plant model modules, and of theorems. The safety property modules are used to develop safe plant specifications describing a list of safety properties to be fulfilled by a hybrid system in order to prevent serious system hazards. A module states either that a system component observes a critical limit (e.g., the power of a heating does not exceed 6000 Watt to prevent overheating) or that a critical system state will be excluded (e.g., a pump is switched off if the flow of liquid is blocked due to a closed valve). A formal description of a single safety property is developed by instantiating a safety property module, while a complete safe plant specification is derived by composing these module instances. Likewise, a specification of a plant system is composed from plant model instantiations. A group of plant model modules is devoted to continuous, discrete, and hybrid system components as vessels, pipes, sensors, actors, and controllers, which are mainly specified in a constraint-oriented way. Another group of plant modules specify component malfunctions to be tolerated by a hybrid system (e.g., leakage of valves, blocking of pipes, failures of active components like pumps or heatings).

The framework theorems which are used to verify that a plant specification fulfills a safe plant description, are implications of the form "plant subsystem $\wedge$ malfunction subsystem $\wedge$ safety property subsystem $\Rightarrow$ safety property". A theorem asserts that a certain plant subsystem guarantees a safety property even if certain malfunctions occur. For instance, a pump can be pre-

```
THEOREM MaxPressureBoilerCondenser LET
  Sys ≜ PROCESSES Subsystem
  Modular components of a technical plant:
    BoilerVol : VesselVolume
                (bcapacity, binitvolume);
    CondenserVol : VesselVolume
                   (ccapacity, cinitvolume);
    Vaporization : VesselVaporize
                   (fluiddense, gasdense,
                    enthalpy);
    ...;
  Already proven safety properties:
    CoolingPowerDecrease : PowerMaxDecrease
                           (cdecrate);
    BoilerMaxFluid : VesselMaxVolume
                     (bfluidmaxvolume);
    ...;
  ACTIONS
    CONT (INPUT ... ; OUTPUT ...) ≜ ...;
    continuous volume flow
    Alarm ≜ ...; alarm cooling below minimum
    Release ≜ ...; alarm cooling above minimum
    HeatingOff ≜ ...; switch off heating
    HeatingOn ≜ ...; switch on heating
  Pars ≜ sensedpower > bmaxpower +
           cdecrate · sensetime ∧ ...;
  EnvCond ≜ Enabled(SensorAlarm.alarm) ⇒
              Sys.e_Alarm = "enab" ∧ ...;
  IN  Sys ∧ Pars ∧ HENCEFORTH EnvCond ⇒
        VesselMaxPressure (maxpress);
```

Bild 8: Framework Theorem

vented from pumping against a closed valve by using a control device which switches it off if a sensor indicates excess pressure. That is reflected by a framework theorem, the plant subsystem of which consists of the pump, the sensor, the controller, the pipe of pumped liquid, and appropriate links between these components. Since this control subsystem may tolerate the blocking of a pipe but no controller or sensor failures, the malfunction subsystem contains only a plant model describing pipe blockings. The safety property subsystem was introduced since plant components often interfere with many other and, perhaps, very distant components causing lengthy plant subsystem descriptions. In order to prevent this and, consequently, a large number of framework theorems, some theorems contain an additional conjunct of safety property modules at the left hand side describing safety requirements for the subsystem environment. These safety properties are verified transitively by using other framework theorems. For instance, consider a proof that a number of serially linked vessels never run empty. Without using a safety property subsystem in theorems, we would need a separate theorem for each line of vessels. The proof, however, can be performed easily by inductive application of a theorem stating that a vessel will not run empty under the safety requirement that its predecessor does not become empty as well.

3.2 Example Verification

Fig. 8 lists a framework theorem [9] of the hazard analysis framework which can be used to prove that the pressure in the boiler T1 and in the condenser C1 of the example system (cf. Fig. 4) does not exceed a certain value. The theorem asserts that the subsystem *Sys* consisting of cTLA plant modules and safety properties implies the safety property *VesselMaxPressure* if the subformulas *Pars* and *EnvCond* hold. *VesselMaxPressure* states that the pressure in a vessel does not become larger than the value *maxpress*. The subsystem description *Sys* contains a list of instantiated cTLA specification modules (PROCESSES) and the couplings of local cTLA module actions to joint subsystem actions (ACTIONS). It describes a typical combination of a boiler and a condenser with a controller switching off the boiler heating if the condenser cooling fails. *Pars* is a boolean condition guaranteeing that the processes of *Sys* and the property to be proven are parametrized consistently. In this example, the sensor which monitors the cooling has to react before the cooling power *sensedpower* falls below the maximum heating power *bmaxpower* to prevent excess pressure. *EnvCond* models an invariant to be kept by the environment. It assures that the environment of *Sys* does not block the actions of *Sys* and therefore does not spoil liveness and realtime assumptions. For instance, the system actions *Alarm* and *HeatingOff* modeling a sensor alarm to the controller resp. the controller command switching off the heating, must never be blocked by other system parts not modeled by *Sys*.

In order to prove the safety property *VesselMaxPressure*, the user instantiates the parameters in the theorem according to the generic parameter replacements in the system specification. Since the theorems are already proven, one only has to check that *Sys* is a subsystem of the system specification, that the instantiated parameters fulfill the condition *Pars*, and that the processes of the system specification not listed in *Sys* keep the environment condition *EnvCond*. Furthermore, the correctness of the safety properties assumed by *Sys* have to be proven by other theorems of the framework. The selection of theorems as well as the checks are facilitated by the tool COAST [5]. Based on the system specification and the specifications of safety properties, COAST selects suitable theorems and performs the checks of *Sys* and *EnvCond* directly. Furthermore, COAST proves the condition *Pars* by means of a theorem prover.

4 Conclusions and Outlook

Two approaches for the formal verification of hybrid systems were presented, one based on algorithmic, one on deductive techniques. In both cases, particular emphasis is put on means for structuring the models and specifications: The algorithmic verification is based on modular, signal-connected models which make it possible to restrict the analysis of properties to those blocks which are relevant. The deductive approach supports not only component-wise system specifications but also constraint-oriented specifications which allow for an additional dimension of structuring. Applications showed that both approaches make it possible to verify technical processes by taking advantage of the available struc-

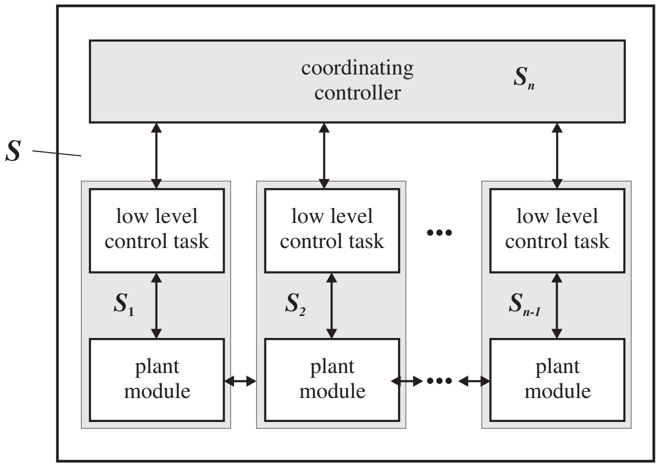

Bild 9: Distributed control structure under consideration

turing techniques. However, a comparison of both methods with respect to the user effort and the maximal size of treatable problems is still a topic for further research.

In the case of the algorithmic approach based on TCES and TA, it was experienced that the computational costs of the analysis algorithm make it impossible to deal with systems where several controller modules are coordinated by an upper level controller (see Fig. 9). The number of modules which have to be considered for showing a specific property in such a structure most often is too large for algorithmic verification. Therefore, a new KONDISK project is concerned with a combined algorithmic/deductive approach to deal with such problems. The main idea is to apply model-checking to show that properties hold for the usually small, low level control systems S_1 to S_{n-1} *under certain assumptions* (like that no other controller module interferes or that no undesired situation occures in another plant module). In a second step, it then has to be proved by deductive techniques that for the complete system S the assumptions for each module hold mutually and are ensured by the coordinating controller. We hope that this approach will help to make formal verification applicable to larger, distributed logic control systems.

Acknowledgements

The work presented here has been supported by the German Research Council (DFG) under the focused research program KONDISK (*Analysis and Synthesis of Technical Systems with Continuous/Discrete Dynamics*) and partially by the temporary graduate school ("Graduiertenkolleg") *Modelling and Model-Based Design of Complex Technical Systems* at the University of Dortmund.

References

[1] *Abadi, M.; Lamport, L.*: An old-fashioned recipe for real time, ACM Transactions on Programming Languages and Systems, vol. 16, no. 5, 1543-1571, 1994.

[2] *Alur, R.; Dill, D.*: A Theory for Timed Automata. Theoretical Computer Science 126, 1994, 183-235.

[3] *Alur, R.; Courcoubetis, C.; Halbwachs, N.; Henzinger, T.; Ho, P.–H.; Nicollin, X.; Olivero, A.; Sifakis, J.; Yovine, S.*: The Algorithmic Analysis of Hybrid Systems. Theoretical Computer Science 138, 1995, 3–34.

[4] *Engell, S.; Hoffmann, I.*: Modular Hierarchical Models of Hybrid Systems. Proc. 35th IEEE Conference on Decision and Control (CDC), Kobe, 1996, 142-143.

[5] *Herrmann, P.; Drögehorn, O.; Geisselhardt, W.; Krumm, H.*: Tool-supported formal verification of highspeed transfer protocol designs. In Proc. 7th Int. Conf. on Telecommunication Systems - Modelling and Analysis, Nashville, 531-541, 1999.

[6] *Herrmann, P.; Graw, G.; Krumm, H.*: Compositional Specification and Verification of Hybrid Systems in cTLA. Proc. 1st IEEE International Symposium on Object-oriented Real-time Computing (ISORC'98), Kyoto, 335-340, IEEE Computer, 1998.

[7] *Herrmann, P.; Krumm, H.*: A Framework for Modeling Transfer Protocols. Computer Networks, 34 (2000) 2, 317–337.

[8] *Herrmann, P.; Krumm, H.*: Formal Hazard Analysis of Hybrid Systems in cTLA. Proc. 18th IEEE Symposium on Reliable Distributed Systems (SRDS'99), Lausanne, 68–77, IEEE Computer Society Press, 1999.

[9] *Herrmann, P.; Krumm, H.*: A Framework for the Hazard Analysis of Chemical Plants. Proc. 11th IEEE International Symposium on Computer-Aided Control System Design (CACSD2000), Anchorage, 35-41, Omnipress, 2000.

[10] *Herrmann, P.; Noel, M.*: Formal Development of a Distributed Control System for Road Marking Machines. Proc. 8th International Conference on Telecommunication Systems - Modeling and Analysis, Nashville, 430–437, ATSMA, 2000.

[11] *Huuck, R.; Lakhnech, Y.; Lukoschus, B; Urbina, L.; Engell, S.; Kowalewski, S.; Preußig, J.*: Integrating Timed Condition/Event Systems and Timed Automata for the Verification of Hybrid Systems. Journal of Parallel and Distributed Computing Practices. Vol. 1, No.2, 45-60, 1998.

[12] *Kowalewski, S.; Bauer, N.; Preußig, J.; Stursberg, O., Treseler, H.*: An Open Tool Architecture for the Formal Verification of Logic Controllers in Processing Systems. Preprints IFAC World Congress, Bejing, P.R. China, 1999.

[13] *Kowalewski, S.; Engell, S.; Preußig, J.; Stursberg, O.*: Verification of Logic Controllers for Continuous Plants Using Timed Condition/Event-System Models. Automatica 35(3), 505–518, 1999.

[14] *Kowalewski, S.; Lakhnech, Y.; Lukoschus, B; Urbina, L.*: On the composition of condition/event systems. Proc. 4th Workshop on Discrete Event Systems (WODES'98), Cagliari, Italien, August 1998, IEE, London, 349-354.

[15] *Kowalewski, S.; Preußig, J.; Engell, J.; Huuck, R.; Lakhnech, Y.; Urbina, L.*: Analyse zeitbewerteter Bedingung/Ereignis-Systeme mittels Echtzeitautomatentools. 5. Fachtagung Entwurf komplexer Automatisierungssysteme (EKA'97), Braunschweig, Mai 1997, Band 1, 180-194 (in German).

[16] *Kowalewski, S.; Stursberg, O.; Treseler, H.*: Diskrete Modellierung verfahrenstechnischer Prozesse zur Steuerungsverifikation. at-Automatisierungstechnik 46, Heft 4, 180-188, 1998 (in German).

[17] *Krogh, B.H.*: Condition/Event Signal Interfaces for Block Diagram Modeling and Analysis of Hybrid Systems. Proc. 8th International Symposium on Intelligent Control Systems, Chicago, USA, 1993, 180-185.

[18] *Kurki-Suonio, R.*: Hybrid Models with Fairness and Distributed Clocks, in: Proc. Workshop on Theory of Hybrid Systems, Eds: R. L. Grossman, A. Nerode, A. Ravn, and H. Rischel, Lynby, Springer-Verlag, LNCS 736, 103-120, 1993.

[19] *Lamport, L.*: Hybrid Systems in TLA$^+$, in: Proc. Workshop on Theory of Hybrid Systems, Eds: R. L. Grossman, A. Nerode, A. Ravn, and H. Rischel, Lynby, Springer-Verlag, LNCS 736, 77-102, 1993.

[20] *Lamport, L.*: The Temporal Logic of Actions, ACM Transactions on Programming Languages and Systems, vol. 16, no. 3, 872-923, 1994.

[21] *Sreenivas, R.S.; Krogh, B.H.*: On Condition/Event Systems with Discrete State Realizations. Discrete Event Dynamic Systems: Theory and Applications 2(1), 1991, 209–236.

Manuskripteingang: 27. November 2000