Proceedings of the 2000 IEEE International
Symposium on Computer-Aided Control System Design
Anchorage, Alaska, USA • September 25-27, 2000

MA2-3 10:10

A Framework for the Hazard Analysis of Chemical Plants[1]

Peter Herrmann and Heiko Krumm
Universität Dortmund, Fachbereich Informatik, D-44221 Dortmund
Email: {herrmann|krumm}@ls4.cs.uni-dortmund.de
Phone: +49-231-7554674, Fax: +49-231-7554730

Abstract

Transposing the notion of software frameworks to the abstraction level of formal specifications and verifications, we developed a framework supporting the formal hazard analysis of chemical plants. It provides generic specification modules for the description of safety properties, specification modules for the description of plant models, and theorems stating that certain subsystem structures of the plant model imply certain safety properties. Using the framework for hazard analysis, one firstly describes the plant and its control equipment as a composition of framework module instances. Secondly, one expresses the different safety properties of interest by parameterized framework modules. Finally, a safety property is proven when an appropriate theorem instance of the framework can be found. Thus, the framework facilitates the formal modeling. Moreover, the efforts for formal verifications are reduced drastically since framework theorem instances can replace explicit proofs.

The framework utilizes modular temporal logic specifications supported by the specification language cTLA which is a variant of Lamport's temporal logic of actions TLA and in particular is devoted to the compositional description of process systems.

1 Introduction

Modern chemical plants are complex hybrid systems consisting of various process and control equipment. Continuous mass and energy flows of chemical production processes are controlled by event-discrete real-time hardware and software systems. As a rule, the operation of chemical plants entails diverse risks. Therefore, the design of plants has to be accompanied with careful and responsible hazard analysis procedures. Since comprehensive and easy-applicable formal methods are not yet available, the analysis procedures are mainly based on informal discussions following the well-established "Hazard and Operability Studies" approach (HazOp) [16]. The potential value of formal methods, however, is appreciated and several approaches were proposed applying tool-assistance, formal modeling, and formal model based reasoning to tasks and subtasks of hazard analysis. Thus, expert systems can provide automated procedure assistance [4, 6, 19, 22] and simulation tools provide helpful insight into a system under design [5, 17, 23].

Moreover, approaches exist which support the direct formal analysis in order to achieve formal safety proofs using qualitative equation models [3, 23], petri net models [20], and temporal logic [18]. The approaches are related to techniques known from the field of formal hardware and software system verification. As in this field, the experience is gained that formal verification introduces considerable additional costs and is moreover prone to errors. Therefore, tool support is of high interest lowering the costs due to automation and increasing the reliability of proofs due to mechanical reasoning. Corresponding tools were developed using exhaustive state space exploration [21] or symbolic model checking techniques [2, 15, 18]. When analyzing complex systems of practical interest, the tool support, however, is not satisfactory. Fully automated state space exploration tools fail due to the very high number of reachable system states to be managed. Symbolic theorem proving tools need active and intelligent user guidance providing suitable proof structures. Consequently, substantial efforts are needed either to develop abstractions and simplifications for automated state space exploration or to design lemmas, strategies, and proof outlines to be supplied to symbolic theorem provers.

Our general approach is also based on symbolic logic theorem proving and uses modular temporal logic specifications [10]. Since we aimed to the efficient verification of complex practical systems, we looked for additional means in order to complement the support provided by an appropriate formal modeling technique and its tools. Firstly, we focused on the modular description of hybrid systems obtaining re-usable specification modules (generic process type definitions) and verification elements (theorems). This supported the efficient re-use and the structuring of verifications into a series of relatively small subproofs [7]. Secondly, we wanted to facilitate the proof design. Therefore we proposed relatively direct mappings of traditional informal HazOp argumentations to formal proofs [9]. Now, we propose a third complementary approach. It adopts the well-known notion of "frameworks" from software engineering (cf. [11]) and transfers it from the construction of software systems to the construction of formal models and proofs. Software frameworks are devoted to special application domains. They provide rules governing the architecture of software systems and moreover supply modules supporting their efficient composition. Similarly, our specification and verification framework is devoted to the special application domain of chemical plants. Besides of architectural rules it comprises two collections of generic specification modules. One collection contains modules describing components used by chemical plants (e.g., vessels, valves, pumps, mixers) as well as specifications of the components of plant control systems (e.g., sensors, actuators, controllers). Based on these modules specifications are de-

[1]This work was funded by the German research foundation DFG.

0-7803-6566-6/00$10.00©2000 IEEE

veloped in two quite easy steps. Firstly, generic modules are instantiated in order to model components of a particular plant. Secondly, the entire plant is specified by a composition of these module instances. The other specification module collection contains modules modeling safety properties to be kept by a plant (e.g., no excess pressure above a certain limit in a vessel). The use of two groups of specification modules addressing two different abstraction levels (i.e., plant component models and abstract safety properties) is an extension to software frameworks which supply only software modules but do not provide formal specification elements. Moreover, there is a second important extension. In addition to these two groups of specification modules, our framework provides verification elements facilitating formal proofs that chemical plants keep certain safety properties. A collection of generic theorems is included where each theorem instance ensures that a safety property is provided by specific subsystems of plant models. Since we already proved the validity of the theorems, framework users only need to select a suitable theorem, to instantiate it according to the parameter settings of the plant models and safety property specifications, and, finally, to perform some simple checks guaranteeing the consistency of the theorem instance and plant model. Therefore, the framework does not only facilitate the efficient construction of formal models and specifications, but also supports the formal reasoning directly, since formal proofs can relatively easily be combined from theorems of the framework. For example, we specified the example plant used in this paper and proved 33 safety properties within four days.

Our approach roots in an approach for the formal verification of complex data communication protocols (cf. e.g. [8]) introducing the temporal logic specification language cTLA for the description of event-discrete distributed and concurrent process systems. cTLA is based on Leslie Lamport's Temporal Logic of Actions TLA [14]. Particularly, it supports the modular description of process types. System descriptions can be composed from implementation-oriented as well as from constraint-oriented processes. Process composition has the character of superposition (cf. [12]), i.e., relevant properties of processes and subsystems are also properties of the embedding system. Therefore, structured verification can be applied, i.e., for the verification of a system property it is sufficient to prove that a subsystem exists which has the property. Moreover, since cTLA facilitates the description of systems as appropriate constraint compositions, specifications can reflect the logical connections and dependencies of systems. Therefore, mostly small subsystems can be identified supporting the structured verification of interesting system properties. Because of these features, cTLA proved to be very well suited for the establishment of specification and verification frameworks. We developed extensions for the modeling of real-time and continuous properties which keep the superposition character of composition [7]. Based on these extensions, frameworks for hybrid systems like the framework for the formal hazard analysis of chemical plants can be established.

In the remainder we firstly concentrate on the essentials of our approach. We discuss the framework structure and describe an example system. Moreover, a short comparison with traditional HazOp shall clarify the approach. Thereafter, we enter into more details. The specification technique cTLA is outlined and a sketch of a formal verification is given verifying a safety property of the example system.

2 Framework

The framework which is available in the World Wide Web (`ls4-www.cs.uni-dortmund.de/RVS/P-HYSYS`) comprises a set of rules concerning the architecture of specifications, a set of specification modules, and a set of theorems. The architecture rules focus on plant models which are composed from instances of generic specification modules. The rules refer to the different module groups in connection with the different types of real plant components. They govern the modeling of plant components by compositions of specification module instances. In fact, each plant component is modeled in a constraint-oriented way by a composition of several modules, since the plant model shall have a relatively fine-grained structure in order to support the proof of system properties from small subsystems. For instance, the framework contains modules each describing only a single aspect of a vessel in the chemical plant (e.g., the amount of a liquid in the vessel, the temperature of the liquid, the pressure in the vessel). The specification of the entire vessel is derived by composing these modules. Moreover, there are rules applying to the architecture of a plant model as a whole. These rules describe, how plant component models are coupled to form the global plant model. Due to the constraint-oriented modules, plant models have a three layered hierarchical structure. A plant model is a composition of subsystems which are compositions of plant component behaviour constraints. This hierarchical structure facilitates the design of models. With respect to verification, however, it is not useful, since, here, we want to form subsystems which consist of small subsets of the constraints of a series of components. Since the composition operation of cTLA is associative and commutative, hierarchical plant models can easily be transformed into constraint-oriented descriptions which are suitable for verification.

The set of specification modules of the framework is split into two collections, one for plant models and the other for safety property specifications (cf. Fig. 1). Safety property specifications have a very simple structure. Each property is described by one instance of a specification module. The composition of the instances corresponds to the logical conjunction of properties and acts as a safety property specifi-

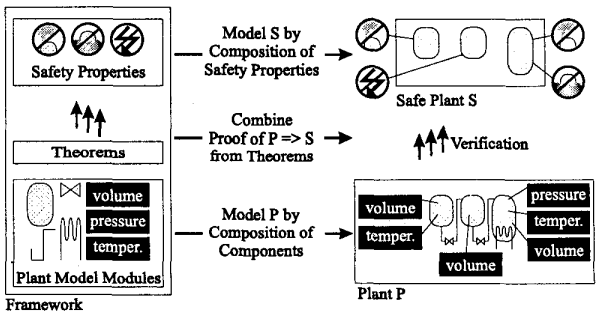

Figure 1: The framework for the formal hazard analysis of chemical plants

cation. A safety property module expresses mainly that a certain limit is observed (e.g., flow, power, pressure, volume, or temperature in a plant component does not exceed or fall below a critical limit). Additionally, there are modules excluding critical combinations of operational component states (e.g., below a certain flow, a pump will be switched off in order not to run dry; below a certain vessel volume, the heating will be switched off in order to prevent overheat).

The set of plant model modules is structured into two main groups. One is devoted to plant components, the other to malfunctions. The plant component group contains four subgroups of component behaviour constraints: continuous, discrete, sensor, and actor constraints. For instance, the continuous volume of a vessel content obeys a constraint of type *VesselVolume* (cf. Sec. 5) which models the current volume depending on the amount of liquid running into resp. draining from the vessel. A constraint of type *SensorAlarm* describes that a sensor will create an alert when the monitored value exceeds a limit. It is an example of a hybrid constraint, since the monitoring of a continuous value is combined with a discrete reaction. The modules of the malfunctions group specify component malfunctions to be tolerated by the plant. Examples are leakage or jam of pipes and valves as well as failures of active components like pumps and heatings.

The theorems of the framework are implications of the form "plant-constraint-subsystem ∧ malfunction-subsystem ⇒ safety-property". They assert that certain plant subsystem structures guarantee safety properties even under the presence of malfunctions. For instance, a very simple subsystem corresponds to a controlled pump where the control switches off the pump in case of low flow. This subsystem guarantees the safety property that there exists a maximum duration limit for time intervals where the pump operates below a critical flow limit. Mostly, however, the subsystems tend to be more complex. Often plant components can interfere with many other and even very distant components. This would lead to very large subsystems. Since there are numerous variants for the structure of large subsystems, the framework would consequently have to contain numerous theorems. We solve this problem by an extended form of theorems. We add a conjunct to the left side of the implication which describes safety requirements of the environment of subsystems. Since these safety requirements are expressed by means of the specification module collection for safety properties, theorems have the form "plant-constraint-subsystem ∧ malfunction-subsystem ∧ safety-property ∧ ... ∧ safety-property ⇒ safety-property" and one can transitively use other theorems for the validation of the safety requirements of the subsystem environment. For instance, consider a set of serially connected tanks. If one can prove that the first tank is never empty and that each other tank does not run empty if its predecessor is not empty, one can infer, that all tanks never are empty. This, however, does not work in the case of circular dependencies where we look for safety property hierarchies.

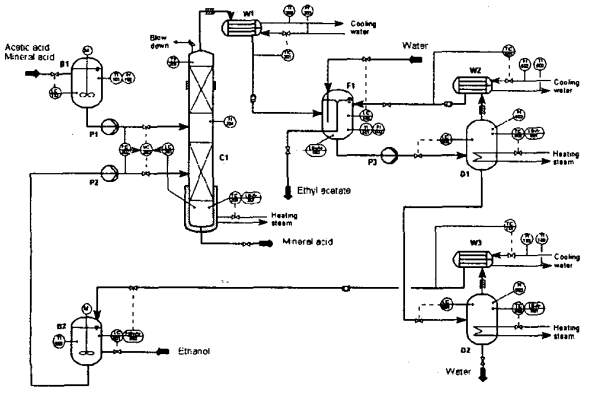

Figure 2: Sketch of the example system

3 Example

To explain our approach, we introduce in Fig. 2 a hybrid technical plant (cf. [5]) for the production of ethyl acetate, an ester used as a solvent. The ethyl acetate is produced in the distillation reactor C1 by esterification of acetic acid and ethanol. The acetic acid and a catalyst are stored in the vessel B1, while the ethanol is held in B2. The catalyst is removed from C1 into a waste tank while the distilled ethyl acetate is condensed in the heat exchanger W1 and drops into the extractor F1. Here, the ethyl acetate is purified by washing it with water into the final product which is removed.

The remaining solution of ethyl acetate, ethanol, and water is pumped into the vessel D1, where the ethyl acetate is removed again by distillation. It condenses in W2 and is fed back into the extractor F1. The remaining mixture of ethanol and water is separated in the distillation vessel D2. The ethanol is distilled, condensed in the heat exchanger W3 and fed back to the storage vessel B2, while the remaining water is removed from D2.

4 Comparing our approach to HazOps

Below, we will outline the differences between traditional Hazard and Operability Studies (HazOps) [16] and our approach by means of the example plant.

In a HazOp, the plant is investigated for possible failures and their consequences. For instance, a failure may occur if the team driving the plant forgets to add acetic acid and the catalyst into the storage vessel B1. Consequently, the vessel eventually becomes empty. This leads to the pump P1 running dry which might cause a damage of this pump. Furthermore, since the controller UC203 guarantees a certain ratio of acetic acid and ethanol feeds into the reactor C1, the valve between pump P2 and C1 is closed. Thus, P2 pumps against a closed valve causing excess pressure. As well, the reactor C1 eventually becomes empty. Thus, the heating of C1 has to be switched off in order to prevent damages of the heating pipes. Likewise, all components of the plant have to be investigated for failures. As a result, counter-measures like controllers switching off the pumps in case of low volume or excess pressure are suggested.

In our framework approach we do not look for failures but try to prove safety properties stating that failures in the plant are not possible. If a proof succeed, we can rule out

system failures due to design errors. Otherwise, we have to take counter-measures which are guided by the framework theorems. For example, we want to rule out that the heating of C1 may be damaged due to too low volume of liquid in the reactor. We prove safety properties by theorems of the framework. In our example the framework supplies two appropriate theorems. One theorem states that the environment of the reactor guarantees that the volume of liquids does not fall below a certain minimum value. As outlined above, this property does not hold since the controller UC203 may close the input valves if vessel B1 runs empty. The other theorem states that the desired safety property holds if the reactor contains a sensor signalling low volume and a controller switching off the heating after this signal and keeping it off until the volume exceeds the minimum again. Since the sensor and controller LS-A-207 fulfill these demands, we can prove by this theorem that the heating of C1 cannot be damaged due to low volume of liquids if LS-A-207 does not fail (cf. Sec. 6).

5 cTLA

TLA [14] is a linear time temporal logic describing safety and liveness properties of state transition systems by means of canonical formulas. To facilitate the understanding of specifications, cTLA [10] omits the canonical parts of TLA formulas. It is oriented at programming languages and introduces the notion of processes. A specification is structured into modular definitions of process types. An instantiation of a process type forms a process which either has the form of a simple process or that of a process composition. Simple processes, which directly refer to state transition systems, are used to model single plant components resp. safety property modules of the framework (cf. Sec. 2).

Fig. 3 depicts the example of three very simple process types. An instance of the type *ReactOnAlarm* (top of Fig. 3) is used to specify the controller LS-A-207 in the example plant (cf. Fig. 2) describing that the controller reacts on a low volume alarm in reactor C1 by switching off the heating of C1. The header of *ReactOnAlarm* declares the type name and generic module parameters (e.g., `initialstate`). These parameters facilitate to model a spectrum of similar but different processes by a single process type specification.

The process type body defines the state transition system which corresponds to an instance of the type. The state space is defined by the state variable `vastate`. The initial condition `INIT` is a condition over state variables and defines the set of starting states. Finally, in the body actions are defined which model the state transitions (e.g., `alarm`, `switch`). An action is a predicate on a pair of an current state and a successor state and models a set of state transitions. The state variables which refer to the successor state occur in the so-called primed form (e.g., `vastate'`). An action can have action parameters. The disjunction of the actions forms the next state relation of the process. In the course of time, a process may perform action steps (i.e., it changes its state in accordance with an action) or stuttering steps (i.e., it does not change its state while the environment performs a state transition).

According to [1], real-time is represented by means of a real-valued state variable `now` which is incremented lively

```
PROCESS ReactOnAlarm
  (initialstate : {"ready","alarm"})
BODY
  VARIABLES
    vastate : {"ready","alarm"}; current alarm state
  INIT ≙ vastate = initialstate;
  ACTIONS
    alarm ≙ signal alarm
      vastate' = "alarm";
    switch ≙ react by switching an actor
      vastate = "alarm" ∧ vastate' = "ready";
END

PROCESS ReactMaxTime ( maxtime : real )
BODY
  ACTIONS
    switch;
  V MAX TIME switch : maxtime;
END

PROCESS VesselVolume ( capacity : real ;
                       initvolume : real )
BODY
  VARIABLES
    vvolume : real; current volume
  INIT ≙ vvolume = initvolume;
  ACTIONS
    CONT (INPUT  inflow, outflow : real;
          OUTPUT volume, room : real) ≙
      vvolume' =
        Max(Min((inflow - outflow) · (now' - now)
                + vvolume, capacity),0) ∧
      volume = vvolume ∧
      room = Max (0,capacity - vvolume);
END
```

Figure 3: Process Types *ReactOnAlarm*, *ReactMaxTime*, and *VesselVolume*

by a clock action `tick`. Unlike other variables, which are private in exactly one process, `now` can be read by all processes of a system. Additional real-time constructs describe activity retarding and activity forcing real-time constraints. In accordance to [13], one can specify minimum waiting times and maximum reaction times for actions. Moreover, one may refer to volatile and to persistent enabling periods of an action. The maximum reaction time construct is conditional in order to ensure the consistency of process compositions. It forces an action only with respect to periods where the action is enabled and the environment does not block it. For instance, the center of Fig. 3 shows the process type *ReactMaxTime*. It declares an action `switch` with a volatile maximum reaction time. In our example, an instance of this process type is used to model the maximum reaction time of the controller LS-A-207.

Continuous properties of a process are expressed by means of an action with the special name `CONT`. All `CONT`-actions of all processes of a system and the *tick*-action of the clock are assumed to occur simultaneously. Thus, the series of `CONT`-steps of the system approximates the continuous behaviour (cf. [13]). Usually, the `CONT`-actions contain difference equations for continuous state variables. The time difference modeled by an execution of `CONT` is expressed by the difference `now'-now`. The inputs and outputs of con-

```
PROCESS SwitchOffHeatingDueToLowVolume
          (CapC1 : real; ...)
  PROCESSES
    VesselVol : VesselVolume(CapC1, InitVolC1);
    volume of liquid in the reactor C1
    Vaporization : VesselVaporize(Liquiddense,
                                  Gasdense, Enthalpy)
    amount of liquid vaporized in C1
    Heater : Heating(MaxPowC1, IncratePowC1,
                     DecratePowC1);
    heating of the reactor C1
    SensorAlarm : SenseMin(SenseVol);
    sensor: liquid volume in C1 falls below SenseVol
    SensorAlarmTime : SenseMaxTime(Sensetime);
    time constraint for sensor
    SensorRelease : SenseMax(SenseVol);
    sensor: liquid volume in C1 exceeds SenseVol
    SwitchOffHeating : ReactOnAlarm("ready");
    control unit LS-A-207 switches heating off
    SwitchOffHeatingTime : ReactMaxTime(Reacttime);
    time constraint for LS-A-207
    KeepHeatingOff : BlockOnAlarm("block");
    LS-A-207 guarantees that heating is not switched on while
    the amount of liquid is below the minimum

  ACTIONS non-listed processes perform stuttering steps
    CONT (INPUT B1C1vol, B2C1vol,
          OUTPUT C1vol, C1vapliquid, C1vapgas,
                 C1power, C1outflow : real)
    continuous volume flow
      VesselVol.CONT (B1C1vol + B2C1vol,
                      C1vapliquid + C1outflow ;
                      C1vol) ∧
      Vaporization.CONT (C1vol, C1power,
                         C1vapliquid, C1vapgas) ∧
      Heater.CONT ( ; C1power) ∧
      SensorAlarm.CONT (C1volume ; ) ∧
      SensorRelease.CONT (C1volume ; ) ∧ ...;
    Alarm ≙ sensor reports liquid volume below minimum
      SensorAlarm.alarm ∧ SensorAlarmTime.alarm ∧
      SwitchOffHeating.alarm ∧
      KeepHeatingOff.alarm ∧ ...;
    Release ≙ sensor reports liquid volume above minimum
      SensorRelease.alarm ∧
      KeepHeatingOff.release ∧ ...;
    HeatingOff ≙ LS-A-207 switches the heating off
      Heater.off ∧ SwitchOffHeating.switch ∧
      SwitchOffHeatingTime.switch ∧ ...;
    HeatingOn ≙ switch on heating only if it is not blocked
      Heater.on ∧ KeepHeatingOff.switch ∧ ...;
```

Figure 4: Subsystem *SwitchOffHeatingDueToLowVolume*

tinuous processes are modeled by action parameters. As an example, the bottom of Fig. 3 depicts the process *VesselVolume* which is used in the example plant to model the volume of liquids in the reactor C1. The process declares a continuous real-valued state variable `vvolume`. The `CONT`-action declares the continuous inflows and outflows of liquids and lists the difference equation for `vvolume`.

Several constructs stating safety, real-time, and continuous properties may be contained in the same process type definition. Thus, one can specify all relevant properties of a hybrid system component by one process. However, to support a fine-grained constraint-oriented system structure (cf. Sec. 2), we mainly use process types which concentrate on properties of a single sort.

Systems and subsystems are described as compositions of concurrent processes. Each process encapsulates its variables and can change its state by atomic executions of its actions. The system state is the vector of the state variables of the processes. State transitions of the system correspond to simultaneous process actions and process stuttering steps. Each process performs either exactly one action or an stuttering step. Thus, the system actions can be defined by conjunctions of process actions and process stuttering steps. Consequently, concurrency is modeled by interleaving while the coupling of processes corresponds to joint actions. Fig. 4 shows an example of a system specification. The process type *SwitchOffHeatingDueToLowVolume* models a part of the example plant. In the part `PROCESSES` the processes of the system are declared as instantiations of process types. Here, it consists of constraint processes modeling important properties of the reactor C1 (the volume liquid, the amount of ethyl acetate vaporized within a time-unit, and the heating), the sensor signalling low volume in C1, and the controller LS-A-207 which realizes the switching off of the heating in C1 after the volume falls below a minimum value. Moreover, LS-A-207 guarantees that the heating remains off until the amount of liquid recovers. Thereafter, in the part `ACTIONS` the system specification describes the coupling of the processes by the definition of the joint system actions. For instance, the action `HeatingOff` models that, if the controller LS-A-207 reacts on a sensor alarm (action `SwitchOffHeating.switch`), the heating is switched off simultaneously (action `Heater.off`). Furthermore, `HeatingOff` must switch within a volatile reaction time (action `SwitchOffHeatingTime.switch`). The cTLA specification of the example plant is available on the internet (`ls4-www.cs.uni-dortmund.de/RVS/P-HYSYS`).

6 Verification Example

Below, we will exemplify the use of the framework by sketching the formal proof that the heating of the reactor C1 in our example system (cf. Fig. 2) does only work if the

```
PROCESS VesselMinVolumeOrNoPower
          (minvolume : real)
BODY
  VARIABLES
    vvolume : real; current volume
    vpower : real; current power
  INIT ≙ vpower = 0;
  ACTIONS
    CONT (OUTPUT volume, power : real) ≙
      IF (vvolume < minvolume)
      THEN vpower' = 0 ELSE vpower' ≥ 0 ∧
      power = vpower ∧
      volume = vvolume;
END
```

Figure 5: Safety Constraint *VesselMinVolumeOrNoPower*

amount of liquid in C1 exceeds a minimum value. Thus, we guarantee that the heating pipes are covered with liquid if the heating runs and therefore cannot run too hot. In Fig. 5 we sketch the process *VesselMinVolumeOrNoPower* which models the safety constraint we want to prove. It contains a process parameter `minvolume` indicating the minimum volume of liquid below which no power must be supplied. The volume of liquid in the reactor is modeled by the variable `vvolume` while `vpower` describes the amount of power supplied by the heating. Initially, no power is supplied at all. The state transitions in this process type are modeled by the continuous action `CONT` stating that power may be supplied only (`vpower'` ≥ 0) if the volume of liquid in the reactor is greater or equal to `minvolume`.

To prove that our example plant fulfills this safety specification, we use the framework theorem *VesselMinOrNoPower* listed in Fig. 6[2]. A framework theorem states that a subsystem specification `Sys` implies a safety constraint (e.g., *VesselMinVolumeOrNoPower*) if the conditions `Pars` and `EnvCond` hold. Similarly to cTLA system processes, `Sys` contains a part `PROCESSES` listing the processes, the subsystem is composed from, and a part `ACTIONS` describing the coupling of the process actions to joint subsystem actions. The subsystem `Sys` is composed from the instance `sohdtlv` of the subsystem process *SwitchOffHeatingDueToLowVolume* (cf. Fig. 4) and the instance `MaxDraining` of the safety process type *LineMaxFlow*. `sohdtlv` describes the subsystem of the plant specification which realizes the safety constraint (i.e., the reactor volume, the amount of liquid vaporized, the heater, the sensor, and the controller LS-A-207). The safety constraint `MaxDraining` models that the environment of the reactor guarantees that at most an amount of `Maxoutflow` liquid drains from the reactor per time unit. The validity of `MaxDraining` is proven separately by another framework theorem.

The condition `Pars` states that the parameters of the processes listed in `Sys` and the safety constraint to be proven are instantiated consistently. Here, `Pars` states that the sensor reacts early in order to guarantee that the heating stops running before the volume of liquid falls below the minimum value `Minvolume`. `Pars` considers the reaction times of the sensor and controller as well as the time the heating needs to cool down. It states that the parameter `SenseVolume`, which indicates the volume of liquid triggering a sensor signal, is greater than the sum of `minvolume`, the amount of liquid vaporized until the sensor and controller react, the amount of liquid vaporized while the heating cools down, and the amount of liquid draining from the reactor until the sensor and controller react and the heating needs to cool down.

By the environment condition `EnvCond` we prove that the processes of the plant specification not listed in `Sys` do not block relevant actions of `Sys`. Here, `EnvCond` guarantees that the environment of `Sys` does not block the actions `Alarm` and `HeatingOff`. Without fulfilling this condition, the environment may prevent the sensor to signal low volume and the controller to switch off the heating spoiling the safety constraint *VesselMinVolumeOrNoPower*.

[2]Fig. 6 shows a shortened form of the theorem. In long form, `Sys` consists directly of the processes listed in *SwitchOffHeatingDueToLowVolume* and `MaxDraining`.

```
THEOREM VesselMinVolumeOrNoPower
LET Pars ≜
SenseVol > Minvolume +
           (MaxPowC1 / (Liquiddense · Enthalpy))
           · (Sensetime + Reacttime) +
           MaxPowC1² / (2 · DecratePowC1 ·
                        Liquiddense · Enthalpy) +
           ((MaxPowC1 / DecratePowC1 + Sensetime
             + Reacttime) · Maxoutflow)
sensor sensitivity guarantees that heating power supply is
stopped before the vessel volume falls below the critical limit
Sys ≜
PROCESSES Subsystem
  sohdtlv : SwitchOffHeatingDueToLowVolume
              (CapC1, InitVolC1, Liquiddense,
               Gasdense, Enthalpy, MaxPowC1,
               IncratePowC1, DecratePowC1,
               SenseVol, Sensetime, Reacttime);
  subsystem of plant containing relevant processes to switch
  off heating due to low volume
  MaxDraining : LineMaxFlow (Maxoutflow);
  maximum flow of fluid out of the vessel

ACTIONS
  CONT ( OUTPUT C1vol, C1vapliquid, C1vapgas,
                C1vappower, C1outflow : real ) ≜
  continuous volume flow
    sohdtlv.CONT
      (CHOOSE x : x ≥ 0, CHOOSE x : x ≥ 0 ;
       no draining through inflow lines from B1 and B2
       C1vol, C1vapliquid, C1vapgas, C1vappower,
       C1outflow) ∧
    MaxDraining.CONT ( ; boutflow );
  Alarm ≜ sensor reports liquid volume below minimum
    sohdtlv.Alarm ∧ MaxDraining.stutter;
  Release ≜ sensor reports liquid volume above minimum
    sohdtlv.Release ∧ MaxDraining.stutter;
  HeatingOff ≜ LS-A-207 switches the heating off
    sohdtlv.HeatingOff ∧ MaxDraining.stutter;
  HeatingOn ≜ switching on heating if it is not blocked
    sohdtlv.HeatingOn ∧ MaxDraining.stutter;

EnvCond ≜
  Enabled(sohdtlv.Alarm) ⇒ Sys.e_Alarm ≠ {} ∧
  Enabled(sohdtlv.HeatingOff) ⇒
    Sys.e_HeatingOff ≠ {};

IN  Pars ∧ Sys ∧ □EnvCond ⇒
      VesselMinVolumeOrNoPower (Minvolume);
```

Figure 6: Theorem *VesselMinVolumeOrNoPower*

In order to prove *VesselMinVolumeOrNoPower*, we replace the process parameters in the theorem by their current values. For example, since both the sensor and the controller react within 0.1 seconds, we instantiate the parameters `Sensetime` and `Reacttime` with 0.1. Afterwards, we check five conditions of the theorem. At first, we have to check that the processes listed in `Sys` (i.e., `sohdtlv`) are also contained in the plant model. Secondly, we check that the safety specifications of `Sys` (i.e., `MaxDraining`) are already proven by means of other theorems. Thirdly, we check that the action coupling of `Sys` is consistent to the action

coupling of the plant model. For instance, the condition on the first parameter of the action `sohdtlv.CONT` in `Sys` models that no liquid may drain through the pipe linking the reactor C1 with the vessel B1. In the plant specification the process modeling the pump P1 guarantees this assumption. Fourthly, we have to prove that the condition `Pars` hold. Due to the replacements of the process parameters this proof is performed by simple mathematical transformations. Finally, we check that the invariant `EnvCond` holds. Since in the plant specification the enabling conditions of the actions `Alarm` and `HeatingOff` only depend on processes in `sohdtlv`, this condition holds, too. One verifies other safety properties in the same manner by choosing suitable framework theorems and performing the five condition checks as described above.

7 Concluding remarks

We introduced the notion of frameworks for the modeling and verification of technical systems and outlined a corresponding framework for the hazard analysis of chemical plants. The framework was successfully applied to the example system (Sec. 3) and other chemical plants. While the general framework approach does not depend on specific specification languages, we profit from the special features of cTLA. It supports the description of hybrid systems. Besides of safety and liveness properties of event-discrete processes, it can express real-time properties and properties of continuous processes. Therefore, integral models comprising plants as well as control hardware and software can be described. This gives — in connection with the superposition property of cTLA's process composition — a very good basis for frameworks in real-time and hybrid system application domains.

References

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real time. In J. W. de Bakker et al., editors, *Real-Time: Theory in Practice*. LNCS 600, Springer-Verlag, 1991.

[2] R. Alur, T. A. Henzinger, and P.-H. Ho. Automatic Symbolic Verification of Embedded Systems. *IEEE Transactions on Software Engineering* 22:181-201, 1996.

[3] C. A. Catino and L. H. Ungar. A model-based approach to automated hazard identification of chemical plants. *AIChE Journal*, 41(3):97–109, 1995.

[4] M. Göring and H. G. Schecker. HAZEXPERT: An integrated expert system to support hazard analysis in process plant design. *Computers Chemical Engineering*, 17:429–434, 1993.

[5] H. Graf and H. Schmidt-Traub. A Model-Based Approach to Process Hazard Identification. In *13th International Congress of Chemical and Process Engineering (CHISA)*, Prague, 1998.

[6] P. Heino, A. Poucet, and J. Soukas. Computer tools for hazard identification, modelling and analysis. *Journal of Hazardous Materials*, 29:445–463, 1992.

[7] P. Herrmann, G. Graw, and H. Krumm. Compositional Specification and Structured Verification of Hybrid Systems in cTLA. In *1st IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC98)*, pages 335–340, Kyoto. IEEE Computer Society Press, 1998.

[8] P. Herrmann and H. Krumm. Modular Specification and Verification of XTP. *Telecommunication Systems*, 9(2):207–221, 1998.

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

[10] P. Herrmann and H. Krumm. A Framework for Modeling Transfer Protocols. To appear in *Computer Networks*, 2000.

[11] R.E. Johnson and B. Foote. Designing reusable classes. *The Journal of Object-Oriented Programming*, 1(2):22–35, 1988.

[12] R. Kurki-Suonio. Hybrid Models with Fairness and Distributed Clocks. In R. L. Grossmann et al., editors, *Workshop on Theory of Hybrid Systems*, pages 103–120, Lyngby. LNCS 736, Springer-Verlag, 1993.

[13] L. Lamport. Hybrid Systems in TLA$^+$. In R. L. Grossmann et al., editors, *Workshop on Theory of Hybrid Systems*, pages 77–102, Lyngby. LNCS 736, Springer-Verlag, 1993.

[14] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, 1994.

[15] K. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Springer International Journal of Software Tools for Technology Transfer*, 1(1+2), 1997.

[16] H. G. Lawley. Operability Studies and Hazard Analysis. *Chemical Engineering Progress*, 70(4):45–56, 1974.

[17] N. Leveson. *Safeware: System Safety and Computers*. Addison Wesley, 1995.

[18] S. T. Probst. *Chemical Process Safety and Operability Analysis using Symbolic Model Checking*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA 15213, 1996.

[19] Y. Shimada, K. Suzuki, and H. Sayama. Computer-aided operability study. *Computers Chemical Engineering*, 20(6/7):905–913, 1996.

[20] R. Srinivasan and V. Venkatasubramanian. Petri Net-Digraph models for automating HAZOP analysis of batch process plants. *Computers Chemical Engineering*, 20:719–725, 1996.

[21] O. Stursberg, H. Graf, S. Engell, and H. Schmidt-Traub. A concept for safety analyses of chemical plants based on discrete models with an adapted degree of abstraction. In *4th International Workshop on Discrete Event Systems (WODES)*, Cagliari, 1998.

[22] R. Vaidhyanathan and V. Venkatasubramanian. Experience with an expert system for automated HAZOP analysis. *Computers Chemical Engineering*, 20:1589–1594, 1996.

[23] A. Waters and J. W. Ponton. Qualitative simulation and fault propagation in process plants. *Chemical Engineering Research Descriptions*, 67:407–422, 1989.