Monitoring Dynamical Signals While Testing Timed Aspects of a System

Goran Frehse, Kim G. Larsen, Marius Mikučionis, Brian Nielsen

Monitoring Dynamical Signals while Testing Timed Aspects of a System

Goran Frehse[1], Kim G. Larsen[2], Marius Mikučionis[2], and Brian Nielsen[2]

[1] Verimag (UJF-CNRS-INPG), 2, av. de Vignate, 38610 Gieres, France
goran.frehse@imag.fr
[2] Department of Computer Science, Aalborg University
Selma Lagerlöfs Vej 300, 9220 Aalborg Øst
{ kgl, marius, bnielsen } @cs.aau.dk

Abstract. We propose to combine timed automata and linear hybrid automata model checkers for formal testing and monitoring of embedded systems with a hybrid behavior, i.e., where the correctness of the system depends on discrete as well as continuous dynamics. System level testing is considered, where requirements capture abstract behavior and often include non-determinism due to parallelism, internal counters and subtle state of physical materials. The goal is achieved by integrating the tools Uppaal [2] and PHAVer [3], where the discrete and hard real-time aspects are driven and checked by Uppaal TRON and strict inclusion of dynamical trajectories is verified by PHAVer. We present the framework, the underlying theory, and our techniques for integrating the tools. We demonstrate the applicability on an industrial case study.

1 Introduction

Timed automata (TA) is a convenient and expressive modelling language for expressing state- and time-dependent discrete behavior such as time constraints on event occurrences. In particular the Uppaal-language has proven useful and expressive in a large number of case studies. The editing, simulation and analysis of Uppaal-TA models is supported by the Uppaal model-checking tool. Similarly, (online) model-based testing is implemented in the Uppaal TRON tool [7].

However, TA cannot directly capture and describe continous behavior, which is normally abstracted away. When this cannot be done, a workaround may be to model discrete approximations; these may however be cumbersome, inaccurate and significantly degrade the performance of the analysis.

In contrast, (linear) hybrid automata ((L)HA) allows continuous evolutions (trajectories) to be described directly through (linear) differential equations associated with the locations of the automata. PHAVer [3] is a model-checker which provides exploration and analysis capabilities for a rich class of hybrid automata through incrementally refined over-approximation of the trajectories. However, for purely timed and sophisticated discrete behavior its performance

cannot compete with timed automata tools like UPPAAL. Furthermore, its language contains none of the advanced feature of UPPAAL (different types of communication channels, committed locations, C-like instructions on transitions, etc.), which are highly useful for effective modelling of control software.

Testing of hybrid systems is not new, e.g. Reactis Validator is based on a hardware-in-the-loop simulation, where the requirements are expressed as assertions on a Simulink model [9]. Osch provides a formal framework [11] and a prototype tool based on TorX [10] architecture and hybrid χ simulator [12]. Both frameworks are based on a simulation, thus only a deterministic behavior with some relative deviation is allowed. Our framework is unique in a sense that it allows imprecision in timed and dynamical behavior (clock drifts, measurement imprecision, limited state observability, parallelism, abstract requirements) by means of non-determinism in a specification model, i.e. the user can explicitly specify ambiguity of any aspect: bounds on timing of events, dynamical variable values, their derivatives, internal/unobservable transitions. This approach is consistent in treating the imprecision in both time and continuous signals, so the ideas from UPPAAL TRON testing framework can be carried to PHAVER with just a few modifications. Consequently, the conformance check is as easy as inequality check of observed outputs with symbolic state bounds, the test is sound (if the test fails then IUT definitely does not conform to specification), but not necessarily exhaustive (some faults may escape detection due to discrete sampling and measurement imprecision).

Our goal is therefore to combine the UPPAAL TRON and PHAVER tools to construct a tool environment that supports effective and efficient testing and monitoring of control systems that contains both significant timed actions and continuous trajectories.

Model-based *monitoring* is a passive observation of a (black-box) system executing in its operating environment, and checking whether the observed behavior is permitted by (conforms-to) the specified behavior in the model. *Model-based testing* also uses the model to generate test input sequences to stimulate the system, and replaces (part of) the system's environment. Using *online testing* the system is stimulated and evaluated interactively and in real-time. Our goal is to extend the framework of online testing of real-time systems presented in [7] to hybrid systems. UPPAAL TRON simultaneously stimulates and evaluates the system behavior, and can be configured to perform either or both tasks, e.g., work solely as a monitor, while PHAVER provides sound over-approximation of continuous dynamics.

Fig. 1 shows a simple system setup where a plant is controlled by an embedded system. The controller is a digital device running an embedded program which inputs (samples) the sensor values and outputs actuations to the plant.

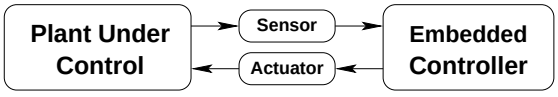


Fig. 1: Hybrid system setup.

The proposed framework will typically be used to monitor and test the behavior of the controller, but can in principle equally well be used to evaluate whether the assumed behavior captured and represented by models of the plant are correct, as only a black-box is assumed.

Fig. 2 shows an arrangement of tools from [6] we use in our test setup: the emulator is a test generator that to the IUT plays the role of an environment (plant under control), the monitor is a test oracle that monitors the observable input/output interaction and decides whether the behavior of IUT is conforming.

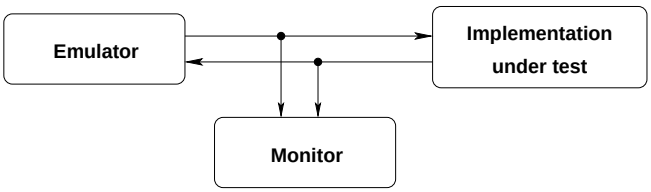


Fig. 2: Test setup.

The main solution idea is to run the two tools in parallel, i.e., UPPAAL TRON for test generation (environment emulation) and monitoring discrete and timed behavior and PHAVER monitoring the continuous behavior. Thus each tool evaluates a part of the overall model.

For this to work, the two tools must be synchronized with respect to real-time and observed events. Moreover, as the behavior in most complex models depends on both aspects, the sub-models also need to exchange information to trigger behavior changes. For instance, when the timed automaton executes an action or switches location the HA model may need to evaluate different trajectories. Similarly, the values (or crossing of thresholds) may be of importance to the possible actions of the TA.

Our contributions are as follows: monitoring dynamical behavior against nondeterministic models, a modeling pattern to keep UPPAAL TA models in synchrony with LHA so that timed models would focus on discrete and timed aspects and hybrid automata mostly on dynamics, a test adapter framework that allows the tools to exchange synchronization events thus enabling the models to "communicate" during online testing, demonstration how our technique and tool can be applied to an industrial case study [8].

Requirement Models. The interaction of discrete events and continuous, time-driven dynamics can be efficiently modeled by a so-called *hybrid automaton* [1]. A hybrid automaton $H = (Loc, Var, Lab, Inv, Flow, Trans, Init)$ consists of a graph in which each vertex $l \in Loc$, also called *location* or *mode*, is associated via $Flow(l)$ with a set of differential equations (or inclusions) that defines the time-driven evolution of the continuous variables. A *state* $s \in Loc \times \mathbb{R}^{Var}$ consists of a location and values for all the continuous variables Var. The edges of the graph, also called *discrete transitions* $Trans$, allow the system to jump between locations, thus changing the dynamics, and instantaneously modify the values of

continuous variables according to a *jump relation* μ. The jumps may only take place when the values of the variables are within the domain of μ. The system may only remain in a location l as long as the variable values are in a range called *invariant* $Inv(l)$ associated with the location. All behavior originates from one *initial state* $Init$.

An *execution* or *trajectory* of the automaton is a sequence of discrete jumps and pieces of continuous trajectories according to its dynamics, and originates from the initial state. *Linear Hybrid Automata* are hybrid automata in which invariants, guards and initial states are given by linear constraints over the variables, jump relations are given by linear constraints over the variables before and after the jump, and the flow constraints are linear constraints over the derivatives only (must not depend on the state variables). Fundamental properties such as reachability are undecidable for (linear) hybrid automata in general. Tools like PHAVer use polyhedra for symbolic state computations, and the operators used in reachability are of exponential complexity in the number of continuous variables.

A *Timed Automaton* is a hybrid automaton with the following restrictions:

- The continuous variables are *clocks*, i.e., their time-derivative is equal to one.
- Invariants and guards are conjunctions of constraints on a single clock, i.e., of the form $\bigwedge_i x_i \bowtie c_i$, where the x_i are clock variables, $\bowtie \in \{\leq, <, =, >, \geq\}$ and the c_i are integer constants.
- The jump relations are *resets*, i.e., each transition may set a subset of the variables to an integer value.

Timed automata have the advantage over general hybrid automata that fundamental properties such as reachability are decidable, and efficient (polynomial complexity) operators are known for symbolic state computation.

Conformance. The formal characterization of correctness is a natural extension of rtioco conformance relation from [7]. We refer to [12] for formal details about hybrid conformance and impact of discrete sampling.

Definition 1 specifies the relation between IUT p and a system specification s represented by states $\langle e, p\rangle$ and $\langle e, s\rangle$ which are composed of IUT model state s and environment model state e.

Definition 1 Relativized timed input/output conformance relation [7]. *$p, s \in \mathcal{S}$ and $e \in \mathcal{E}$ are input-output compatible:*

$$p \ \mathsf{rtioco}_e \ s \stackrel{def}{=} \forall \sigma \in \mathsf{TTr}(e).\mathsf{Out}(\langle e, p\rangle \ \mathsf{after} \ \sigma) \subseteq \mathsf{Out}(\langle e, s\rangle \ \mathsf{after} \ \sigma) \qquad (1)$$

where $\mathsf{TTr}(e)$ *is a set of timed traces generated by* e*, operator* after *denotes reachable states after a trace is executed,* $\mathsf{Out}(T)$ *denotes a set of possible outputs from a set of states* $T \subseteq \mathcal{E} \times \mathcal{S}$*.*

Intuitively the definition says that in order to establish the conformance between IUT state p and specification state s we have to do the following: generate

a trace σ from the environment specification e, execute the trace σ on both IUT composed with environment ($\langle p, e\rangle$) and a complete system specification ($\langle s, e\rangle$), and then check whether the output produced by IUT is included in the outputs of specification. Explicit treatment of the environment model e allows to test the IUT in relation to specific environment.

For hybrid monitoring we adapt the hioco relation proposed by [11]:

Definition 2 Hybrid input/output conformance relation [12]. *Let $p, s \in \mathcal{S}$ be input-output compatible implementation and specification respectively, environment $e \in \mathcal{E}$, then:*

$$p \ \mathsf{hioco}_e \ s \stackrel{def}{=} \forall \sigma \in traces(\langle s, e\rangle) \ . \ \mathsf{Out}\big(\langle p, e\rangle \ \mathsf{after} \ \sigma\big) \subseteq \mathsf{Out}\big(\langle s, e\rangle \ \mathsf{after} \ \sigma\big) \wedge (2)$$
$$\mathsf{traj}(\langle p, e\rangle \ \mathsf{after} \ \sigma) \subseteq \mathsf{traj}(\langle s, e\rangle \ \mathsf{after} \ \sigma) \quad (3)$$

The definitions means that in order to establish the conformance relation, one must consider all specification traces (discrete inputs and outputs intermingled with continuous time trajectories) and check that resulting responses from implementation (discrete outputs and trajectories) are included in the specification. The first part of definition (2) can be checked by UPPAAL TRON as it is concerned with discrete I/O and is compatible with rtioco where trajectories are replaced with time delays, and PHAVER is used to monitor the second part (3). We check (monitor) only the output trajectories, while the original hioco from [11] has additional operators for composing continuous input and output trajectories.

Online Test Algorithm. Algorithm 1 is an abstract version of the timed online testing algorithm which generates sound and exhaustive tests [7]. The algorithm is based on maintaining a current estimate of system state S and consists of three parts: generating an input action based on S, performing a time delay, checking that the output is consistent with the current state estimate, and potentially restarting the test.

2 Monitoring Trajectories

The monitoring of continuous dynamics is implemented by supplementing the action and delay cases of Algorithm 1 with PHAVER symbolic computations in a parallel process, where the continuous output signals are discretized by periodic sampling and checked that the values are included in a model behavior.

We formally define the operators included in PHAVER for monitoring and testing and discuss their implementation for linear hybrid automata. Let s, s' be hybrid states, i.e., each a pair of a discrete location and a valuation of the continuous variables. We consider the set of labels (actions) Lab of the automaton to consist of two disjoint sets of *observable actions* Σ_O and *unobservable actions* Σ_U.

We write $s \xrightarrow{a} s'$ if there is a discrete transition with label a that leads from s to s'. We write $s \xrightarrow{\delta} s'$ for $\delta \in \mathbb{R}^{\geq 0}$ if there is a continuous trajectory leading from

Algorithm 1: Test generation and execution, $OnlineTest(\mathcal{S}, \mathcal{E}, \mathsf{IUT}, T)$.

```
1  S := {⟨s_0, e_0⟩};                          // let the set contain an initial state
2  while S ≠ ∅ ∧ ♯iterations ≤ T do
3      switch Random({action, delay, restart}) do
4          case action                                              // offer an input
5              if EnvOutput(S) ≠ ∅ then
6                  randomly choose i ∈ EnvOutput(S);
7                  send i to IUT, S := S after i;
8          case delay                                          // wait for an output
9              randomly choose d ∈ Delays(S);
10             sleep for d time units or wake up on output o at d' ≤ d;
11             if o occurs then
12                 S := S after d';
13                 if o ∉ ImpOutput(S) then return fail;
14                 else S := S after o
15             else S := S after d;                  // no output within d delay
16         case restart                                          // reset and restart
17             S := {⟨s_0, e_0⟩};
18             reset IUT
19 if S = ∅ then return fail else return pass
```

state s to s' without taking any discrete transitions in between. The notation extends to sets of states S, S' in a straightforward manner.

Let $Post_c(S) = \{s' \mid \exists s \in S, \delta \in \mathbb{R}^{\geq 0} : s \xrightarrow{\delta} s'\}$. Let $Post_d(S, a) = \{s' \mid \exists s \in S : s \xrightarrow{a} s'\}$. For a given alphabet $\Sigma \subseteq Lab$, let $Reach_\Sigma(S)$ be the smallest fixed point of the sequence $P_0 = Post_c(S)$, $P_{k+1} = P_k \cup \bigcup_{a \in \Sigma} Post_c(Post_d(P_k, a))$.

Some of the operators required for testing make explicit use of the time at which events occur. We extend the above operators to explicitly include time by modifying the hybrid automaton: We include a clock variable κ, which has derivative zero and does not change its value during transitions. For the sake of simplicity, we shall abuse notation a little and write (s, κ) for a state of the extended automaton where κ denotes the added clock variable. We annotate the operators on the extended automaton by adding the superscript κ, e.g., $Reach^\kappa_\Sigma(S, \kappa_0)$ denotes the reachable sets of states by taking only discrete transitions with a label in Σ, and starting from an extended state (s, κ) where $s \in S$ and $\kappa = \kappa_0$. We have implemented the following operators:

- $\mathsf{delayTop}(S, \Sigma) = max_\kappa Reach^\kappa_{Lab \setminus \Sigma}(S, 0)$ computes an upper bound on the time the automaton can execute without taking a transition with a label in Σ.
- $\mathsf{delayBottom}(S, \Sigma) = min_\kappa Reach^\kappa_{Lab \setminus \Sigma}(S, 0)$ computes a lower bound on the time the automaton can execute without taking a transition with a label in Σ.

- $\mathsf{transition}(a, x = c) = Post_d(S, a) \cap \{s \mid s(x) = c\}$ computes the states the automaton can be in after taking a transition with label a and having the variable x have the constant value c.
- $\mathsf{observation}(x = c) = S \cap \{s \mid s(x) = c\}$ computes the states the automaton can be in if the variable x has the constant value c.

We will now briefly discuss the complexity of the above operations as they are implemented in PHAVer. Recall that PHAVer represents sets of continuous states as sets of convex polyhedra. The complexity of the above *Post* operators is exponential in the number of continuous variables.

The above operators delayTop and delayBottom make use of the automaton extended with an additional clock. This increases the number of continuous variables in the system by one. Since the complexity of the post-operators grows exponentially with the number of variables, this might significantly affect performance. Once the *Reach* set of the extended system is computed, the upper and lower bounds on κ are found efficiently using linear programming.

If the hybrid automaton has bounded invariants and affine dynamics of the form $\dot{x} = Ax + b$, with A being a matrix and b a vector of rational coefficients, PHAVer overapproximates the behavior by transforming it into a LHA. The transformation splits each location into a number of equivalent locations whose invariants cover the original invariant, and where each invariant is smaller than a given size. For each of the ingoing and outgoing transitions of the original location, equivalent copies are added to the generated locations. In each of the generated locations l_i, the derivatives are overapproximated with the set $\dot{x} \in \{x' \mid \exists x \in Inv(l_i) : x' = Ax + b\}$, which brings the automaton to LHA form. By splitting all locations into parts that are small enough, an overapproximation of arbitrary accuracy can be achieved, albeit for the price of generating a potentially very large number of locations and transitions.

3 Modelling Pattern

Our solution consists of running two tools in parallel: Uppaal TRON for test generation and monitoring discrete and timed behavior and PHAVer monitoring continuous behavior.

We set up both tools to keep track of the state estimate by using symbolic operations on the corresponding requirement models and declare failure if the state estimate of the IUT becomes empty, signaling that the observed behavior is outside the specification.

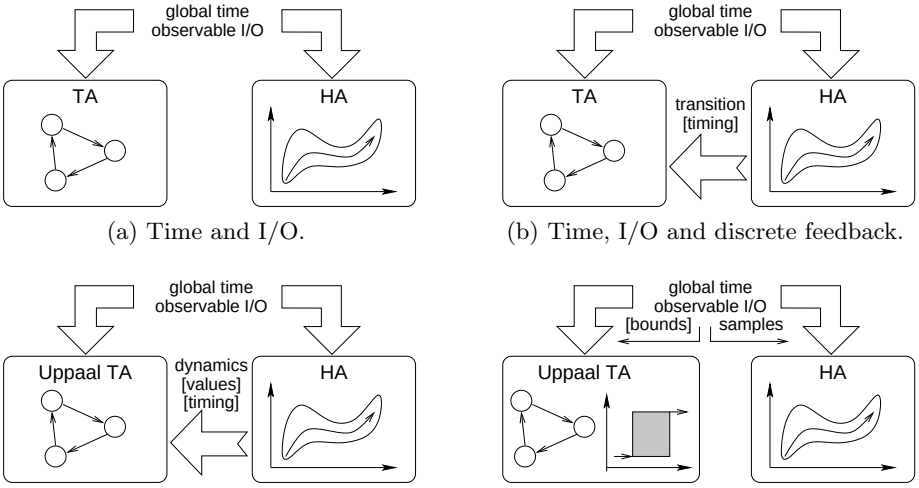
We propose to split the dynamic and timed/discrete aspects into two models:

- A timed automata model responsible for discrete and timely behavior.
- A hybrid automata model handling the dynamic features and using as few discrete aspects as possible.

The reason for the separation is that the model-checking tools are optimised to analyze such aspects on their own. Moreover, performance wise, it is cheaper to

keep the two components separately in parallel than operate on a flat product of them which can be exponentially large.

Depending on the concrete system setup there are various possibilities on how to keep the models in synchrony as shown in Figure 3:



(a) Time and I/O.

(b) Time, I/O and discrete feedback.

(c) Time, I/O and dynamic values.

(d) Time, I/O and direct bounds.

Fig. 3: Various ways to synchronize TA and HA model state estimates.

Time and I/O. Timed and dynamic aspects may be independent of each other (e.g. two parallel processes without interaction in between), and hence no special handling is needed. Fig. 3a shows that timed automata (TA) model state estimate is calculated independently from hybrid automata (HA) model but using the same input/output sequence synchronised on global time reference.

Event synchronization. When an important state change occurs in the HA model it is communicated via a dedicated internal event to TA model. Fig. 3b shows the same TA and HA model setup synchronised on the same timed sequence of input/output events, but in addition it is possible to transfer the information when certain discrete transitions (mode changes) in the HA can and should occur, thus effectively TA and HA models are synchronised on discrete transitions.

Data synchronization. In case of UPPAAL TA models we may handle the dynamic variable values from HA model state estimate as discrete integer data. Fig. 3c shows that parts of HA dynamics is passed to TA model state estimation as discrete data values. However, a special care must be taken to ensure that the data is handled consistently between models.

Over-approximation. Finally, one can use methods from [5] to produce abstractions of hybrid automata in a form of stop-watch timed automata (UPPAAL TRON can handle those too), thus we can use the resulting abstract model to handle timed and discrete aspects and use the hybrid model to

monitor the continuous behavior more precisely. In a similar fashion we propose to model each continuous variable v from HA by two integer bounds v_{low} and v_{upper} in the UPPAAL TA model and update the bound variables. Fig. 3d shows that both models are synchronised on continuous signals: the HA model state estimate is updated with concrete sample values (measurements from the IUT), while UPPAAL TA is updated by bounds which approximate the sample values. In order to use this feature, the UPPAAL TA model needs to be adjusted to handle two boundary values for each continuous signal non-deterministically instead of a single concrete value. As a result, UPPAAL TA handles not just the timed automata aspects but also dynamical behavior in an abstract way: the continuous signal is represented by a rectangular region which is adjusted and synchronised with the rest of the model at runtime.

The over-approximation approach requires only a few changes in the adapter protocol but is general enough to support the other above techniques, thus we focus only on over-approximation. Note that the abstract bounds need not to be updated continuously with each sample if the new sample signal is still within bounds.

4 Architecture

The tools are connected using the UPPAAL TRON test adapter framework. The adapter is a software layer that takes care of translating and delivering input/output actions between tester and an implementation under test (IUT), thus the major parts of it is IUT-specific. In addition, every input/output action is translated into a script and fed into the PHAVER model-checker for monitoring dynamical signals. Fig. 4 shows the setup of tools:

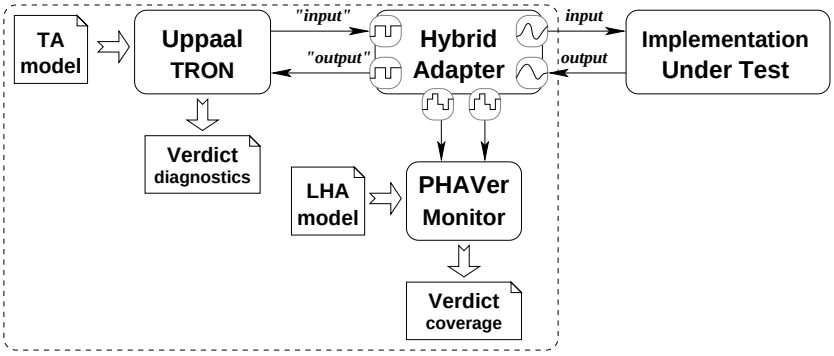


Fig. 4: Framework for online testing of hybrid systems.

– UPPAAL TRON reads a TA model, interacts by abstract representation of discrete input/output action sequences and delivers a verdict at the end of test. Some diagnostics are provided in case the test fails.

– A Hybrid Adapter translates the abstract discrete input and performs the actual input, continuously observes concrete output and translates it into abstract output. Both input and output are reported further to PHAVer in the form of PHAVer script which contain digitised timed input/output sequence.
– PHAVer reads a linear hybrid automata (LHA) model, computes the reachable set of states according to the script and produces a plot of reachable symbolic states until the state set becomes empty and the test consequently fails.

In Fig. 4 all entities in rounded rectangles contain a separate thread and communicates asynchronously so that input/output delivery would not block the test progress.

Fig. 5 shows an example of the events happening during delivery of input and processing of output. The Uppaal TRON and Hybrid Adapter time-stamp events independently of each other and PHAVer is just processing a stream of commands containing already time-stamped input/output events. Each input/output event is time-stamped by a time interval [from; till], where from is the earliest possible moment in time an action has happened, and till is the latest[3].

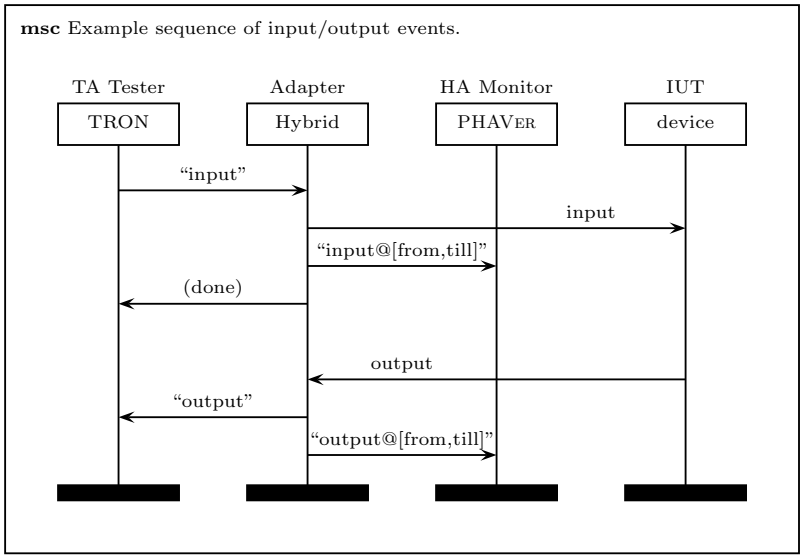


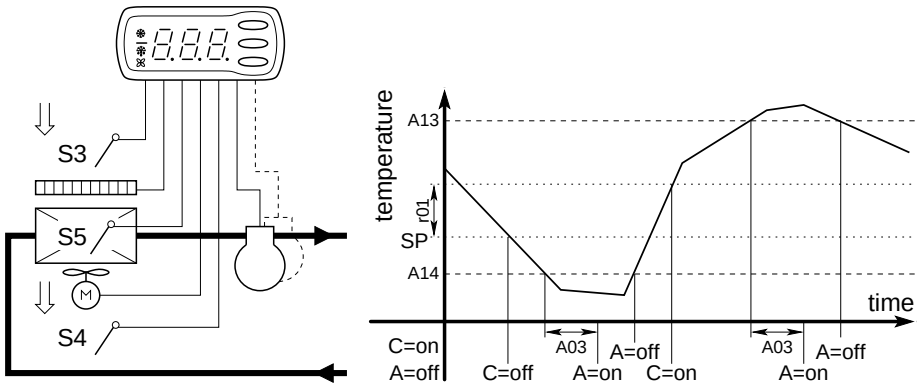
Fig. 5: Message sequence chart of input/output handling.

[3] Inputs are time-stamped before and after sending an event and it is also possible to use communication latency when estimating the time-stamp of output.

5 Application

We use electronic cooling controller EKC204A provided by Danfoss for case study [8] as a hybrid implementation which contain discrete, timely and dynamical aspects.

Temperature Controller. Fig. 6a shows how Danfoss EKC204A temperature controller can be deployed at industrial refrigeration systems. The device can be



(a) Sensor and actuator setup. (b) Temperature regulation principles.

Fig. 6: Danfoss EKC204A temperature controller deployment.

applied in many sensor and actuator configurations, in particular the controller supports the following connections shown in Figure 6a: sensor S3 for outside room air temperature, a relay for controlling defrost heater (electric or gas burner), sensor S5 for temperature on evaporator transferring the heat from air to evaporation of cooling liquid, a relay for fan motor to ensure air rotation around evaporator and/or entire room, sensor S4 for inside room air temperature, actuator relay for compressor used to push cooling liquid in the loop from evaporator to condenser and back. The sensors provide fixed precision number reading and actuators are binary relays having states of "on" and "off".

The controller can be configured to specific strategy using a register database. Fig. 6b shows the main principles behind temperature controller:

- The temperature can vary from $-50°C$ to $+50°C$. The device reads the temperature sensor data and calculates the displayed temperature.
- The objective is to control the temperature between SP and $SP + r01$.
- The compressor (relay C) is used to cool down the air temperature by turning it on ($C = on$) whenever temperature is higher than $SP + r01$ and turning it off ($C = off$) whenever temperature is below SP. Compressor has a requirement to stay on or off for at least some time in order minimise wear and tear of hardware.

– The controller should ring a temperature alarm ($A = on$)whenever the temperature is higher than register $A13$ or lower than register $A14$ for more than $A03$ amount of time and turn it off ($A = \mathit{off}$) when the temperature is within acceptable bounds $[A13; A14]$.

For our purposes we use only inside room air temperature sensor (S4). The controller assumes that the temperature sensors are not perfect (readings usually fluctuate), hence PID-like temperature algorithm is applied to filter and stabilize the room temperature estimate. The temperature is estimated and displayed using equation $T_n = \frac{4 \cdot T_{n-1} + T_s}{5}$, where T_s is a temperature reported by sensor and T_i is temperature estimate at moment i. The temperature is recalculated about once per each second and may vary slightly (the software is soft real-time). The filter is applied only if the temperature change is within $1°C$ and is adjusted immediately otherwise. All other aspects (compressor, alarm and defrost control) depend on the calculated temperature rather than raw sensor values, thus even TA model needs to be aware of this temperature calculation.

Model Setup. The requirements are modelled by a combination of UPPAAL timed automata and PHAVER hybrid automata.

[8] and later work resulted in an almost complete model of EKC204A aspects. In this paper we describe only the relevant part—temperature estimation—which is described by hybrid automata. [4] The UPPAAL model consists of the following processes modelling various aspects:

– The test is part of environment which controls how the temperature should be (e.g. decreased until it is below $-7°C$ and then increased again).
– The tempGen generates a sequence of concrete temperatures that are sent as raw temperature values injected into sensors.
– tempMonitor read the temperature sensor values and provides the calculated temperature values to the rest of the TA instances.
– The lowTempAlarm monitors the calculated temperature and triggers an alarm if the temperature is below the threshold for longer than allowed time bound.

The hybrid model consists of just two hybrid automata: TempMonitor monitors the calculated temperature values and TempRoom monitors whether the sensed temperature is changing reasonably (e.g. not too rapidly). The latter is optional, but it demonstrates that we can monitor the behavior of the tester as well as IUT.

Fig. 7 provides an overview on how the two models are synchronised together with IUT:

– $temp\_t(ENVTemp)$ is a signal generated from tempGen to tempMonitor which results in concrete action $inject\_temp(T)$ sent to the IUT and a $set\_temp(room\_temp)$ synchronization in hybrid model between TempRoom and TempMonitor.

[4] The implementation of the adapter for PHAVER, a generated script instance and requirement models can be downloaded at http://www.cs.aau.dk/~marius/phaver.

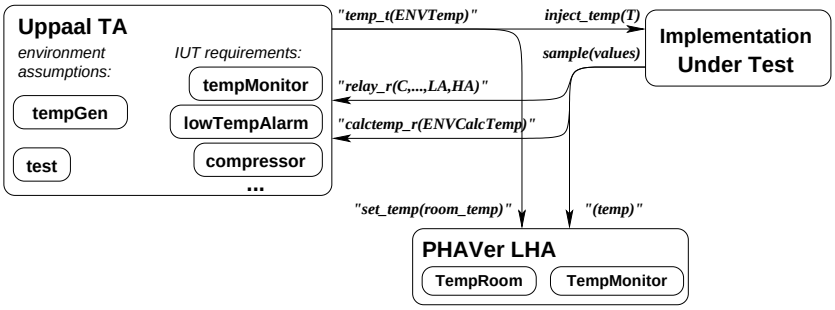


Fig. 7: UPPAAL and PHAVER model setup for temperature controller.

– $sample(values)$ is an output from IUT which is translated into relay changes ($relay\_r(C, F, D, A, HA, LA)$) to the UPPAAL model and a calculated temperature snapshots $calctemp\_r(ENVCalcTemp)$ to the UPPAAL model and ($temp$) update in the PHAVER model.

The Abstract Timed Model. In UPPAAL model the tempMonitor is responsible for estimating and displaying the temperature value on the screen as well as notifying all other processes. The calculated temperature is estimated by $CalcTL$ and $CalcTU$ variables denoting lower and upper bounds.

Fig. 8 shows the temperature estimation abstraction when a new temperature value is injected into the sensors and the resulting UPPAAL timed automaton which computes the abstract temperature bounds.

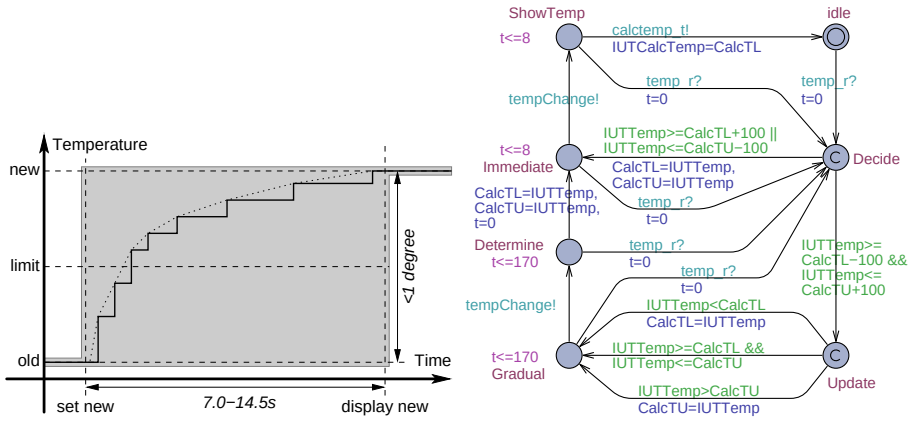


Fig. 8: Temperature estimation and its abstract model as tempMonitor process.

LowTempAlarm is modelled as a parallel process (Fig. 9) interpreting the calculated temperature bounds $[CalcTL; CalcTU]$ using non-deterministic edges with guard comparisons to $LowTempLimit$.

Hybrid Model Fig. 10 shows a hybrid automaton for temperature calculation which has four locations: *idle* – temperature is constant, *decide* – there is temperature change and controller instantaneously decides whether it is going up

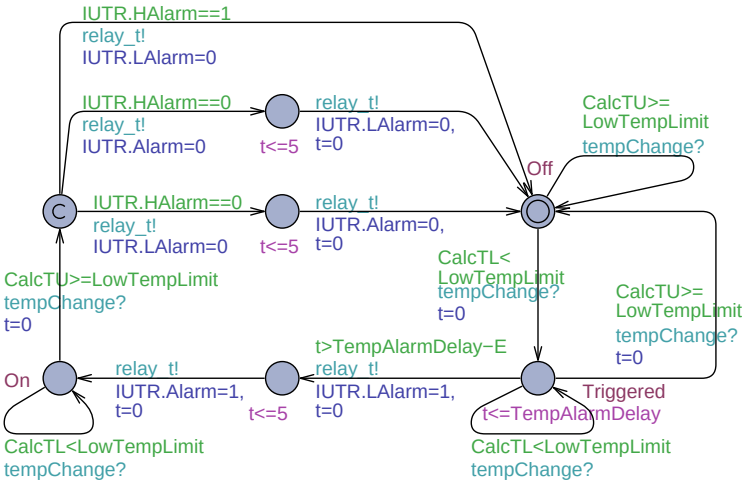


Fig. 9: Requirement model for low temperature alarm.

or down, *adjustUp* – the temperature is rising, *adjustDown* – the temperature is dropping. The current temperature estimate is kept in variable *temp* and the new sensor value is changed with *set_temp* event in variable *target*.

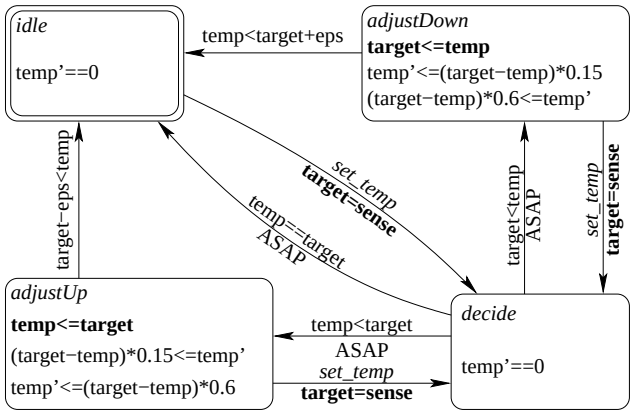


Fig. 10: Model of a controller temperature sensing and estimation.

Results. Fig. 11 shows the temperature region computed from the model by an over-approximation for each instance of time during testing:

- Calculated temperature estimation starts from a single (known) point at $16.8°C$.
- Uppaal TRON then generates a new air temperature at $16.6°C$ and feeds it to IUT.
- PHAVer estimates that the calculated temp should fall withing first region.

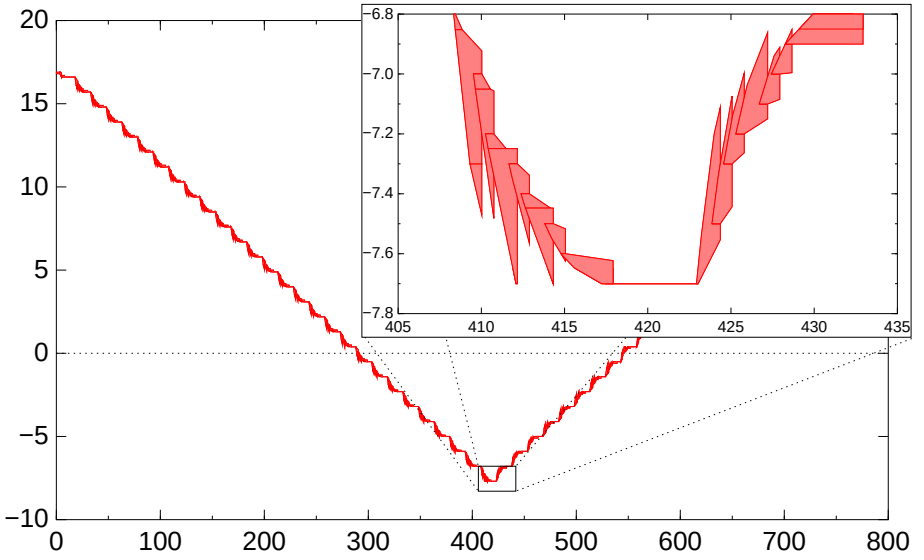


Fig. 11: Symbolic state evolution in PHAVer from test trace monitoring: time in seconds on horizontal axis, temperature in $°C$ on vertical axis.

- HybridAdapter samples a new calculated temperature reading and reports it to PHAVerAdapter, but does not report it yet to Uppaal TRON.
- PHAVer finds the observed temperature point (an interval in time) in the estimated region, updates it and recomputes future temperature region (second region).
- The process continues in this way until the calculated temperature reaches the injected temperature value (within 15s) where both Uppaal TRON and PHAVer are notified.
- The calculated temperature estimate collapses to one point in the PHAVer plot, until a new temperature injection is dictated by Uppaal TRON.

6 Conclusions

We have shown how timed automata and hybrid automata model-checkers can be combined to achieve online testing and monitoring of embedded software controller at different degrees of precision: timed/discrete and sampled continuous signals.

In the current setting, PHAVer can only function as a monitor, because it lags behind Uppaal TRON (due to the high computation cost of computing state set updates). However, ongoing work on improvements on the PHAVer engine, and we are optimistic that it will be capable of also functioning as trajectory stimulator. Alternatively a simulator tool can be used to generate dynamical stimuli.

In the future we will implement the operators used in monitoring using reachability algorithms that are based on time-discretization and bounded time horizon, see [4].

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.
2. Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in Lecture Notes in Computer Science, pages 232–243. Springer–Verlag, October 1995.
3. Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In Manfred Morari and Lothar Thiele, editors, *HSCC*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2005.
4. Goran Frehse and Rajarshi Ray. Design principles for an extendable verification tool for hybrid systems. In *ADHS'09 : 3rd IFAC Conference on Analysis and Design of Hybrid Systems*, 2009.
5. Thomas A. Henzinger and Pei-Hsin Ho. Algorithmic analysis of nonlinear hybrid systems. In *Proceedings of the 7th International Conference on Computer Aided Verification*, pages 225–238. Springer-Verlag, 1995.
6. Anders Hessel, Kim Larsen, Marius Mikucionis, Brian Nielsen, Paul Pettersson, and Arne Skou. Testing real-time systems using uppaal. In Robert Hierons, Jonathan Bowen, and Mark Harman, editors, *Formal Methods and Testing*, volume 4949 of *Lecture Notes in Computer Science*, pages 77–117. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-78917-8_3.
7. Kim Larsen, Marius Mikucionis, and Brian Nielsen. Online testing of real-time systems using uppaal. In Jens Grabowski and Brian Nielsen, editors, *Formal Approaches to Software Testing*, volume 3395 of *Lecture Notes in Computer Science*, pages 79–94. Springer Berlin / Heidelberg, 2005. 10.1007/978-3-540-31848-4_6.
8. Kim G. Larsen, Marius Mikucionis, Brian Nielsen, and Arne Skou. Testing real-time embedded software using uppaal-tron: an industrial case study. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 299–306, New York, NY, USA, 2005. ACM.
9. Steve Sims and Daniel C. DuVarney. Experience Report: The Reactis Validation Tool. In *Proceedings of the 2007 ACM SIGPLAN international conference on Functional programming - ICFP '07*, volume 42, page 137, New York, New York, USA, October 2007. ACM Press.
10. Jan Tretmans and Axel Belinfante. Automatic testing with formal methods. In *EuroSTAR'99: 7^{th} European Int. Conference on Software Testing, Analysis & Review*, Barcelona, Spain, November 8–12, 1999. EuroStar Conferences, Galway, Ireland.
11. Michiel van Osch. Hybrid input-output conformance and test generation. In Klaus Havelund, Manuel Núñez, Grigore Rosu, and Burkhart Wolff, editors, *Formal Approaches to Software Testing and Runtime Verification*, volume 4262 of *Lecture Notes in Computer Science*, pages 70–84. Springer Berlin / Heidelberg, 2006. 10.1007/11940197_5.
12. Michiel van Osch. *Automated Model-based Testing of Hybrid Systems*. PhD thesis, Technische Universiteit Eindhoven, 2009.