Evaluating On-line Model Checking in UPPAAL-SMC using a Laser Tracheotomy Case Study

Xintao Ma[1], Jonas Rinast[1], Sibylle Schupp[1], and Dieter Gollmann[2]

1 **Institute of Software Systems, Hamburg University of Technology**
21073 Hamburg, Germany
{xintao.ma,jonas.rinast,schupp}@tuhh.de

2 **Security in Distributed Systems, Hamburg University of Technology**
21073 Hamburg, Germany
diego@tuhh.de

Abstract

On-line model checking is a variant of model checking that evaluates properties of a system concurrently while deployed, which allows overcoming limitations of inaccurate system models. In this paper we conduct a laser tracheotomy case study to evaluate the feasibility of using the statistical model checker UPPAAL-SMC for on-line model checking in a medical application. Development of automatic on-line model checking relies on the precision of the prediction and real-time capabilities as real-time requirements must be met. We evaluate the case study with regards to these qualities and our results show that using UPPAAL-SMC in an on-line model checking context is practical: relative prediction errors were only 2% on average and guarantees could be established within reasonable time during our experiments.

1998 ACM Subject Classification D.2.4 Software Engineering – Software/Program Verification – Model Checking, F.1.1 Computation by Abstract Devices – Models of Computations – Automata, F.1.2 Computation by Abstract Devices – Modes of Computations – Online Computation, I.6.3 Simulation and Modeling – Applications, I.6.7 Simulation and Modeling – Simulation Support Systems, J.3 Life and Medical Sciences – Health

Keywords and phrases On-line Model Checking, Laser Tracheotomy, UPPAAL-SMC, Patient-in-the-loop

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.100

1 Introduction

In the medical domain not only the devices must operate reliably but also the safety of connected patients must be guaranteed. This requirement becomes more and more pressing with the increased development of patient-in-the-loop systems that monitor and treat patients autonomously and where a malfunction could seriously harm the patient. Model checking, a widely known technique to show that a system fulfills certain properties, might be an option to ensure safe operation of such systems, but is often not adequate in the medical context. Classic model checking relies on models that accurately predict the system state also in a distant future for all system components for reasoning about the system. When human physiology is involved such models are unavailable most of the time. For example, predicting the long-term behavior of the blood oxygen concentration of a human patient currently is infeasible since the present understanding of the processes within the human body only permit short-time predictions.

Medical Cyber Physical Systems – Medical Device Interoperability, Safety, and Security Assurance (MCPS'14).
Editors: Volker Turau, Marta Kwiatkowska, Rahul Mangharam, and Christoph Weyer; pp. 100–112
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On-line model checking relaxes the need for accurate long-term models required by classic model checking. Instead of statically proving a property of the system on-line model checking yields guarantees that are only valid for a limited time using bounded model checking. To still provide a safety guarantee at all times the on-line model checking approach repeatedly evaluates the property to extend its period of validity indefinitely. This iterative process allows on-line model checking to dynamically adapt the underlying system model. Thus, the on-line model checking approach can employ measures to adjust the model parameters such that they match runtime observations if the model accuracy decreases significantly, and thus reestablish a consistent state. Accurate long-term models are no longer required as the dynamic adaptation of the short-term models to the current real-world situation may still yield the desired long-term guarantees.

In this paper we carry out an on-line model checking case study using the model checker UPPAAL and evaluate its statistical model checking module UPPAAL-SMC regarding its suitability for on-line model checking. UPPAAL is one of few available tools that have the potential to carry out automatic on-line model checking. Evaluating whether its performance in practice meets the real-time requirements imposed by on-line model checking is crucial to tapping its full potential. Therefore we encode the models of a previous on-line model checking case study that models a laser tracheotomy surgery with hybrid models for the model checker PHAVer using the timed automata formalization used by UPPAAL. This relation enables us to compare our results to the previous work and lets us focus on questions regarding UPPAAL-SMC's performance and suitability. We then carry out the on-line model checking process with our derived models using a prototype for automatic on-line model checking with UPPAAL. Next, the collected data on the accuracy of parameter prediction and the run-time performance is compared to the results of the original case study. Relative errors of SpO_2 estimations were on average about 2% which is slightly worse than the original case study. For performance, a verification step took on average about 50ms which is a significant improvement. As a general result it follows that it is practical to use UPPAAL in an on-line model checking context.

The rest of the paper is organized as follows: Section 2 introduces related work. Section 3 provides a short introduction to hybrid automata and their relation to UPPAAL-SMC. The on-line model checking approach and its implementation with UPPAAL is the topic of Section 4. Section 5 shows the analyzed case study and its on-line models. Section 6 provides our experiment results and an evaluation of those. At last, Section 7 summarizes the paper and gives ideas on future research.

2 Related Work

In general on-line model checking can be put into the context of self-adaptive software. More specifically, verification at runtime enables ways to produce self-adapting software systems [6]. Zhao et al. introduce the on-line model checking approach as a lightweight verification technique to reduce the state space explosion problem [18]. They argue that on-line model checking is significantly different from the runtime verification approach: in contrast to on-line model checking runtime verification operates directly on the execution trace without involving a system model. Thus, the approach is not capable of predicting property violations. Steering is a control-theoretic approach trying to resolve this drawback [10]. Li et al. apply on-line model checking to a laser tracheotomy surgery scenario to ensure the patient safety [14, 15]. They use the hybrid model checker PHAVer in combination with a custom implementation to carry out the model checking procedure. This is the reference case study we compare our UPPAAL implementation to. Bartocci et al. [2] and Chen et al. [8] deal with the model repair problem, a related approach that tries to adjust model parameters to satisfy system properties in case they are violated. However, here the adjustment goal for on-line model checking is not to satisfy a property but to ensure that the model does not deviate from the observed real-world state.

Regarding the implementation of on-line model checking, Bu et al. pursue the development of an on-line model checking tool set called BACH$_{\mathrm{OL}}$, which is based on the linear hybrid automaton model checker BACH, to facilitate verification of complex cyber-physical systems [4]. Furthermore, in earlier work we began implementing a framework for on-line model checking with UPPAAL [17]. This framework automatically performs the necessary state reconstruction for seamless model simulation and verification.

In the context of closed-loop medical systems, King et al. report on their experience with their Medical Device Coordination Framework (MDCF) when modeling a closed-loop medical system to control an infusion pump [13]. Their research focuses on the interoperability between medical devices. Arney et al. also analyze a patient-in-the-loop system [1]: they develop UPPAAL and MATLAB models to show in advance potential flaws in the control loop that endanger the patient. The development and verification of formal models for pacemaker systems is the topic of work by Chen at al. [7] and Jiang et al. [12].

3 Timed and Hybrid Automata

Both the models from the original case study our work is based on and our models use variations of finite state machines to represent the system. The original case study derives a *hybrid automata* model. In UPPAAL, however, modeling of hybrid automata is only possible with the statistical model checking extension UPPAAL-SMC as UPPAAL normally uses networks of *timed automata* as the underlying modeling formalism, a subset of hybrid automata.

A hybrid automaton, according to Henzinger, is a finite state automaton extended with a set of continuous variables [11]. Thus, a hybrid automaton may model discrete and continuous behavior. Such a hybrid automaton consists of the following parts:

- **Graph.** A finite directed multigraph (V, E) that models the topology of the discrete transitions with the locations V and the edges E.
- **Variables.** A finite set of variables $X = \{ x_1, \dots, x_n \}$ valued in the reals ($\mathbb{R}$) together with its set of derivatives $\dot{X} = \{ \dot{x_1}, \dots, \dot{x_n} \}$.
- **Condition Predicates.** Three labeling functions that assign predicates to locations $l \in V$: *init* assigns initial valuations, *inv* assigns an invariant condition, and *flow* assigns a flow condition that determines how variables evolve over time in a location. These are generally linear differential equations.
- **Guards.** A labeling function *guard* that assigns predicates to edges $e \in E$ that specify when a transition over an edge may be triggered.
- **Actions.** A labeling function *action* that assigns actions $a \in \Sigma$ to edges $e \in E$ that are performed when a transition over the edge is triggered.

In UPPAAL-SMC a hybrid automaton is defined in terms of the underlying timed automata definition such that most of the known UPPAAL features, e.g., synchronization, could be carried over. In UPPAAL, a timed automaton is defined as follows

▶ **Definition 1.** A *timed automaton* T is a tuple $T = \{ L, l_0, C, A, E, I \}$ where L is a set of locations, $l_0 \in L$ is the initial location, C is a set of real-valued clock variables, A is the action set, E is the set of edges of the form (l, a, g, R, l') where $l, l' \in L$, $a \in A$, g a predicate over C, and R a subset of C, and I is a function that assigns invariant predicates to locations $l \in L$.

In Definition 1 the set R on an edge is its reset set, i.e., the set of clocks that are set to certain values when a transition involving the edge is fired. Note that in UPPAAL a reset of a clock does not necessarily mean the clock is set to zero; any integer value is allowed. Furthermore, UPPAAL restricts predicates over C to conjunctions of terms that bound a clock or a difference of clocks by an integer.

Definition 1 contains nearly all the necessary components for modeling a hybrid automaton. When taking into account the renaming of components only the *flow* function defining the behavior of individual variables in a location can not be specified directly. Therefore, UPPAAL-SMC extends the timed automaton definition with an additional component F, which allows modeling of hybrid automata [9]. The function F corresponds to the *flow* labeling in Henzinger's hybrid automata and is called a *delay function.* It allows modification of the default delay function of UPPAAL-SMC, which advances all clocks synchronously at the same rate, in certain locations l by defining explicit rates for clocks: $x' = e$ where e only depends on the discrete part of the state. It follows that the transformation of hybrid models to UPPAAL-SMC models may be carried out if their *flow* function can be expressed by explicit rates. In this case study we determine the delay function by performing a linear regression (see Section 5 and Section 6).

For more information on UPPAAL, Behrman et al. provide a complete introduction [3]. The statistical model checking module, UPPAAL-SMC, is covered in the publication by Bulychev et al. [5].

4 On-line Model Checking

This section introduces the on-line model checking process in Subsection 4.1 and then provides details on the implementation aspects of it in Subsection 4.2.

4.1 The On-line Model Checking Approach

On-line model checking is a technique to apply classic model checking to domains where accurate modeling of a system may be infeasible. In such cases classical model checking, if based on approximate models, may yield seemingly satisfactory results. But in reality those properties can not be guaranteed because the model does not correctly reflect the system. On-line model checking overcomes the model inaccuracies by periodically adjusting the underlying model to the real-world values observed from the system.

Figure 1 and Figure 2 depict the relation of the state spaces of both approaches. Figure 1 shows the classical model checking approach where a single model of the system is constructed. Here, the model does not correctly model all aspects of the systems. Thus, the state space of the model is only a subset of the state space of the system and an actual trace of the system as shown by the arrow starting with the circle may leave the model state space. As only the state space of the model is checked for compliance with the requirements for the system there are cases where the model checking approach assures a system property but in reality that property may not be satisfied. Thus, when an exact model of a system is not available classical model checking does not yield reliable results.

In contrast, Figure 2 shows the situation when applying on-line model checking. Here, the model is adjusted periodically and a new model is generated based on the current system

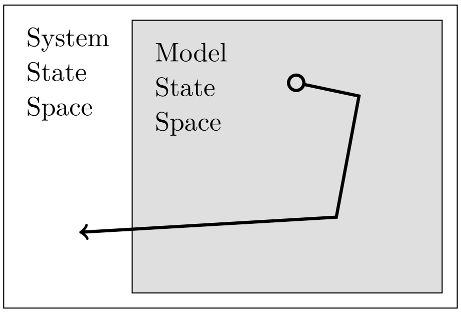

Figure 1 State Space in Classic Model Checking.

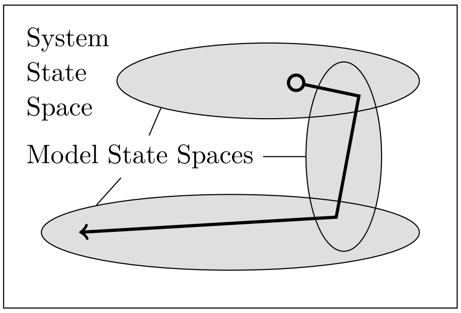

Figure 2 State Space in On-line Model Checking.

trace. The first benefit is that the concrete system state is always valid in the current model. Thus, leaving the model state space is impossible and a guarantee obtained from the model checker is always reliable although the models do not at all times accurately reflect the system. Consequently, the obtained guarantees only have limited periods of validity because of the limited model scope. The limited scope though is responsible for another benefit: the model state spaces in general are smaller and thus the model checking performance becomes better. It follows that with the on-line model checking approach the model checking technique can be applied to domains where models are likely to be inaccurate because model adjustments may overcome any inaccuracies.

▶ **Example 2.** As an example assume a light bulb is supposed to be switched on and off every five seconds. Experience shows that this kind of light bulb malfunctions after 1000 on-off-cycles. It is critical that the light cycle in the system never stops and thus we want to ensure a 30 second grace period to exchange the bulb when approaching the end of its lifetime. If the model is correct a simple calculation yields the time when a change is necessary. Now, assume that the real system does not switch the light on and off every five seconds but the switching delay varies unpredictably. Then verifying every 30 seconds that no malfunction occurs within the next minute would achieve the same 30-second grace period only if the model variables are updated with the current light state and the number of on-off-cycles that actually occurred.

4.2 Implementing On-line Model Checking

The implementation of on-line model checking of a real system can be divided into three phases, one before deployment of the system, and two during deployment:

- **Modeling.** During the modeling phase first the requirements for a system are specified. Then a model of the system is developed that allows reasoning about those requirements. Also, the initial state of the model is defined, i.e., the values with which the real system starts operation.
- **Verification.** In the verification phase a current system model is passed to the model checking engine together with the requirement properties and checked for compliance. In case the verification fails an emergency handling routine may be triggered to resolve the issue. Otherwise a guarantee is obtained that the requirements are fulfilled for a limited time bound T.
- **Adjustment.** In the adjustment phase the real system is observed and the previous model is adjusted accordingly to accurately represent the current and near-future states.

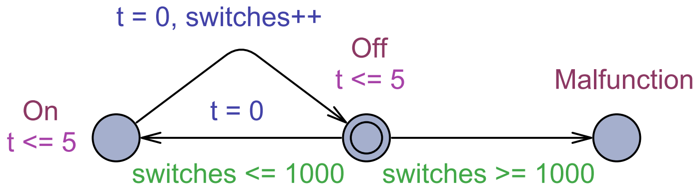

Figure 3 Model Checking Example System.

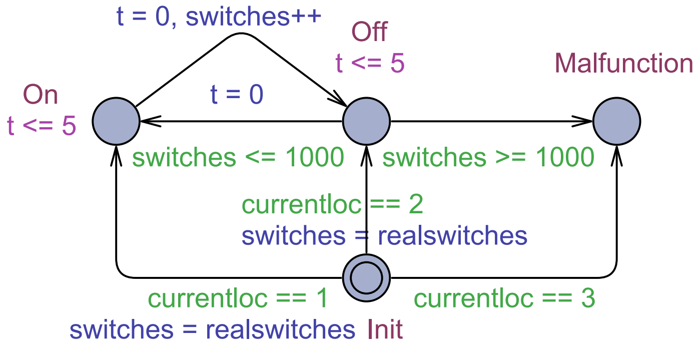

Figure 4 On-line Model Checking Example System.

The adjustment must be performed before the period of validity T runs out to ensure continuity of the guarantees. The newly created model is then forwarded to the next verification phase.

When implementing on-line model checking it is thus necessary that the model provides means for adjusting the model. In this paper we manually modify the basic system models to allow the necessary modifications because automatically providing these means induces a reconstruction problem of the previous system state [17] and solving the reconstruction problem introduces additional machinery, which would dilute the focus on UPPAAL's performance of this case study.

▶ **Example 3.** Recall Example 2. Figure 3 shows the basic UPPAAL model for the example system. The adaptation of the model is rendered possible by introducing two parameters, the performed number of switches and the current state of the lamp. They are passed during the adaptation phase from the real system to the model as the constant values `realswitches` and `currentloc`. The resulting on-line model is depicted in Figure 4.

5 A Medical Case Study

In this section we present an on-line model checking case study on a medical laser tracheotomy scenario and demonstrate the applicability of UPPAAL in such scenarios. In Subsection 5.1 we introduce laser tracheotomy in general and related safety requirements. The concrete UPPAAL system models derived from the case study by Li et al. [14] are the focus of Subsection 5.2.

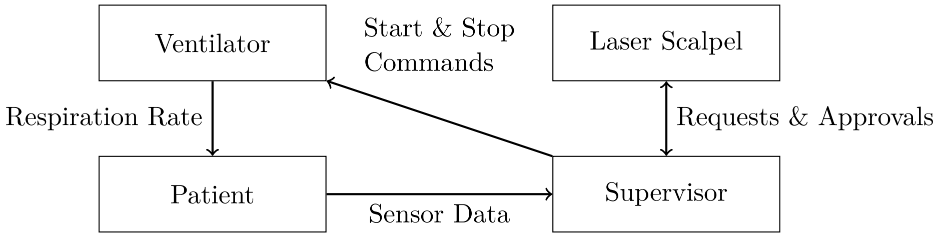

Figure 5 Laser Tracheotomy System [14].

5.1 Laser Tracheotomy

Tracheotomy is a surgery performed on patients that have problems breathing through their nose or mouth, e.g., when the tongue muscle falls back and blocks the air flow while sleeping. During the surgery a direct access to the windpipe of the patient is created, usually from the front side of the neck. Laser tracheotomy refers to the kind of tracheotomy where the access to the windpipe is created using a laser scalpel, a medical device capable of cutting tissue with focused light. Using laser for the cut has several benefits such as a greater precision and a reduction of blood loss due to blood vessels being closed immediately. However, during tracheotomy the laser also poses the threat of tissue burns in case the oxygen concentration in the windpipe of the patient is too high.

In this case study we want to ensure that the laser may only be triggered when an operation is safe. Additionally, as the patient is ventilated during the surgery, we want to ensure that the blood oxygen of the patient does not drop to dangerous levels, because ventilation needs to be suspended during cutting. Lastly, for convenience of the surgeon, an additional requirement is that once the use of the laser is approved the laser should emit for a minimum time such that the cut is not interrupted unnecessarily. The verification properties for the statistical model checking are given in $\text{WMTL}_{\leq}$ (Weighted Metric Temporal Logic) [5]:

- O_2 above threshold while laser emits

  -
    ```
    Pr[<=100](<> O2 > Th_O2 && LaserScalpel.LaserEmitting)
    ```
- SpO_2 below threshold while laser emits

  -
    ```
    Pr[<=100](<> SpO2 < Th_SpO2 && LaserScalpel.LaserEmitting)
    ```
- Laser stops emitting early

  -
    ```
    Pr[<=100](<> (O2 > Th_O2 || SpO2 < Th_SpO2) &&
              t_appr < Th_appr && LaserAppr == true)
    ```

These properties characterize unreachable states and thus the probabilities should be zero with the configured confidence.

5.2 System Modeling

The laser tracheotomy scenario described by Li et al. consists of four different components [14]:

- **Patient.** The patient under surgery characterized by current windpipe oxygen level (O_2) and blood oxygen level (SpO_2).
- **Ventilator.** The medical ventilator device regulating the patient's breathing rate during the surgery. The ventilator is characterized by the current height of the pressure cylinder.
- **Laser Scalpel.** The laser scalpel is used to cut the opening to the windpipe. The laser scalpel is characterized by whether or not the surgeon currently wants to operate the laser and if operation is allowed.

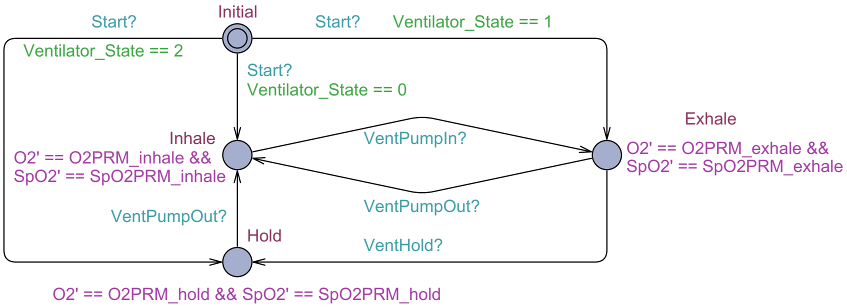

Figure 6 Patient UPPAAL Model.

- **Supervisor.** The supervisor is responsible for ensuring the safety requirements of the system as given in Subsection 5.1. The supervisor approves usage of the laser scalpel and operates the ventilator.

Figure 5 shows the connections of the system components with respective communication data. The ventilator regulates the respiration rate of the patient. The physiological signals of the patient are measured by sensors and forwarded to the supervisor. The supervisor analyzes the values and either approves usage of the laser scalpel requested previously and consequently stops the ventilator, or usage is prohibited and the ventilator continues normal operation. Additionally, when an approved cut is finished the ventilator starts operating again by instruction of the supervisor. We now discuss our UPPAAL models of the components for on-line model checking in more detail. All of the models were derived from the original hybrid models using the encoding from Section 4. The main difficulty in the transformation of the models is representing the continuous variables O_2 and SpO_2 in the hybrid models using clock variables in UPPAAL-SMC and ensuring correct system behavior using synchronization. The remaining parts are straight-forward because the graph components and transition constraints carry over directly due to the same finite state machine formalism.

5.2.1 Patient

The patient model consists of three locations plus the initialization location. The locations correspond to the patient inhaling and exhaling assisted by the ventilator and the patient exhaling without assistance when the ventilator is switched off. In the three locations the O_2 and SpO_2 values are predicted using a linear regression approach taking into account a history of 30 seconds (see Section 6).

5.2.2 Ventilator

The ventilator model also has three main locations, an initialization location and two intermediate locations for communication reasons. The main locations correspond to the ventilator pumping air into the patient, out of the patient, and not pumping at all. Here the current height of the ventilation cylinder, H_vent, is modified accordingly. Communication with the patient model is implemented such that the patient always inhales and exhales as enforced by the ventilator. Furthermore input from the supervisor model is accepted to enable and disable the ventilator.

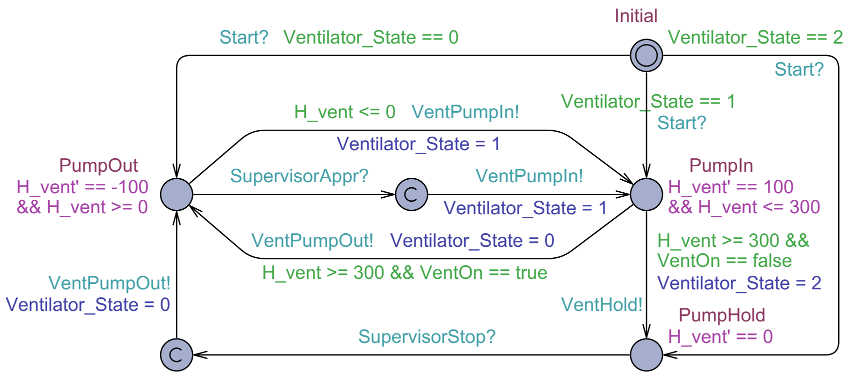

Figure 7 Ventilator UPPAAL Model.

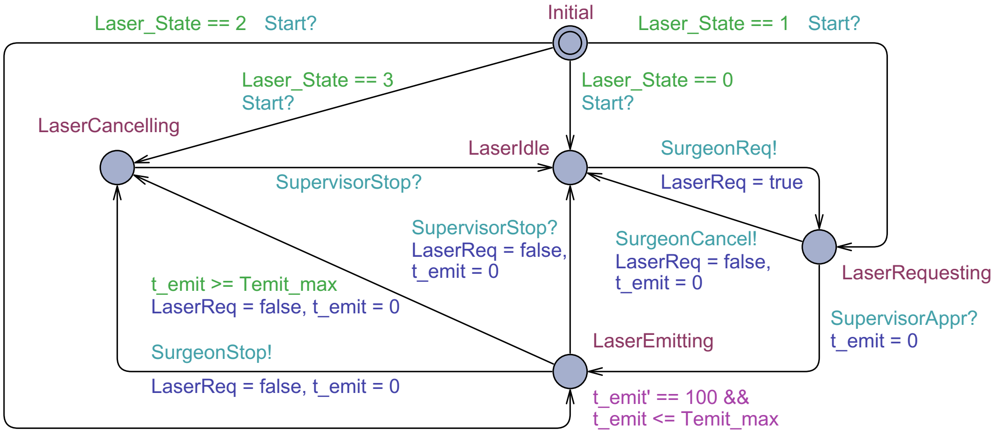

Figure 8 Laser Scalpel UPPAAL Model.

5.2.3 Laser Scalpel

The laser scalpel model represents the interaction between the surgeon and the laser scalpel. It uses four locations. The surgeon may send a request to the supervisor model to trigger the laser, which eventually gets approved. When the laser emits the surgeon can either switch the laser off or revoke the approval if conditions necessitate action. Communication thus takes part between the laser scalpel and the supervisor model. The surgeon inputs are left open meaning that any external input may be executed at any time during verification.

5.2.4 Supervisor

The supervisor model checks if the physiological parameters of the patient are within safe boundaries and approves the laser usage for a maximum duration. If any of the safety requirements gets violated the supervisor revokes its approval. The interesting part here is that the initialization part also checks if a safety requirement was violated. This behavior is necessary because when the model is adapted the O_2 and SpO_2 values may change, which might invalidate a previous approval. Also, when the supervisor approves usage of the laser the ventilator is put on hold.

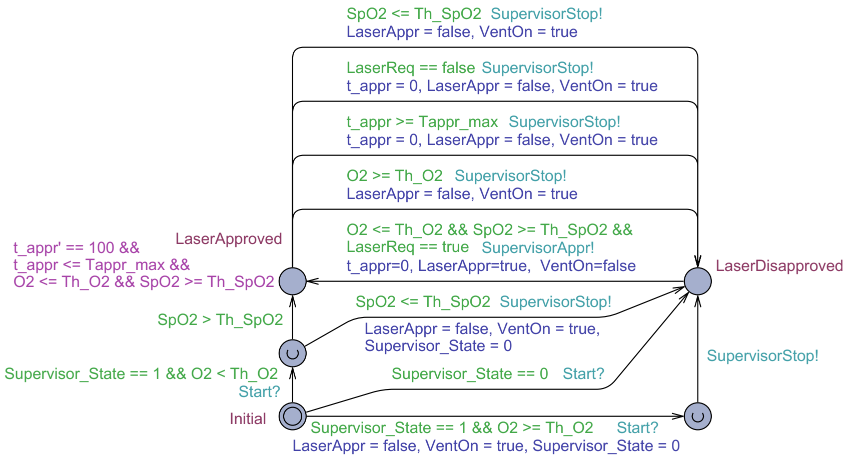
SpO2 <= Th_SpO2 SupervisorStop!
LaserAppr = false, VentOn = true
LaserReq == false SupervisorStop!
t_appr = 0, LaserAppr = false, VentOn = true
t_appr >= Tappr_max SupervisorStop!
t_appr = 0, LaserAppr = false, VentOn = true
O2 >= Th_O2 SupervisorStop!
LaserAppr = false, VentOn = true
O2 <= Th_O2 && SpO2 >= Th_SpO2 &&
LaserReq == true SupervisorAppr!
t_appr=0, LaserAppr=true, VentOn=false
LaserApproved
t_appr' == 100 &&
t_appr <= Tappr_max &&
O2 <= Th_O2 && SpO2 >= Th_SpO2
LaserDisapproved
SpO2 <= Th_SpO2 SupervisorStop!
LaserAppr = false, VentOn = true,
Supervisor_State = 0
SpO2 > Th_SpO2
SupervisorStop!
Supervisor_State == 1 && O2 < Th_O2
Start?
Supervisor_State == 0 Start?
Initial
Supervisor_State == 1 && O2 >= Th_O2 Start?
LaserAppr = false, VentOn = true, Supervisor_State = 0

Figure 9 Supervisor UPPAAL Model.

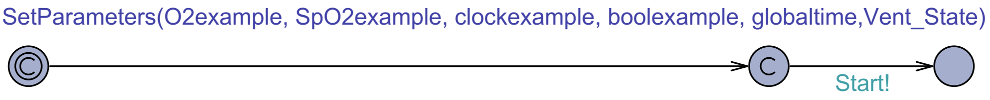
SetParameters(O2example, SpO2example, clockexample, boolexample, globaltime,Vent_State)
C
Start!

Figure 10 Initialization UPPAAL Model.

5.2.5 Initialization

Lastly, the initialization model has the purpose of initializing all constants that may have been adapted to real-world values during model adaptation. Using broadcast synchronization, a starting transition guarantees a common starting point for the whole model.

6 Experiments and Evaluation

To evaluate the on-line model checking approach with UPPAAL we carried out several experiments with our models. Real-world patient data necessary for the adaptation steps was extracted from the PhysioNet database, an open medical database offering a large collection of recordings of medical signals of various kind (http://www.physionet.org). Six different patient traces were assembled and used as a basis. Every patient trace was executed ten times yielding 60 experiments in total. Table 1 shows the PhysioNet data bases and the patient IDs of the data used. More information on the data can be found in the original thesis on this topic [16]. All experiments ran the system for 600 seconds where every three seconds a model adaptation and verification was performed. Thus, the workflow of every three-second cycle is as follows: first we adjust the O_2 and SpO_2 values in the model to the observed values. Then we try to verify the system properties for the next six seconds. And lastly, we evaluate the verification results such that if a property was not verified we derive that in three seconds an unsafe state occurs and thus emergency measures should be taken

Table 1 PhysioNet Databases and Patient IDs.

	Database	#1	#2	#3	#4	#5	#6
O_2 (CO_2)	MGF/MF	mgh077	mgh077	mgh089	mgh057	mgh019	mgh110
SpO_2	MIMIC v2	a45463	a45436n	439n	n10301n	a45611n	477n

Table 2 Relative Errors of O_2 and SpO_2 Estimation.

[%]	#1	#2	#3	#4	#5	#6
Min SpO_2	0	1.43	0.52	2.28	0.48	0.59
Max SpO_2	6.01	1.62	0.63	2.99	0.59	4.18
Avg SpO_2	1.59	1.54	0.60	2.82	0.54	2.27
Min O_2	0.6	18.0	12.3	16.8	11.3	8.3
Max O_2	66.0	23.2	14.0	20.5	15.3	10.5
Avg O_2	21.7	20.8	13.4	18.9	12.3	9.2

Table 3 Model Checking Execution Times.

[s]	Minimum	Maximum	Average
UPPAAL-SMC	0.033	0.32	0.047
PHAVer	0.571	1.445	0.727

before the unsafe state is reached. Note that if the models correctly predict the short-term behavior of O_2 and SpO_2 and the supervisor strategy is effective such an emergency can not arise. The history window for the linear regression was 30 seconds in all cases. The confidence level for the statistical model checker was set to 99%. We evaluated three aspects of the approach: first we checked whether the safety requirements given in Subsection 5.1 are violated for any patient trace. Then we compared the relative errors of our O_2 and SpO_2 predictions to the values in the reference paper [14]. Lastly, we evaluated the execution times with a focus on the real-time requirements.

The first result is straightforward: during all experiments all three safety properties were satisfied at all times with the confidence level of 99%. Thus, our models seem to be accurate enough to predict the physiological parameters of the patient for a time bound of three seconds. Moreover, the supervisor strategy implemented in the models proves to be effective at preventing accidental tissue burns resulting from triggering the laser at inappropriate times.

Table 2 shows the relative errors of our parameter estimation. The SpO_2 estimates are very consistent and in general show a relative error of about 2%. These results are accurate enough to guarantee the safety of the patient with regards to the blood oxygen. In contrast, the estimation of windpipe oxygen is not that precise with an average relative error of about 16%. However, due to the supervisor strategy the safety of the patient is still guaranteed. Still, the laser could potentially be allowed to fire more often. Thus, a more sophisticated prediction strategy than linear regression is likely to yield better prediction results, which enable the supervisor to approve the use of the laser more often. Compared to the results of the original case study our SpO_2 results are slightly less accurate but still useful for safety statements. As the original case study does not specify exactly which patient traces were used as an experiment basis differences in the results may simply stem from the selection of different traces. For the O_2 results Li et al. provide no relative error results.

Table 3 shows the execution times of an adaptation step of the models and the following verification of the safety properties. Our experiments were carried out on a Macbook Pro 2.66 GHz with 4GB memory using iOS 10.6.8. In the experiments our approach took at worst 320 milliseconds for a cycle while in the original case study nearly 1.5 seconds elapsed. Unfortunately, the original case study does not specify the used hardware. Still, the approach using simulation of timed automata in UPPAAL-SMC for verification performs significantly better than the symbolic verification of hybrid automata in PHAVer. Thus, we assume the speedup can not be attribute only to differences in hardware, especially because our hardware is not on the top end. Looking at the absolute values with the hard real-time constraints of three seconds for one cycle in mind, using UPPAAL-SMC provides a performance advantage in practice. With execution times of about 10% of the real-time deadlines the implementation in a hard real-time system seems feasible.

7 Conclusion and Future Work

This paper presented the on-line model checking approach, a variant of model checking that allows reasoning about systems where accurate long-term models are unavailable. We implemented a medical laser tracheotomy case study using UPPAAL-SMC and used it to evaluate the on-line model checking approach in practice. The on-line model checking approach periodically adjusts the underlying system model to real-world values and analyzes the new models, e.g., for patient safety issues. The case study showed that this approach is capable of providing reliable safety guarantees even if the patient's physiological behavior is modeled only roughly using a simple linear regression approach when parameters are continuously adapted to the real-world values. Although this paper identifies on-line model checking as a useful technique to ensure safety of complex systems, further research is necessary to support this claim. Future research should focus on larger scale case studies and provide a unified approach including automatic adaptation interfaces to ease the development of systems that should be monitored using on-line model checking. Such an automatic adaptation interface would synthesize necessary means to adapt a model from a classical model checking model and execute the on-line model checking procedure to allow seamless simulation and verification of the system in question.

Acknowledgements. We thank the anonymous reviewers of MedicalCPS 2014 for references to related work and comments that helped to improve the presentation.

References

1 David Arney, Miroslav Pajic, Julian M. Goldman, Insup Lee, Rahul Mangharam, and Oleg Sokolsky. Toward patient safety in closed-loop medical device systems. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems – ICCPS '10*, pages 139–148, Stockholm, Sweden, 2010. ACM New York, NY, USA.

2 Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C.R. Ramakrishnan, and Scott A. Smolka. Model Repair for Probabilistic Systems. In Parash Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *Lecture Notes in Computer Science*, pages 326–340. Springer Berlin Heidelberg, 2011.

3 Gerd Behrmann, Alexandre David, and Kim G. Larsen. A Tutorial on Uppaal 4.0. Technical report, Department of Computer Science, Aalborg University, Aalborg, Denmark, 2006.

4 Lei Bu, Dingbao Xie, Xin Chen, Linzhang Wang, and Xuandong Li. Demo Abstract: BACHOL – Modeling and Verification of Cyber-Physical Systems Online. In *Proceedings of the 3rd ACM/IEEE International Conference on Cyber-Physical Systems – ICCPS '12*, pages 222–222, Beijing, China, April 2012. IEEE.

5 Peter Bulychev, Alexandre David, Kim G. Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. In Herbert Wiklicky and Mieke Massink, editors, *10th Workshop on Quantitative Aspects of Programming Languages and Systems (QAPL 2012)*, volume 85

of *Electronic Proceedings in Theoretical Computer Science*, pages 1–16, Tallinn, Estonia, July 2012.

6 Radu Calinescu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, 2012.

7 Taolue Chen, Marco Diciolla, Marta Kwiatkowska, and Alexandru Mereacre. Quantitative Verification of Implantable Cardiac Pacemakers. In *Real-time Systems Symposium (RTSS 2012)*, pages 263–272. IEEE, December 2012.

8 Taolue Chen, Ernst Moritz Hahn, Tingting Han, Marta Kwiatkowska, Hongyang Qu, and Lijun Zhang. Model Repair for Markov Decision Processes. In *Theoretical Aspects of Software Engineering (TASE 2013)*, pages 85–92. IEEE, July 2013.

9 Alexandre David, Dehui Du, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical Model Checking for Stochastic Hybrid Systems. In *Electronic Proceedings in Theoretical Computer Science*, volume 92, pages 122–136, August 2012.

10 Arvind Easwaran, Sampath Kannan, and Oleg Sokolsky. Steering of Discrete Event Systems: Control Theory Approach. *Electronic Notes in Theoretical Computer Science*, 144(4):21–39, 2006.

11 Thomas A. Henzinger. The Theory of Hybrid Automata. In *Logic in Computer Science, 1996. LICS'96*, pages 278–292. IEEE, 1996.

12 Zhihao Jiang, Miroslav Pajic, Salar Moarref, Rajeev Alur, and Rahul Mangharam. Modeling and Verification of a Dual Chamber Implantable Pacemaker. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, pages 188–203. Springer Berlin Heidelberg, 2012.

13 Andrew King, Dave Arney, Insup Lee, Oleg Sokolsky, John Hatcliff, and Sam Procter. Prototyping Closed Loop Physiologic Control with the Medical Device Coordination Framework. In *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care (SEHC 2010)*, pages 1–11. ACM, 2010.

14 Tao Li, Feng Tan, Qixin Wang, Lei Bu, Jian-Nong Cao, and Xue Liu. From Offline toward Real-Time: A Hybrid Systems Model Checking and CPS Co-design Approach for Medical Device Plug-and-Play (MDPnP). In *Proceedings of the 3rd ACM/IEEE International Conference on Cyber-Physical Systems – ICCPS '12*, pages 13–22, Beijing, China, April 2012. IEEE.

15 Tao Li, Qixin Wang, Feng Tan, Lei Bu, Jian-nong Cao, Xue Liu, Yufei Wang, and Rong Zheng. From Offline Long-Run to Online Short-Run: Exploring A New Approach of Hybrid Systems Model Checking for MDPnP. In *Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability (HCMDSS-MDPnP 2011)*, 2011.

16 Xintao Ma. Online Checking of a Hybrid Laser Tracheotomy Model in UPPAAL-SMC. Master thesis, TU Hamburg-Harburg, December 2013.

17 Jonas Rinast, Sibylle Schupp, and Dieter Gollmann. State Space Reconstruction for On-Line Model Checking with UPPAAL. *VALID 2013, The Fifth Internation Conference on Advances in System Testing and Validation Lifecycle*, pages 21–26, 2013.

18 Yuhong Zhao and Franz Rammig. Online Model Checking for Dependable Real-Time Systems. In *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 154–161. IEEE, April 2012.