Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2014-IJ-001

From Offline towards Real-time: A Hybrid System model Checking and CPS Co-Design Approach for Medical Device Plug-and-Play Collaborations.

Tao Li Feng Tan Qixin Wang Lei Bu Jian-nong Cao Xue Liu

IEEE Transactions on Parallel and Distributed Systems (Volume:25 , Issue: 3)

From Offline toward Real-Time: A Hybrid Systems Model Checking and CPS Co-Design Approach for Medical Device Plug-and-Play Collaborations

Tao Li, *Student Member, IEEE,* Feng Tan, *Student Member, IEEE,* Qixin Wang, *Member, IEEE,* Lei Bu, *Member, IEEE,* Jian-nong Cao, *Senior Member, IEEE,* and Xue Liu, *Member, IEEE*

Abstract—Hybrid systems model checking is a great success in guaranteeing the safety of computerized control *cyber-physical systems* (CPS). However, when applying hybrid systems model checking to *Medical Device Plug-and-Play* (MDPnP) CPS, we encounter two challenges due to the complexity of human body: i) there are no good offline differential equation based models for many human body parameters; ii) the complexity of human body can result in many variables, complicating the system model. In an attempt to address the challenges, we propose to alter the traditional approach of *offline* hybrid systems model checking of time-*un*bounded (i.e., *infinite-horizon*, a.k.a., long-run) future behavior to *online* hybrid systems model checking of time-*bounded* (i.e., *finite-horizon*, a.k.a., short-run) future behavior. According to this proposal, online model checking runs as a real-time task to prevent faults. To meet the real-time requirements, certain design patterns must be followed, which brings up the *co-design* issue. We propose two sets of system co-design patterns for hard real-time and soft real-time respectively. To evaluate our proposals, a case study on laser tracheotomy MDPnP is carried out. The study shows the necessity of online model checking. Furthermore, test results based on real-world human subject trace show the feasibility and effectiveness of our proposed co-design.

Index Terms—Cyber-physical systems, medical device plug-and-play, hybrid systems model checking, real-time.

1 Introduction

Thanks to the rapid development of embedded systems technology, we now have thousands of kinds of embedded medical devices. So far, these devices are mainly designed for isolated use. However, people envision that by coordinating these devices, we can significantly increase medical treatment safety, capability, and efficiency. This vision led to the launch of the *Medical Device Plug-and-Play* (MDPnP) [1] effort, which aims to enable the safe composition and collaboration of disparate embedded devices in medical contexts. An MDPnP system is a typical *Cyber-Physical System* (CPS) [2]. On the one hand, it involves cyber-world discrete computer logic of various embedded medical devices. On the other hand, it involves physical-world patient-in-the-loop, which is a continuous complex biochemical system.

The top concern of any MDPnP system is safety. In the cyber-world, for a safety-critical system, people often carry out model checking [3] *before* the system is put online. In such case, model checking builds an *offline* model of the system, and checks the system's possible behaviors in the *time-unbounded future* (i.e., *infinite-horizon*). Only after passing model checking may the system be allowed to run.

This practice is a great success. For CPS verification, the state-of-the-art model checking tools are the *hybrid systems* model checking tools [4][5], which integrate the discrete automata models with the continuous differential equation (and other control theory) models. Today, hybrid systems model checking can already analyze many computerized control systems, i.e., control CPS.

The success of hybrid model checking in control CPS inspires the interest to apply it in MDPnP CPS. However, this faces a major challenge: in most MDPnP CPS, there are *no* good *offline* models to describe the complex biochemical system of the patient [6]. Even if some vital signs can be modeled offline, the models may not (with some exceptions [7]) fit into existing hybrid systems model checking tools, which mainly use linear differential equations to describe the physical world.

To deal with the above challenges, we propose to alter the traditional practice of *offline* model checking of hybrid system's behavior in the *infinite-horizon*. Instead, we carry out periodical *online* model checking. In every period, we only model check the hybrid system's behavior in the next (few) period(s); i.e., we only model check the hybrid system's behavior in *time-bounded future* (i.e., *finite-horizon*).

The merits of the proposed approach are as follows. First, though many human body parameters are hard to model

- *T. Li, F. Tan, Q. Wang, and J. Cao are with Dept. of Computing, The Hong Kong Polytechnic Univ., Hong Kong S. A. R. Email: {cstaoli, csftan, csqwang, csjcao}@comp.polyu.edu.hk*
- *L. Bu is with State Key Lab for Novel Software Technology, Dept. of Computer Sci. and Tech., Nanjing Univ., Nanjing, 210093, P. R. China. Email: bulei@nju.edu.cn*
- *X. Liu is with School of Computer Science, McGill Univ., Montreal, Canada. Email: xue.liu@mcgill.ca*
- *Corresponding Authors: Qixin Wang, Lei Bu.*

offline, their online behaviors in finite-horizon are quite predictable. For example, after injecting 1ml of morphine, it is hard to accurately predict the blood oxygen level curve in the next 40 minutes, as it depends on too many factors, even including the patient's emotion [8][9]. However, it is easy to predict the blood oxygen level curve in the next 4 seconds: it cannot plunge from 100% to 10%, nor show a saw-toothed wave form; instead, it has to be smooth, which can be effectively described with existing tools, such as linear regression. Also, within short finite-horizon, we can approximate many variables as constants, and/or approximate nonlinear behaviors as linear behaviors. This would further simplify our model and computation.

The proposed approach can be formalized as follows. Given an MDPnP system $\mathcal{S}$, we periodically sample the observable state parameters every T seconds. At time instance kT ($k = 0, 1, 2, \ldots$), we build a hybrid system model (i.e., the "online model") of $\mathcal{S}$ with the observed numerical values of state parameters, and verify its safety in the time interval $[kT, (k + 1)T]$. We hence call T the *finite-horizon* of our online model checking. If the online model is proven safe, the system can run for another T seconds. Otherwise, the system immediately switches to an application dependant fall-back plan.

Such model checking must finish within bounded and short time, i.e. *real-time*, to allow decision making (on whether to run the system for another T seconds or switch to fall-back plan) *before* any fault happens. To support real-time, the MDPnP CPS design must follow certain patterns, which brings up the issue of hybrid systems model checking and CPS *co-design*.

In the rest of the paper, we discuss our proposed co-design approach through the context of laser tracheotomy, a representative MDPnP application [7][10]. Section 2 introduces the background on hybrid systems model checking; Section 3 proposes our online hybrid systems modeling approach; Section 4 proposes the corresponding system design patterns; Section 5 evaluates our approach; Section 6 further examines our proposal under relaxed assumptions; Section 7 discusses related work; and Section 8 concludes the paper.

This paper is based on our previous conference paper published in [11], which is in turn based on our workshop paper published in [12][13]. Compared to these previous versions, this paper mainly added Section 4.1, Theorem 3, Section 6.2, and the supplementary file.

2 Background

Hybrid systems model checking is first proposed by Alur, Henzinger, et al. [14][15][16] and has since evolved into a family of state-of-the-art tools in CPS. The main idea is to combine the discrete automata models of computer logic with continuous differential equation models of control systems, which leads to the modeling tool of *hybrid automata*.

2.1 Syntax

Following [15]'s conventions on symbols, a hybrid automaton A is syntactically a tuple of $A = (\vec{x}, \vec{x}^0, V, v^0, inv, dif, E, act, L, syn)$, where

$\vec{x}$ is a vector of n *data variables* $\vec{x} = (x_1, x_2, \ldots, x_n)$. $\vec{x}$ is regarded as a function of time, and we use $\dot{\vec{x}}$ to denote the first order derivative of $\vec{x}$. We also use $\vec{x}' = (x'_1, x'_2, \ldots, x'_n)$ to denote the new values of $\vec{x}$ after an event (see the definitions for E and act). A specific evaluation of $\vec{x}$, denoted as $\vec{s} = (s_1, s_2, \ldots, s_n) \in \mathbb{R}^n$ is called a *data state* of A. In addition, Boolean values of **true** and **false** can be denoted with real number 1 and 0 respectively; hence a data variable can also serve as a Boolean variable.

$\vec{x}^0$ is the initial data state.

V is a set of *locations*, a.k.a., *control locations*, where different control laws apply. Each location corresponds to a vertex in the graphical representation of hybrid automaton A. A *state* of hybrid automaton A is denoted as $(v, \vec{s})$, where $v \in V$ and $\vec{s} \in \mathbb{R}^n$ is a data state.

v^0 is the initial location.

inv is the *location invariants*, a function that assigns each location $v \in V$ a set of inequalities over data variables $\vec{x}$. That is, when in location v, the value of $\vec{x}$ must satisfy $inv(v)$.

dif is the *continuous activities*, a function that assigns each location $v \in V$ a set of inequalities over $\dot{\vec{x}}$ and $\vec{x}$. That is, when in location v, the values of $\dot{\vec{x}}$ and $\vec{x}$ must satisfy $dif(v)$.

E is the set of *events*, a.k.a. *transitions*: edges between locations. Formally, $E \subseteq V \times V$. For an event $e = (v, v') \in E$, v is the *source location* and v' is the *target location*.

act is the *discrete actions*, a function assigns to each event $e = (v, v') \in E$ a set of inequalities over $\vec{x}$ and $\vec{x}'$, where $\vec{x}' = (x'_1, x'_2, \ldots, x'_n)$ refers to the new value of $\vec{x}$ after event e. The event $e = (v, v')$ is enabled only when the value of $\vec{x}$ in v satisfies $act(e)$, and the new value of $\vec{x}'$ after the event is chosen nondeterministically such that $act(e)$ is satisfied. For example, suppose $\vec{x} = (x_1)$, then for $act(e) = (x_1 \leq 3 \wedge x'_1 \leq 5 \wedge x'_1 \geq 5)$, event e is only enabled when $x_1 \leq 3$; and after the event, x_1 is assigned the new value of 5. Like this example, if $\vec{x}$ and $\vec{x}'$ do not mix in any inequalities in $act(e)$, and $\vec{x}'$ has a deterministic value $\vec{s}'$, then we can call the subset of inequalities involving only $\vec{x}$ to be the *guard* of event e, and event e *updates* $\vec{x}$ to $\vec{s}'$, denoted as $\vec{x} := \vec{s}'$.

L is a set of *synchronization labels*.

syn is the *synchronization function* that assigns each event $e \in E$ an $l \in L$. L and syn are for composition of multiple hybrid automata. Suppose we have two hybrid automata $A_1 = (\vec{x}_1, \vec{x}^0_1, V_1, v^0_1, inv_1, dif_1, E_1, act_1, L_1, syn_1)$ and $A_2 = (\vec{x}_2, \vec{x}^0_2, V_2, v^0_2, inv_2, dif_2, E_2, act_2, L_2, syn_2)$, if $e_1 \in E_1$, $e_2 \in E_2$ and $syn_1(e_1) = syn_2(e_2)$, then event e_1 and e_2 must always take place together.

Furthermore, when inv, dif, and act only involve linear inequalities, and dif does not involve $\vec{x}$, hybrid automaton A is called *linear hybrid automaton* (LHA)[14].

Reference [15] also describes how to combine several hybrid automata into one hybrid automaton. Particularly, the location set of the combined hybrid automaton $V_{comb} = V_1 \times V_2 \times \ldots \times V_n$, where V_i ($i = 1, \ldots, n$) is the location set of the ith component hybrid automaton; and "$\times$" is Cartesian product. For $v \in V_{comb}$, we use $v|_i$ to denote the projection of v on V_i.

2.2 Semantics

This paper adopts the semantic concepts and the corresponding symbol definitions of [15]. Due to page limit, interested readers shall refer to [15] for these definitions. Of particular importance are the concepts of *state predicate*, *trajectory*, *number of hops (of a trajectory)*, *non-blocking*, *non-zeno*.

We, however, want to emphasize that to simplify narration, in the following, unless explicitly denoted, "model checking" refers to "*model checking of finite-horizon reachability semantics*", i.e., whether a state σ of hybrid automaton A satisfies $\varphi_1 \exists \mathcal{U}_{\leq T} \varphi_2$, where φ_1 and φ_2 are state predicates of A, and T is the *finite-horizon*. Also, unless explicitly denoted, *we only discuss non-blocking hybrid automata*.

3 Hybrid Systems Modeling Approach

In this section, we shall use laser tracheotomy, a representative MDPnP application [7][10], as the context to discuss the proper hybrid systems modeling approach for MDPnP. We shall see through this case study why offline model checking must be replaced by online model checking.

Laser tracheotomy MDPnP interlocks various medical devices to increase safety. It has the following entities (see Fig. 1):

Patient: the patient that receives the surgery;
O_2 **Sensor:** the patient's trachea oxygen level sensor;
SpO_2 **Sensor:** the patient's blood oxygen level sensor;
Ventilator: the medical device that administrates the patient's respirations;
Surgeon: the doctor that conducts the surgery;
Laser Scalpel: the medical device for the surgeon to cut the patient's trachea;
Supervisor: the central computer that connects all medical devices and makes decisions to guarantee safety.

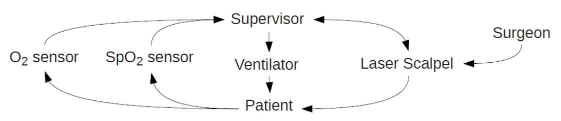


Fig. 1. Layout of Laser Tracheotomy MDPnP

The application context is as follows. In the surgery, due to general anesthesia, the patient is paralyzed, hence has to depend on the ventilator to breathe. The ventilator has three modes: pumping out (the patient inhales oxygen), pumping in (the patient exhales), and hold (the patient exhales naturally due to chest weight). However, when the laser scalpel is to cut the patient's trachea, the oxygen level inside the trachea must be lower than a threshold. Otherwise, the laser may trigger fire. Therefore, before the laser scalpel is allowed to emit laser, the ventilator must have stopped pumping out (oxygen) for a while. On the other hand, the ventilator can neither stop pumping out for too long, or the patient will suffocate due to too low blood oxygen level.

In summary, the laser tracheotomy MDPnP must avoid the following safety hazards:
Safety Hazard 1: when the laser scalpel emits laser, the patient's trachea oxygen level exceeds a threshold Θ_{O_2};
Safety Hazard 2: the patient's blood oxygen level reaches below a threshold Θ_{SpO_2}.
Note that the setting of constant thresholds Θ_{O_2} and Θ_{SpO_2} are medical experts' responsibility and are beyond the coverage of this paper.

The formal expressions of safety hazards will become clear by the end of Section 3.2, when the corresponding hybrid automata are defined.

3.1 Traditional Approach: Offline Modeling

Because the laser tracheotomy MDPnP involves both discrete medical device logic and physical world patient, it is a hybrid system. Therefore we try to model laser tracheotomy MDPnP with hybrid automata.

The traditional approach of model checking, including hybrid systems model checking, is carried out offline. That is, the model is built and its infinite-horizon behavior is verified *before* the system runs. We choose to start with this approach. As a common practice, our offline modeling of laser tracheotomy MDPnP assumes a global time t: t is initialized to 0 second, and $\dot{t} \equiv 1$.

Intuitively, we intend to start with modeling the patient, the core entity of the laser tracheotomy MDPnP. However, the patient's behavior is directly administrated by the ventilator, which has to be understood first.

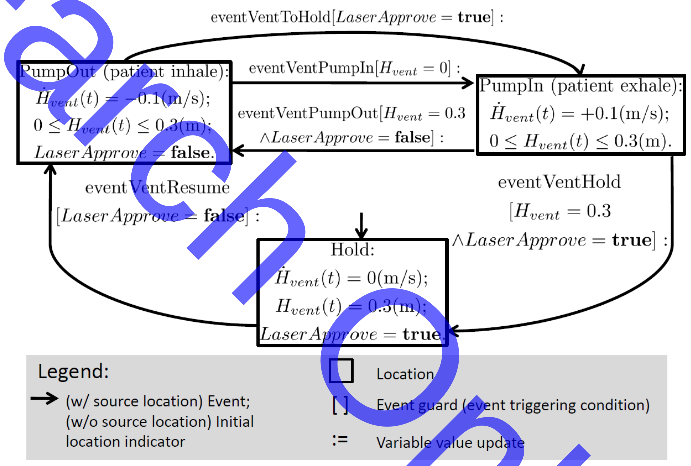

eventVentToHold[$LaserApprove = \textbf{true}$] :
PumpOut (patient inhale): $\dot{H}_{vent}(t) = -0.1$(m/s); $0 \leq H_{vent}(t) \leq 0.3$(m); $LaserApprove = \textbf{false}$.
eventVentPumpIn[$H_{vent} = 0$] :
PumpIn (patient exhale): $\dot{H}_{vent}(t) = +0.1$(m/s); $0 \leq H_{vent}(t) \leq 0.3$(m).
eventVentPumpOut[$H_{vent} = 0.3$ $\wedge LaserApprove = \textbf{false}$] :
eventVentResume [$LaserApprove = \textbf{false}$] :
eventVentHold [$H_{vent} = 0.3$ $\wedge LaserApprove = \textbf{true}$] :
Hold: $\dot{H}_{vent}(t) = 0$(m/s); $H_{vent}(t) = 0.3$(m); $LaserApprove = \textbf{true}$.
Legend:
(w/ source location) Event; (w/o source location) Initial location indicator
Location
Event guard (event triggering condition)
Variable value update

Fig. 2. Offline hybrid automaton of Ventilator

The ventilator is basically a compressible air reservoir [17]: a cylinder of height $H_{vent}(t)$ $(0 \leq H_{vent}(t) \leq 0.3(\text{m}))$. The movement of the ventilator cylinder (indicated by $\dot{H}_{vent}(t)$) pumps out/in oxygen/air to/from patient, thus helping the patient to inhale/exhale. The ventilator behavior is defined by the hybrid automaton in Fig. 2. The automaton has three locations: PumpOut, PumpIn, and Hold. When the supervisor (will be discussed later in Fig. 8) allows the ventilator to work (i.e., when data variable $LaserApprove$ is set to **false**), the ventilator switches between pumping out (where $\dot{H}_{vent} = -0.1\text{m/s}$) and pumping in (where $\dot{H}_{vent} = +0.1\text{m/s}$). This causes the patient to inhale oxygen and exhale respectively. When the supervisor pauses the ventilator (i.e., when $LaserApprove$ is set to **true**), the ventilator cylinder will try to restore to its maximum height (0.3m) and holds there until the ventilator is allowed again ($LaserApprove$ set to **false**).

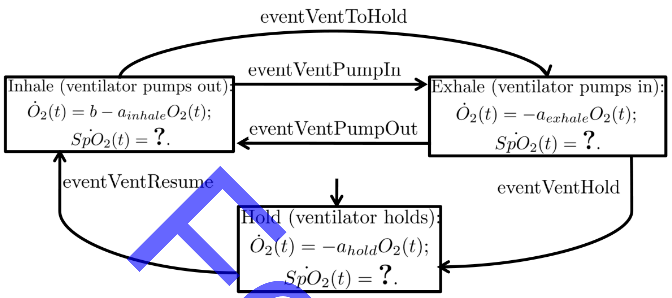

eventVentToHold
eventVentPumpIn
Inhale (ventilator pumps out): $\dot{O}_2(t) = b - a_{inhale}O_2(t)$; $\dot{SpO_2}(t) = ?$.
eventVentPumpOut
Exhale (ventilator pumps in): $\dot{O}_2(t) = -a_{exhale}O_2(t)$; $\dot{SpO_2}(t) = ?$.
eventVentResume
eventVentHold
Hold (ventilator holds): $\dot{O}_2(t) = -a_{hold}O_2(t)$; $\dot{SpO_2}(t) = ?$.

Fig. 3. Offline hybrid automaton of Patient. Though good offline models for $\dot{O}_2$ exists [7], the offline model for $\dot{SpO_2}$ is still an open problem. Also note that in location Hold (which corresponds to ventilator Hold), the patient still exhale due to chest weight.

With the ventilator hybrid automaton at hand, we can now start modeling the patient. The patient hybrid automaton (see Fig. 3) is tightly coupled with the ventilator hybrid automaton (see Fig. 2). It also has three locations: Inhale, Exhale, and Hold, which respectively correspond to the ventilator hybrid automaton's locations of PumpOut, PumpIn, and Hold. The events between the three locations are also triggered by corresponding events from the ventilator hybrid automaton.

Inside of each location are the offline continuous time models for trachea oxygen level $O_2(t)$ and blood oxygen level $SpO_2(t)$. Unfortunately, though there are good offline models for $\dot{O}_2(t)$ [7], the offline model for $\dot{SpO_2}(t)$ is still an open problem [8][9]. This is because blood oxygen level are strongly affected by complex human body biochemical reactions, even emotions.

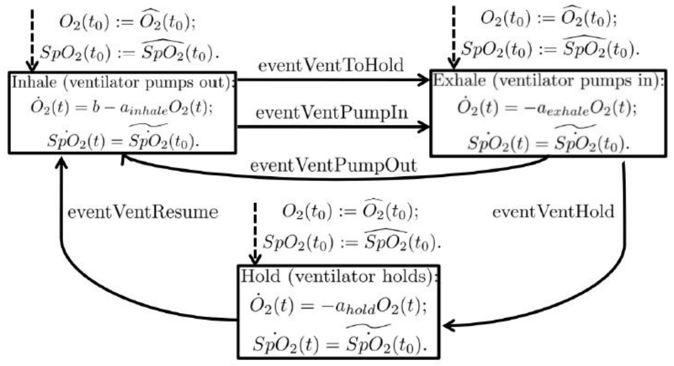
Therefore, we fail to model $SpO_2(t)$ offline, and hence fail to model the patient offline. What is worse, as the patient model is an indispensable component of the holistic offline model, *the offline model checking of laser tracheotomy MDPnP fails.*

3.2 Proposed Approach: Online Modeling

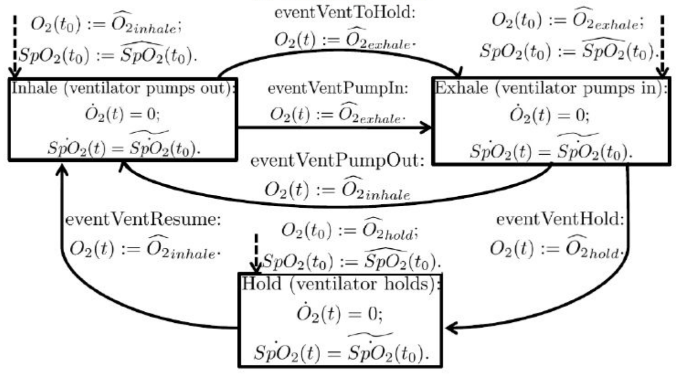
The failure of offline approach forces us to consider the proposed online approach (see Section 1) instead. Specifically, we sample the patient's trachea/blood oxygen level every T seconds. Suppose at $t_0 = kT$ ($k \in \mathbb{Z}_{\geq 0}$), we get the most up-to-date trachea/blood oxygen level sensor reading $\widehat{O_2}(t_0)$ and $\widehat{SpO_2}(t_0)$, we can then build the hybrid systems model for interval $[t_0, t_0+T]$, where T is therefore the *finite-horizon*. This model is built as follows.

First, same as the offline model checking, we use global variable t to represent the global clock, except that now t is initialized to t_0 and stops at $(t_0 + T)$ as we only care about the system's finite-horizon safety until $(t_0 + T)$.

The patient hybrid automaton now looks like Fig. 4(a). The biggest change is the continuous time model for the blood oxygen level $SpO_2(t)$. In offline model checking, we have to describe the infinite-horizon behavior of $SpO_2(t)$, which is an open problem. However, in online model checking, we only have to describe $SpO_2(t)$'s behavior in interval $[t_0, t_0 + T]$,


$O_2(t_0) := \widehat{O_2}(t_0)$; $SpO_2(t_0) := \widehat{SpO_2}(t_0)$.
eventVentToHold
Inhale (ventilator pumps out): $\dot{O}_2(t) = b - a_{inhale}O_2(t)$; $\dot{SpO_2}(t) = \widetilde{\dot{SpO_2}}(t_0)$.
eventVentPumpIn
Exhale (ventilator pumps in): $\dot{O}_2(t) = -a_{exhale}O_2(t)$; $\dot{SpO_2}(t) = \widetilde{\dot{SpO_2}}(t_0)$.
eventVentPumpOut
eventVentResume
eventVentHold
Hold (ventilator holds): $\dot{O}_2(t) = -a_{hold}O_2(t)$; $\dot{SpO_2}(t) = \widetilde{\dot{SpO_2}}(t_0)$.

(a) non-linear model


$O_2(t_0) := \widehat{O_2}_{inhale}$; $SpO_2(t_0) := \widehat{SpO_2}(t_0)$.
eventVentToHold: $O_2(t) := \widehat{O_2}_{exhale}$.
$O_2(t_0) := \widehat{O_2}_{exhale}$; $SpO_2(t_0) := \widehat{SpO_2}(t_0)$.
Inhale (ventilator pumps out): $\dot{O}_2(t) = 0$; $\dot{SpO_2}(t) = \widetilde{\dot{SpO_2}}(t_0)$.
eventVentPumpIn: $O_2(t) := \widehat{O_2}_{exhale}$.
Exhale (ventilator pumps in): $\dot{O}_2(t) = 0$; $\dot{SpO_2}(t) = \widetilde{\dot{SpO_2}}(t_0)$.
eventVentPumpOut: $O_2(t) := \widehat{O_2}_{inhale}$
eventVentResume: $O_2(t) := \widehat{O_2}_{inhale}$.
$O_2(t_0) := \widehat{O_2}_{hold}$; $SpO_2(t_0) := \widehat{SpO_2}(t_0)$.
eventVentHold: $O_2(t) := \widehat{O_2}_{hold}$.
Hold (ventilator holds): $\dot{O}_2(t) = 0$; $\dot{SpO_2}(t) = \widetilde{\dot{SpO_2}}(t_0)$.

(b) *linear hybrid automaton* (LHA) model (see Section 2.1 for definition of LHA), where $\widehat{O_2}_{inhale}$, $\widehat{O_2}_{exhale}$, and $\widehat{O_2}_{hold}$ are constants, which can be estimated from historical data.

Fig. 4. Online hybrid automaton of Patient.

where the finite-horizon T is just a few seconds. If we only look into such short-run future, blood oxygen level curve $SpO_2(t)$ is very describable and predictable. For example, it cannot plunge from 100% to 10% within just 4 seconds, neither can it show a saw-toothed wave form. Instead, it must be smooth; in fact smooth enough to be safely predicted with standard tools (such as linear regression) based on its past history.

In Fig. 4(a), we use a simple way to predict/describe $SpO_2(t)$ in $t \in [t_0, t_0 + T]$:

$$\dot{SpO_2}(t) \equiv \widetilde{\dot{SpO_2}}(t_0), \qquad \forall t \in [t_0, t_0 + T],$$

where $\dot{SpO_2}(t)$ is the derivative of $SpO_2(t)$ at time t; and $\widetilde{\dot{SpO_2}}(t_0)$ is the estimation (e.g., via linear regression) of $\dot{SpO_2}(t_0)$ based on $SpO_2(t)$'s history recorded during $(t_0 - T_{past}, t_0)$. T_{past} is a configuration constant picked empirically offline. In our case study, we pick $T_{past} = 6$ seconds.

Also, depending on the patient's state at time t_0, the initial location can be Inhale, Exhale, or Hold. Whichever location it is, the initial value of trachea/blood oxygen value should be $\widehat{O_2}(t_0)$ and $\widehat{SpO_2}(t_0)$ respectively.

The patient model of Fig. 4(a) can be further simplified. Human subject respiration traces (see Fig. 5) show that the values of a_{inhale}, a_{exhale}, and a_{hold} in Fig. 4(a) are large: so large that $O_2(t)$ almost behaves as rectangular waves when the patient hybrid automaton changes locations. Therefore, we can simplify Fig. 4(a) into Fig. 4(b), where $O_2(t)$ remains constant within every location, and its value is only updated on the corresponding transitions. This simplification turns the patient hybrid automaton (in fact the whole system) into an *linear hybrid automaton* (LHA) (see Section 2.1 for definition of LHA), which is much easier to verify [18].

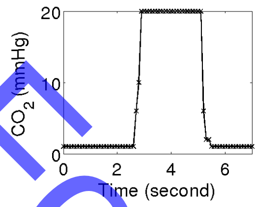
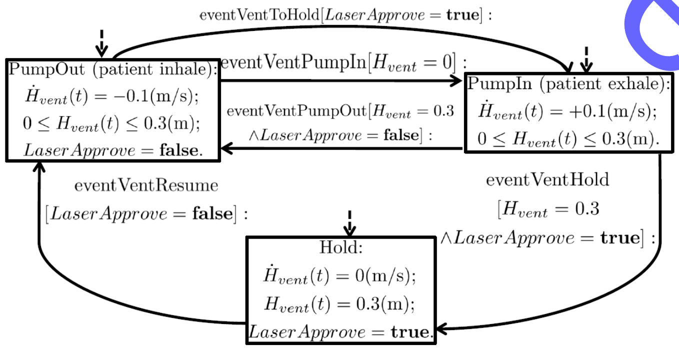


Fig. 5. A typical example excerpt of trachea CO_2 level trace (measured on human subjects with Nonin 9843 [19]); note $O_2(t) = C_1 - C_2 \cdot CO_2(t)$, where C_1 and C_2 are two constants, whose derivation can be found in classic physics textbooks [20].

We now check other laser tracheotomy MDPnP entities.

First, since the online model only looks into the finite-horizon of $[t_0, t_0 + T]$, where T is also the sensor sampling period, there are no interactions with sensors throughout the interval of $(t_0, t_0 + T)$. Therefore, in online model checking, the hybrid automata of O_2 sensor and SpO_2 sensor are unnecessary.



eventVentToHold[$LaserApprove = \textbf{true}$] :
PumpOut (patient inhale):
$\dot{H}_{vent}(t) = -0.1(\text{m/s})$;
$0 \le H_{vent}(t) \le 0.3(\text{m})$;
$LaserApprove = \textbf{false}$.
eventVentPumpIn[$H_{vent} = 0$] :
PumpIn (patient exhale):
$\dot{H}_{vent}(t) = +0.1(\text{m/s})$;
$0 \le H_{vent}(t) \le 0.3(\text{m})$.
eventVentPumpOut[$H_{vent} = 0.3$ $\wedge LaserApprove = \textbf{false}$] :
eventVentResume
[$LaserApprove = \textbf{false}$] :
eventVentHold
[$H_{vent} = 0.3$
$\wedge LaserApprove = \textbf{true}$] :
Hold:
$\dot{H}_{vent}(t) = 0(\text{m/s})$;
$H_{vent}(t) = 0.3(\text{m})$;
$LaserApprove = \textbf{true}$.

Fig. 6. Online hybrid automaton of Ventilator.

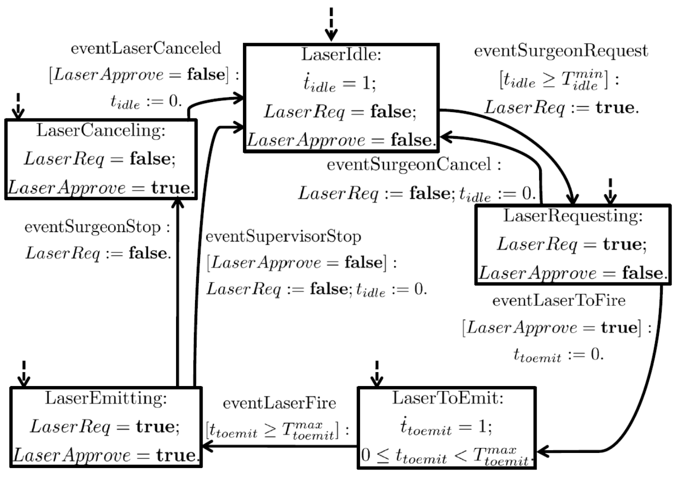
Next, the ventilator hybrid automaton in online model (see Fig. 6) is almost the same as its offline counterpart (see Fig. 2) A main difference is that the online model's initial location can be any location depending on the ventilator's state at t_0.

The last entity that directly interacts with the patient is the laser scalpel. We can actually model the laser scalpel and the surgeon with one hybrid automaton: the laser scalpel hybrid automaton (see Fig. 7).

The automaton's key elements are the two Boolean variables: $LaserApprove$ and $LaserReq$.

$LaserApprove$ indicates whether the supervisor (see Fig. 1) allows the laser scalpel to emit laser (**true** for yes and **false** for no). Its value can only be set by the supervisor hybrid automaton (see Fig. 8), which is to be explained later.

$LaserReq$ indicates whether the laser scalpel wants to emit laser (**true** for yes and **false** for no). Its value can only be set by the laser scalpel hybrid automaton. The value setting is triggered by following events: *i*) when in LaserIdle, the surgeon can request emitting laser through eventSurgeonRequest,



eventLaserCanceled
[$LaserApprove = \textbf{false}$] :
$t_{idle} := 0$.
LaserIdle:
$\dot{t}_{idle} = 1$;
$LaserReq = \textbf{false}$;
$LaserApprove = \textbf{false}$.
eventSurgeonRequest
[$t_{idle} \ge T_{idle}^{min}$] :
$LaserReq := \textbf{true}$.
LaserCanceling:
$LaserReq = \textbf{false}$;
$LaserApprove = \textbf{true}$.
eventSurgeonCancel :
$LaserReq := \textbf{false}; t_{idle} := 0$.
eventSurgeonStop :
$LaserReq := \textbf{false}$.
eventSupervisorStop
[$LaserApprove = \textbf{false}$] :
$LaserReq := \textbf{false}; t_{idle} := 0$.
LaserRequesting:
$LaserReq = \textbf{true}$;
$LaserApprove = \textbf{false}$.
eventLaserToFire
[$LaserApprove = \textbf{true}$] :
$t_{toemit} := 0$.
LaserEmitting:
$LaserReq = \textbf{true}$;
$LaserApprove = \textbf{true}$.
eventLaserFire
[$t_{toemit} \ge T_{toemit}^{max}$] :
LaserToEmit:
$\dot{t}_{toemit} = 1$;
$0 \le t_{toemit} < T_{toemit}^{max}$

Fig. 7. Online hybrid automaton of Laser Scalpel. This is the only automaton that sets the value of state variable $LaserReq$.

which sets $LaserReq$ to **true**; *ii*) when in LaserRequesting or LaserEmitting, the surgeon can request stopping laser emission through eventSurgeonCancel and eventSurgeonStop respectively, which both set $LaserReq$ to **false**; *iii*) when in LaserEmitting, the supervisor can stop the laser emission at any time by setting $LaserApprove$ to **false**, which triggers eventSupervisorStop and sets $LaserReq$ to **false**.

The four possible combinations of $LaserApprove$ and $LaserReq$'s values define the major locations in the laser scalpel hybrid automaton: LaserIdle, LaserRequesting, LaserEmitting, and LaserCanceling. Particularly, laser scalpel emits laser in and only in LaserEmitting. There is an additional location, LaserToEmit, which models the additional delay T_{toemit}^{max} between LaserRequesting and LaserEmitting. This delay is to further ensure oxygen level in trachea falls below threshold before the actual laser emission.

The laser scalpel hybrid automaton's initial location can be anywhere depending on the laser scalpel's state at t_0. One thing to note is that all variables should be initialized to their actual value at t_0. For example, if initial location is LaserIdle, and Laser Scalpel has been idling for 10 seconds by t_0, then t_{idle} shall be initialized to 10 seconds instead of 0.

Finally, all medical device entities are interlocked by the supervisor, the central decision making computer (see Fig. 1). The supervisor maneuvers data variable $LaserApprove$. Setting $LaserApprove$ to **true/false** determines the off/on of the ventilator and the permission/denial of emitting laser respectively.

The value setting decisions are made dependent on the most up-to-date information on the patient's trachea oxygen level $O_2(t)$ and blood oxygen level $SpO_2(t)$. Based on the models given in the patient hybrid automaton (see Fig. 4), we can predict $O_2(t)$ and $SpO_2(t)$ for any $t \in [t_0, t_0 + T]$. Therefore, we can construct the supervisor hybrid automaton as Fig. 8, which directly uses $O_2(t)$ and $SpO_2(t)$ predicted by the patient hybrid automaton for decision making.

The supervisor hybrid automaton has two locations: LaserDisapproved and LaserApproved.

When in LaserDisapproved, the supervisor needs eventSu-

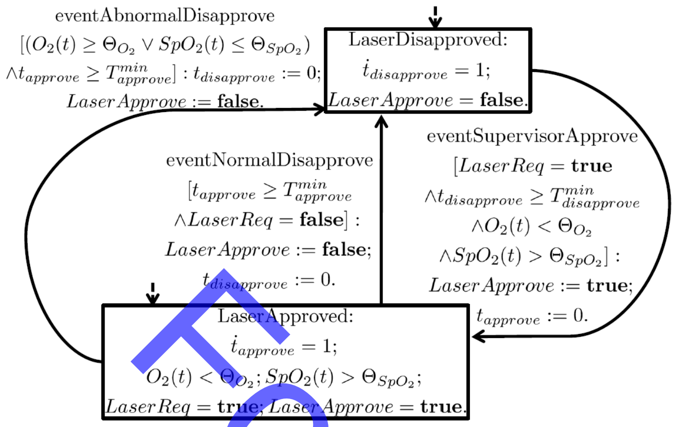

eventAbnormalDisapprove
$[(O_2(t) \geq \Theta_{O_2} \vee SpO_2(t) \leq \Theta_{SpO_2})$
$\wedge t_{approve} \geq T^{min}_{approve}] : t_{disapprove} := 0;$
$LaserApprove :=$ **false**.
LaserDisapproved:
$\dot{t}_{disapprove} = 1;$
$LaserApprove =$ **false**.
eventSupervisorApprove
$[LaserReq =$ **true**
$\wedge t_{disapprove} \geq T^{min}_{disapprove}$
$\wedge O_2(t) < \Theta_{O_2}$
$\wedge SpO_2(t) > \Theta_{SpO_2}] :$
$LaserApprove :=$ **true**;
$t_{approve} := 0.$
eventNormalDisapprove
$[t_{approve} \geq T^{min}_{approve}$
$\wedge LaserReq =$ **false**$] :$
$LaserApprove :=$ **false**;
$t_{disapprove} := 0.$
LaserApproved:
$\dot{t}_{approve} = 1;$
$O_2(t) < \Theta_{O_2}; SpO_2(t) > \Theta_{SpO_2};$
$LaserReq =$ **true**; $LaserApprove =$ **true**.

Fig. 8. Online hybrid automaton of Supervisor. This is the only automaton that sets the value of data variable $LaserApprove$. Note $t_{approve}$ can be totally removed from the model in *soft* real-time online model checking.

pervisorApprove to move to LaserApproved. This event is triggered when the following prerequisites all hold:

Prerequisite 1: the laser scalpel is requesting emitting laser (i.e., $LaserReq =$ **true**);

Prerequisite 2: $O_2(t)$ is less than threshold Θ_{O_2};

Prerequisite 3: $SpO_2(t)$ is greater than threshold Θ_{SpO_2}.

Prerequisite 4: $t_{disapprove} \geq T^{min}_{disapprove}$. This is a minimal dwelling time requirement to guarantee the automaton's non-zeno property. The purpose will become clear in a latter example (Example 1 of Appendix B in the supplementary file). This requirement also models the time cost in switching between LaserDisapproved and LaserApproved modes in the supervisor.

Through eventSupervisorApprove, the supervisor approves the emission of laser by setting $LaserApprove$ to **true**. This event also resets a clock $t_{approve}$, and moves the location to LaserApproved.

Like $t_{disapprove}$, clock $t_{approve}$ is for guaranteeing a minimal dwelling time of $T^{min}_{approve}$ in LaserApproved. After that, if Prerequisite 1 no longer holds (i.e., when $LaserReq$ becomes **false**), the eventNormalDisapprove is triggered. This event moves the supervisor back to location LaserDisapproved and resets $LaserApprove$ to **false**, and $t_{disapprove}$ to 0.

In contrast to eventNormalDisapprove, event*Abnormal*Disapprove is triggered when the supervisor is in LaserApproved while Prerequisite 2 or 3 stops to hold. This event also moves the supervisor back to location LaserDisapproved and resets $LaserApprove/t_{disapprove}$ to **false**/0 respectively.

Finally, same as the other online hybrid automata, the initial location for the online supervisor automaton can be either LaserDisapproved or LaserApproved, depending on the state of the supervisor at time t_0; and the variables should be initialized to the actual values at t_0.

With the above hybrid automata model of the laser tracheotomy MDPnP, we can formally express Safety Hazard 1 and 2 (see the beginning of Section 3) as follows.

Safety Hazard 1: For any given initial state σ_0, $\sigma_0 \models$ **true**$\exists\mathcal{U}_{\leq T} \bigcup_{v \in V_{comp} \wedge v|_{ls} = \text{LaserEmitting}} (v, O_2(t) \geq \Theta_{O_2})$;

Safety Hazard 2: For any given initial state σ_0, $\sigma_0 \models$ **true**$\exists\mathcal{U}_{\leq T} \bigcup_{v \in V_{comp}} (v, SpO_2(t) \leq \Theta_{SpO_2})$;
where V_{comp} is the location set of the combined automaton of the Ventilator, Patient, Laser Scalpel, and Supervisor; $v|_{ls}$ is v's projection on the Laser Scalpel automaton location set.

When model checking any one of the above safety hazards, a "yes" answer means the system is unsafe; while a "no" answer means this system is safe.

4 System Co-Design Pattern

The evolution from offline model checking to online model checking must also be matched with system design changes.

4.1 Hard Real-Time System Design

First, the overall system architecture shall integrate online model checking as a runtime fault prediction and prevention mechanism.

A straightforward thought is to run online model checking periodically. So far, we have assumed the period to be the same as the online model checking's finite-horizon T. That is, at the beginning of each period T, online model checking predicts whether unsafe states are reachable within the coming T seconds. If so, the system switches to a fall-back plan for the current period. The fall-back plan is application dependent. For laser tracheotomy MDPnP, a simple fall-back plan is that the supervisor locks $LaserApprove$ at **false**, hence forbidding laser emission and keeping the ventilator active.

The above overall architecture works if online model checking costs 0 time. In practice, this is an over simplification. However, if the online model checking has a *worst case execution time bound* $D < T$ (where T is the online model checking's finite-horizon), then we can run the online model checking as a *hard real-time* task and use pipelining to carry out fault prediction and prevention. This is formally described by the algorithm in Fig. 9, which, without loss of generality, runs a pipeline with $T = 2D$, and D replaces T to be the new sampling period.

```
//This code assumes online model checking (see line 4, 5) can always
//finish within hard real-time deadline $D = \frac{T}{2}$.
1. main(){
2.   wait till current time $t$ satisfies ($t \bmod \frac{T}{2} = 0$);
3.   $t_0 := t$;
4.   read sensors and build online model $A$;
5.   if ($A$ may reach unsafe states in $[t_0, t_0 + T]$){
6.       /*non-blocking call:*/ switch the hybrid system to fall-back plan;
7.   }else
         /*non-blocking call:*/ allow the hybrid system to run normally;
8.   goto line 2;
9. }
```

Fig. 9. Overall system architecture for *hard real-time* online model checking, with worst case execution time bound of D (for line 4, 5). Without loss of generality, the code runs a pipeline with $T = 2D$ (see line 2, 5). To "run normally" means that the hybrid system runs according to online model A's (see line 4) descriptions.

To run the hard real-time algorithm of Fig. 9, the online model checking problem must be *decidable*. That is, a time cost upper bound must exist. In the following, we show a large family of hybrid automata systems, *strongly non-zeno*

LHA systems (SNZ-LHA-Systems) [21] to be exact, satisfy the decidability requirement.

Definition 1 (SNZ-LHA-System): Let $\mathcal{S}$ be a set of *linear hybrid automata* (LHA). For each LHA $A \in \mathcal{S}$, let $\mathcal{T}_A \overset{def}{=} \{\tau | \tau$ is a trajectory (see [15] for the definition of "trajectory") of A and τ passes a transition of A twice$\}$. If $\exists \varepsilon > 0$, such that $\forall A \in \mathcal{S}, \inf_{\forall \tau \in \mathcal{T}_A}\{\delta_\tau\} \geq \varepsilon$ (where δ_τ is τ's duration; $\inf \varnothing \overset{def}{=} \infty$), then $\mathcal{S}$ is called a *strongly non-zeno LHA system* (SNZ-LHA-System).

For an SNZ-LHA-System, we have the following:

Theorem 1 (Decidability): Finite-horizon reachability model checking of an SNZ-LHA-System is decidable.

Proof: See Appendix A in the supplementary file. ■

What is more, the proof of Theorem 1 also shows a time cost upper bound for finite-horizon reachability model checking of an SNZ-LHA-system exists. In fact, interested readers can refer to [22] for a loose time cost upper bound, though a tight time cost upper bound is still an open problem.

Therefore, if we ensure an online hybrid systems model to be an SNZ-LHA-System, *real-time* worst case execution time (i.e., deadline) exists.

Given a set $\mathcal{S}$ of LHAs, we claim in the following that $\mathcal{S}$ is ensured to be an SNZ-LHA-System if it complies with certain design patterns stated in Theorem 2.

Theorem 2 (Decidable Design Pattern): If every cycle of transitions in $\mathcal{S}$ complies with one of the following design patterns: ε-Minimal Dwelling Time, ε-Alternating Cyber-Value, or ε-Alternating Physical-Value, then finite-horizon reachability model checking on the LHA set $\mathcal{S}$ is decidable.

Proof: See Appendix B in the supplementary file for detailed definitions and proof. ■

If we review the laser tracheotomy MDPnP online LHA model (see Fig. 4(b) $\sim$ 8), we find its design pattern complies with Theorem 2. Hence *online finite-horizon reachability hybrid systems model checking* (simplified as "*online model checking*" in the following, unless explicitly denoted) on laser tracheotomy MDPnP is decidable. That is, theoretically, a worst case execution time bound for hard real-time exists.

4.2 Soft Real-Time System Design

Though online *hard* real-time model checking of SNZ-LHA-Systems is theoretically possible due to Theorem 1, a tight bound on worst case execution time is still an open problem. A very loose bound is known (see [22]), but it is often too large to be practical. In fact, we know the following:

Theorem 3: Finite-horizon reachability model checking of an SNZ-LHA-System is NP-Hard.

Proof: See Appendix C in the supplementary file. ■

Theorem 3 implies online hard real-time model checking of SNZ-LHA-Systems is only practical for very small scale cases; *soft* real-time online model checking instead has more practical value.

In soft real-time online model checking, we directly specify a desired deadline D, without requiring *hard* real-time guarantee. The selection method of D is empirical: as long as D makes deadline misses satisfactorily rare and the online modeling satisfactorily accurate. For example, we can use standard benchmarks to find a desirable D (see Section 5.2).

Even though deadline D may be missed, soft real-time online model checking can still serve the MDPnP hybrid system in at least two ways: one conservative and the other aggressive, as described by the pseudo code in Fig. 10.

```
//Online model checking deadline is $D = \frac{T}{2}$ (see line 4, 6, 7, 11, 12).
1. main($mode$){
2.   wait till current time $t$ satisfies ($t$ mod $\frac{T}{2} = 0$);
3.   $t_0 := t$;
4.   read sensors and build online model $A$;
5.   if ($mode$ ="conservative way"){
6.     if (($A$ may reach unsafe states in $[t_0, t_0 + T]$)
7.       or (current time $t \geq t_0 + \frac{T}{2}$)){
8.       /*non-blocking call:*/ switch the hybrid system to fall-back plan;
9.     }else
         /*non-blocking call:*/ allow the hybrid system to run normally;
10.  else {//mode ="aggressive way"
11.    if ((not ($A$ may reach unsafe states in $[t_0, t_0 + T]$))
12.      or (current time $t \geq t_0 + \frac{T}{2}$)){
13.      /*non-blocking call:*/ allow the hybrid system to run normally;
14.    }else
         /*non-blocking call:*/ switch the hybrid system to fall-back plan;
15.  }
16.  goto line 2;
17. }
```

Fig. 10. Revised overall system architecture that allows *soft real-time* online model checking. Without loss of generality, the code runs a pipeline with $T = 2D$ (see line 2, 6, 11), where $D = \frac{T}{2}$ is the real-time online model checking deadline. To "run normally" means that the hybrid system runs according to online model A's (see line 4) descriptions.

In the conservative way, if online model checking misses deadline D, the MDPnP hybrid system always switches to the (application dependent) fall-back plan. Assuming the modeling is accurate, the conservative way can prevent all accidents. However, if deadline misses are too often, the system will frequently switch to fall-back plan, annoying the users. In other words, the conservative way can raise a lots of false alarms, but can prevent all accidents.

Take our laser tracheotomy MDPnP for example. Every time the online model checking misses the D seconds deadline on safety check, the supervisor will disapprove any laser emission request for the next D seconds (i.e., the "fall-back plan"). Instead, only when the online model checking confirms safety within the D seconds deadline will the supervisor follow Fig. 8's descriptions in the next D seconds.

In the aggressive way, if online model checking misses deadline D, the MDPnP system does not switch to fall-back plan. The aggressive way only invokes fall-back plan when it is certain the system is facing risks. In other words, the aim of aggressive way is not to prevent all accidents, but to *reduce* accidents. In medical practice, a method that can significantly reduce accidents is still a useful method; in fact, most medical routines are of such nature [23].

Again take our laser tracheotomy MDPnP for example. Every time the online model checking misses the D seconds deadline on safety check, the supervisor will nevertheless follow Fig. 8's descriptions in the next D seconds. The fall-back plan (that the supervisor disapproves any laser emission requests) only kicks in when online model checking is certain that unsafe state is reachable within the D seconds deadline. Therefore, the online model checking is not to *eliminate* all possible accidents that a human surgeon may make, but to *reduce* such accidents as an additional protection.

To summarize, each deadline miss means the online model checking is uncertain about the safety of the MDPnP hybrid system in the next D seconds. In the conservative way, the system always switches to the fall-back plan when the online model checking ends up uncertain (of course it also switches to the fall-back plan when the online model checking is certain of pending risks). In the aggressive way, the system only switches to fall-back plan when the online model checking is certain of pending risks.

5 EVALUATIONS

To validate our proposed approach, especially the effectiveness (usefulness) of soft real-time online model checking for MDPnP (the "conservative way" and the "aggressive way", see Section 4.2), we carry out evaluations using real-world trachea/blood oxygen level traces.

5.1 Effectiveness

We run soft real-time online model checking program $\mathcal{P}$ (see Fig. 10) upon emulated trachea/blood oxygen level sensors for 1200 seconds. We choose soft real-time deadline to be $D = 2$ seconds (see Section 5.2 for why). According to the soft real-time pseudo code of Fig. 10, this means every $D = \frac{T}{2} = 2$ seconds, $\mathcal{P}$ queries the emulated sensors for trachea/blood oxygen level readings, then builds and verifies an online model with finite-horizon of $T = 2D = 4$ seconds.

We have two sets of 1200-second traces for the emulated sensors. The first set of 1200-second traces comes from PhysioNet [24], a comprehensive online public database (set up by NIH, NIBIB, and NIGMS) of real-world medical traces logged by hospitals. For simplicity, we call it "PhysioNet Traces". The other set of 1200-second traces comes from our own experiments on two human subjects. Human Subject 1 (HS1) mimics the combined behavior of the supervisor, laser scalpel, and surgeon in laser tracheotomy MDPnP. As shown by Fig. 11(a), HS1 randomly swaps between holding the flag of "Laser Disapproved" and "Laser Approved". Human Subject 2 (HS2) mimics the combined behavior of the ventilator and the patient in the laser tracheotomy MDPnP. When HS1 holds the "Laser Disapproved" flag, HS2 breathes smoothly at the rate of 6 seconds per respiration-cycle. When HS1 holds the "Laser Approved" flag, HS2 first tries to exhale (to his very best) and then holds his breath until HS1 raises the "Laser Disapproved" flag again (in case HS1 holds the "Laser Approved" flag for too long, HS2 is free to abort the experiment by resuming normal breath). Meanwhile, HS2's trachea and blood oxygen level are recorded by Nonin 9843 [19]. We call the derived traces the "HKPolyU Traces".

The two emulated sensors read corresponding real-world traces (PhysioNet or HKPolyU) respectively. Based on the

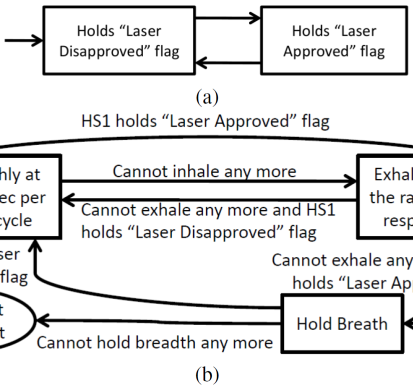

Holds "Laser Disapproved" flag
Holds "Laser Approved" flag
(a)
HS1 holds "Laser Approved" flag
Inhale smoothly at the rate of 6 sec per respiration-cycle
Cannot inhale any more
Exhale smoothly at the rate of 6 sec per respiration-cycle
Cannot exhale any more and HS1 holds "Laser Disapproved" flag
HS1 holds "Laser Disapproved" flag
Cannot exhale any more and HS1 holds "Laser Approved" flag
Experiment stop/restart
Cannot hold breadth any more
Hold Breath
(b)

Fig. 11. Human subjects roles and behaviors. (a) HS1; (b) HS2.

readings, $\mathcal{P}$ builds online hybrid systems models as described in Section 3.2, and verifies it. The specific modeling and verification software used is PHAVer [18], a well-known hybrid systems model checking tool. Our computation platform is a Lenovo Thinkpad X201 with Intel Core i5 and 2.9G memory; the OS is 32-bit Ubuntu 10.10.

For each trace, throughout its 1200-second emulation period, program $\mathcal{P}$ carries out $1200/D = 1200/2 = 600$ trials of online modeling and verifications. The statistics of execution time cost is depicted by Table 1.

The statistics show that more than 97.8% of the online model checking trials finished within the $D = 2$ (sec) deadline. In other words, only no more than 2.2% of the online model checking trials missed deadline.

Assume the modeling is accurate (which is going to be validated soon), in case $\mathcal{P}$ runs the "conservative way" (see Fig. 10), the above result means not only all accidents are prevented, the false alarm probability is no more than 2.2%. In case $\mathcal{P}$ runs the "aggressive way", the above result means more than 97.8% of accidents can be reduced (every time the system can reach unsafe states in the next D seconds, there is a $\geq 97.8\%$ chance that online model checking finishes within deadline, hence triggering the fall-back plan). Such reduction of accidents is significant according to the standards of medical practice [23]. In either case, the results provide strong evidence that (soft) real-time online model checking is effective (i.e., feasible and useful).

TABLE 1
Statistics of execution time cost of online model checking (unit: second; deadline $D = 2$ seconds)

	% of trials missed deadline	Execution time of those caught deadline (secs)			
		Min	Max	Mean	Std
PhysioNet Trace	2.2%	0.817	1.720	0.932	0.126
HKPolyU Trace	1.7%	0.818	1.940	0.965	0.146

To validate the assumption that the online modeling is accurate, we carry out statistics on the prediction error of blood oxygen level curve.

During the online model checking, at every time instance $t_0 = kD$ ($k \in \{0, 1, \ldots, 599\}$, and $D = 2$ seconds), we sample the blood oxygen level and predict (see Fig. 4) the blood oxygen level curve in $[t_0, t_0+T]$ ($T = 2D = 4$ seconds). Let the predicted blood oxygen level at time $(t_0 + T)$ be $\widetilde{SpO}_2(t_0 + T)$. Let the PhysioNet/HKPolyU trace reading of blood oxygen level at time $(t_0 + T)$ be $\widehat{SpO}_2(t_0 + T)$. We define the relative prediction error at time $(t_0 + T)$ to be

$$ERR_{SpO_2}(t_0 + T) = \frac{|\widehat{SpO}_2(t_0 + T) - \widetilde{SpO}_2(t_0 + T)|}{\widehat{SpO}_2(t_0 + T)}.$$

The statistics of the relative prediction errors throughout the 600 trials for each trace are depicted by Table 2. The statistics show that our online model checking's predictions on the finite-horizon behavior of blood oxygen level curve match the real-world traces quite accurately (with maximum relative error of 3.92%).

TABLE 2
Statistics of blood oxygen level online modeling relative errors (%)

	Min	Max	Mean	Std
PhysioNet Trace	0.03	2.53	0.51	0.52
HKPolyU Trace	< 0.01	3.92	0.61	0.60

5.2 Selection of Soft Real-Time Online Model Checking Deadline

Now we show why $D = 2$ seconds is an empirically desirable soft real-time online model checking deadline for the pseudo code of Fig. 10.

We use both the 1200-second PhysioNet Trace and the 1200-second HKPolyU Trace as benchmark, and try out different values of D.

Table 3 shows the statistics on online modeling relative errors under different Ds. The statistics show that $D = 2$ seconds incurs least maximum relative error compared to other candidates. Note $D = 2$ seconds might not be the optimal choice, but based on the evaluations on the 2400-second medical traces, it turns out to be an empirically effective choice. A lot of parameters used in medicine are derived from such empirical studies.

TABLE 3
Online Model Checking Relative Error Statistics under Different Ds

Trace	D(sec)	Relative Error (%)			
		Min	Max	Mean	Std
PhysioNet	2	0.03	2.53	0.51	0.52
	3	0.04	4.52	0.76	0.74
	4	< 0.01	5.98	0.96	0.94
HKPolyU	2	< 0.01	3.92	0.61	0.60
	3	< 0.01	4.81	0.90	0.90
	4	< 0.01	6.29	1.18	1.12

6 Discussions

6.1 False Negatives and False Positives

If the online model is absolutely accurate, the online model checking either misses deadline, or produces true-positive/true-negative conclusions.

Interestingly, even if the online model is inaccurate, i.e., if the online model checking *can* produce false-positive/false-negative conclusions, our proposed method can still be useful for medical practices. Please see Appendix D in the supplementary file for details.

6.2 Wireless Communications Links

So far, we have assumed reliable communications links between entities. Though this assumption is empirically valid for wired communications links, it is not for wireless.

How to adopt unreliable wireless communications links in life/safety critical medical settings is a nontrivial and active research area [25][26][27][28]. A comprehensive solution is beyond the scope of this paper. However, we can still provide a simple hybrid solution to allow wireless links between the sensors and the supervisor. Our solution is as follows.

According to the pseudo code of Fig. 10, every D seconds, the sensors are supposed to update the supervisor with the new readings of the patient's vital sign(s). Suppose at time instance iD ($i \in \mathbb{Z}_{\geq 0}$), the corresponding reading is X_i. Suppose at time instance iD, the supervisor needs to look at X_{i-k}, X_{i-k+1}, ..., X_i to build the online model. If any reading(s) of $X_{i-k} \sim X_i$ is(are) lost due to wireless communications failures, then for the period of $[iD, (i + 1)D]$, the supervisor shall refuse to carry out online model checking, to cause a deliberate "deadline miss". This deliberately created deadline miss shall then be treated as a usual deadline miss.

In this way, any wireless communications failures will only result in more deadline misses. The designs and analysis described in the previous sections (and subsections) still sustain.

For further evaluations of this wireless approach, please refer to Appendix E of the supplementary file.

7 Related Work

Our approach is different from the well-known runtime verification [29]. Runtime verification aims to discover latent bugs of programs by logging and analyzing the programs' execution traces under varied inputs/configurations. It is not for *predicting/preventing* faults *before* they ever happen; whilst our approach is. For many medical CPS systems, the cost/consequence of possible faults in test runs is high or even unbearable. This necessitates our approach of predicting and *preventing* faults before they ever happen.

Sen et al. [30] propose an online safety analysis method for multithreaded programs. However, this work only focuses on how to infer other potential executions that can take place in the *past*. Our work tries to predict the *future* state of patient based on recent observations

Easwaran et al. [31], Qi et al. [32], and Harel et al. [33] also propose bringing model checking online. But they are still focusing on discrete (automata) model checking, rather than hybrid systems model checking that this paper is about.

Sauter et al. [34] propose a lightweight hybrid-system model checking method, which uses ordinary differential equations (ODE) to predict temporal logic properties However, in the MDPnP systems it is not uncommon to be lack of differential equations governing patients dynamics, i.e., patients model.

Li et al. [35] propose one online model checking approach aiming at automatically estimating parameters in simulation models, which are often used for biological purpose to understand complex regulatory mechanisms in cell.

Larsen et al. [36] propose an online model-based testing tool for real-time systems, UPPAAL TRON. The tool is based on UPPAAL engine and models real-time systems as timed automata, whereas our online model checking of MDPnP systems focuses on more general hybrid systems.

Also, our approach is not model-checker specific, though our evaluation in this paper uses PHAVer. In fact, we are considering integrating our approach with other well-known model checkers, such as Bogor [37], CellExcite [38] etc..

8 Conclusion

Through our case study on laser tracheotomy MDPnP, we show that online model checking of short-run future behavior can effectively address the two challenges in MDPnP CPS hybrid systems model checking. By focusing on online and short-run future, many originally hard to describe/predict human body parameters become describable and predictable; and many variable parameters become fixed numerical values, which greatly simplifies verification. The online model checking can go real-time if the proposed hard/soft real-time system co-design patterns are followed. Our empirical evaluations based on real-world human subject traces show that our online model checking and co-design approach is feasible and effective.

Acknowledgments

The research project related to this paper is supported in part by Hong Kong GRF PolyU 5328/12E, PolyU 5245/09E, The Hong Kong Polytechnic Univ. A-PJ68, A-PJ80, A-PK46, A-PL82. Lei Bu is supported by the National Natural Science Foundation of China (No.91118007, No.61100036) , the National 863 High-Tech Programm of China (No.2011AA010103, No.2012AA011205) and the Jiangsu Province Research Foundation (BK2011558). Xue Liu is supported in part by NSERC Discovery Grant 341823-07, and FQRNT grant 2010-NC-131844. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of sponsors.

References

[1] *Medical Device Plug-and-Play (MDPnP)*. http://www.mdpnp.org.
[2] L. Sha *et al.*, "Cyber-physical systems: A new frontier," *Machine Learning in Cyber Trust: Security, Privacy, and Reliability*, 2009.
[3] C. Baier *et al.*, *Principles of Model Checking*. MIT Press, 2008.
[4] P. Tabuada, *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer, 2009.
[5] P. J. Antsaklis *et al.*, "Hybrid systems: Review and recent progress," *Software-Enabled Control: Information Technology for Dynamical Systems*, pp. 273–298, 2003.
[6] I. Lee and O. Sokolsky, "Medical cyber physical systems," *DAC*, 2010.
[7] C. Kim *et al.*, "A framework for the safe interoperability of medical devices in the presence of network failures," *ICCPS'10*, Apr. 2010.
[8] J. A. Grass, "Patient-controlled analgesia," *Anesthesia & Analgesia*, vol. 101, no. 5S, pp. S44–S61, Nov. 2005.
[9] J. X. Mazoit *et al.*, "Morphine in postoperative patients: Pharmacokinetics and pharmacodynamics of metabolites," *Anesthesia and Analgesia*, vol. 105, no. 1, pp. 70–78, 2007.
[10] "Medical devices and medical systems - essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ice), part 1: General requirements and conceptual model," no. STAM F2761-2009, 2009.
[11] T. Li *et al.*, "From offline toward real-time: A hybrid systems model checking and cps co-design approach for medical device plug-and-play (mdpnp)," *Proc. of IEEE/ACM 3rd Intl. Conf. on Cyber-Physical Systems (ICCPS)*, pp. 13–22, Apr. 2012.
[12] ——, "From offline long-run to online short-run: Exploring a new approach of hybrid systems model checking for mdpnp," *Joint Workshop on HCMDSS/MDPnP*, Apr. 2011.
[13] L. Bu *et al.*, "Toward online hybrid systems model checking of cyber-physical systems time-bounded short-run behavior," *ICCPS'11 Work-in-Progress Session*, Apr. 2011.
[14] R. Alur *et al.*, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," *Hybrid Systems*, vol. 736, pp. 209–229, 1992.
[15] ——, "Automatic symbolic verification of embedded systems," *IEEE Trans. on Software Engineering*, vol. 22, no. 3, pp. 181–201, Mar. 1996.
[16] T. A. Henzinger *et al.*, "Hytech: a model checker for hybrid systems," *STTT*, vol. 1, no. 1-2, pp. 110–122, 1997.
[17] J. A. Dorsch *et al.*, *Understanding Anesthesia Equipment, 5th ed.* Lippincott Williams and Wilkins, 2007.
[18] G. Frehse, "PHAVer: Algorithmic Verification of Hybrid Systems past HyTech," *Proc. of HSCC'05*, vol. LNCS 2289, pp. 258–273, 2005.
[19] *Nonin 9843 oximeter/Co2 detector*. http://www.nonin.com.
[20] M. J. Moran *et al.*, *Fundamentals of Engineering Thermodynamics*. Wiley, 2003.
[21] O. Roux and V. Rusu, "Uniformity for the decidability of hybrid automata," *Static Analysis*, vol. 1145, pp. 301–316, 1996.
[22] T. Li *et al.*, *From Offline toward Real-Time: A Hybrid Systems Model Checking and CPS Co-Design Approach for Medical Device Plug-and-Play (Technical Report Appendices)*. http://www.comp.polyu.edu.hk/%7Ecsqwang/research/appendix.html.
[23] F. Brunicardi *et al.*, *Principles of Surgery*. McGraw-Hill, Sep. 2009.
[24] *PhysioNet: the Research Resource for Complex Physiologic Signals*. http://www.physionet.org.
[25] Y. Wang *et al.*, "WiCop: Engineering WiFi temporal white-spaces for safe operations of wireless body area networks in medical applications," *Proc. of RTSS'11*, pp. 170–179, 2011.
[26] J. Huang *et al.*, "Beyond co-existence: Exploiting WiFi white space for ZigBee performance assurance," *Proc. of ICNP'10*, pp. 305–314, Oct. 2010.
[27] Q. Wang *et al.*, "Building robust wireless LAN for industrial control with the DSSS-CDMA cell phone network paradigm," *IEEE Trans. on Mobile Computing*, vol. 6, no. 6, pp. 706–719, Jun. 2007.
[28] S. Baker *et al.*, "Medical-grade, mission-critical wireless networks," *IEEE EMB Magazine*, vol. 27, no. 2, pp. 86–95, 2008.
[29] B. Finkbeiner *et al.*, "Collecting statistics over runtime executions," *ENTCS*, vol. 70:4, 2002.
[30] K. Sen *et al.*, "Online efficient predictive safety analysis of multithreaded programs," in *Proc. of TACAS'04*, 2004, pp. 123–138.
[31] A. Easwaran *et al.*, "Steering of discrete event systems: Control theory approach," *Workshop on Runtime Verif.*, 2006.
[32] Z. Qi *et al.*, "A hybrid model checking and runtime monitoring method for c++ web services," *Intnl' Joint Conf. on INC, IMS and IDC*, 2009.
[33] D. Harel *et al.*, "Smart play-out of behavioral requirements," in *Proc. of FMCAD '02*, London, UK, 2002, pp. 378–398.
[34] G. Sauterand *et al.*, "Lightweight hybrid model checking facilitating online prediction of temporal properties," in *Proc. of the 21st Nordic Workshop on Programming Theory, NWPT 09*, 2009, pp. 20–22.
[35] C. Li *et al.*, "Online model checking approach based parameter estimation to a neuronal fate decision simulation model in caenorhabditis elegans with hybrid functional petri net with extension," *Mol. BioSyst*, pp. 1576–1592, 2011.
[36] K. G. Larsen, M. Mikucionis, and B. Nielsen, "Uppaal tron user manual," 2007.
[37] Robby *et al.*, "Bogor: An extensible and highly-modular software model checking framework," *Proc. of ESEC/FSE-11*, 2003.
[38] E. Bartocci *et al.*, "Cellexcite: An efficient simulation environment for excitable cells," *BMC Bioinformatics*, vol. 9, no. 2, pp. 1–13, Mar. 2008.

Tao Li Tao Li received the BE degree in software engineering from Southeast University, Nanjing, China, in 2007, the MSc degree in computer science and technology from Nanjing University, Nanjing, China, in 2010. He is currently pursuing the Ph.D. degree in the Department of Computing at the Hong Kong Polytechnic University, Hong Kong. His research interests include cyber-physical systems (CPS), pervasive/ubiquitous computing, and their applications in medical treatment and healthcare. He is a student member of the IEEE.

Feng Tan Feng Tan received his B.E and M.E degree in Industrial Engineering from Univ. of Electronic Science and Technology of China, Chengdu, China in 2009 and 2012 respectively. He is currently pursuing the Ph.D. degree in the department of Computing at the Hong Kong Polytechnic University, Hong Kong. His research interests include Cyber-Physical Systems (CPS), particularly on dependable system architecture design in CPS applications. He is a student member of the IEEE.

Qixin Wang (M'08) received the B.E. and M.E. degrees from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 1999 and 2001, respectively, and the Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana-Champaign (UIUC) in 2008. He was a Research Assistant/Associate in UIUC from 2001 to 2008. Since 2009, he is an Assistant Professor in the Department of Computing at the Hong Kong Polytechnic University. His research interests include cyber-physical systems, real-time/embedded systems and networking, wireless technology, and their applications in industrial control, medicine, and assisted living. He has published about 30 research papers on various venues. He has received a best paper award from the IEEE Transactions on Industrial on Informatics (2008). Dr. Wang is also a member of the ACM.

Lei Bu (M'12) is an assistant professor in the Department of Computer Science and Technology, State Key Laboratory for Novel Software Technology at Nanjing University, P.R.China. He received his B.S. and PH.D. degree in Computer Science from Nanjing University in 2004 and 2010 respectively. He has been visiting scholar in Carnegie Mellon University, Fondazione Bruno Kessler, and University of Texas at Dallas. His main research interests include formal method, model checking, especially verification of hybrid system and cyber-physical system. He has published more than 30 research papers in major peer-reviewed international journals and conference proceedings. Dr. Bu is also a member of the ACM.

Jian-nong Cao Dr. Cao is currently a chair professor and head of the Department of Computing at Hong Kong Polytechnic University. His research interests include parallel and distributed computing, computer networks, mobile and pervasive computing, fault tolerance, and middleware. He has co-authored 4 books, co-edited 9 books, and published over 300 papers in major international journals and conference proceedings. He has directed and participated in numerous research and development projects and, as a principal investigator, obtained over HK$25 million grants. Dr. Cao is a senior member of China Computer Federation, a senior member of IEEE, and a member of ACM. He is the Chair of the Technical Committee on Distributed Computing of IEEE Computer Society. Dr. Cao has served as an associate editor and a member of the editorial boards of many international journals, and a chair and member of organizing / program committees for many international conferences. Dr. Cao received the BSc degree in computer science from Nanjing University, Nanjing, China, and the MSc and the Ph.D degrees in computer science from Washington State University, Pullman, WA, USA.

Xue Liu is an associate professor in the School of Computer Science at McGill University. He received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 2006. He received his B.S. degree in Mathematics and M.S. degree in Automatic Control both from Tsinghua University, Beijing, China. He has also worked as the Samuel R. Thompson Associate Professor in the University of Nebraska-Lincoln and HP Labs in Palo Alto, California. His research interests are in computer networks and communications, smart grid, real-time and embedded systems, cyber-physical systems, data centers, and software reliability. Dr. Liu has published more than 150 research papers in major peer-reviewed international journals and conference proceedings, including the Year 2008 Best Paper Award from IEEE Transactions on Industrial Informatics, and the First Place Best Paper Award of the ACM Conference on Wireless Network Security (WiSec 2011).