Toward Patient Safety in Closed-Loop Medical Device Systems *

David Arney
Computer & Information Science
University of Pennsylvania
arney@seas.upenn.edu

Miroslav Pajic
Electrical & System Engineering
University of Pennsylvania
pajic@seas.upenn.edu

Julian M. Goldman
MD PnP Program
Massachusetts General Hospital & CIMIT
jmgoldman@partners.org

Insup Lee
Computer & Information Science
University of Pennsylvania
lee@cis.upenn.edu

Rahul Mangharam
Electrical & System Engineering
University of Pennsylvania
rahulm@seas.upenn.edu

Oleg Sokolsky
Computer & Information Science
University of Pennsylvania
sokolsky@cis.upenn.edu

ABSTRACT

A model-driven design and validation of closed-loop medical device systems is presented. Currently, few if any medical systems on the market support closed-loop control of interconnected medical devices, and mechanisms for regulatory approval of such systems are lacking. We present a system implementing a clinical scenario where closed-loop control may reduce the possibility of human error and improve safety of the patient. The safety of the system is studied with a simple controller proposed in the literature. We demonstrate that, under certain failure conditions, safety of the patient is not guaranteed. Finally, a more complex controller is described and ensures safety even when failures are possible. This investigation is an early attempt to introduce automatic control in clinical scenarios and to delineate a methodology to validate such patient-in-the-loop systems for safe and correct operation.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; D.2.4 [**Software/Program Verification**]: Formal methods; D.4.5 [**Reliability**]: Fault tolerance; H.3.4 [**Systems and Software**]: Distributed systems; J.3 [**Life and Medical Sciences**]: Medical information systems

1. INTRODUCTION

Clinical scenarios for critical care patients often involve large numbers of medical devices. Some of these devices, such as bedside monitors, provide vital information about the state of the patient. Other devices, for example, infusion pumps, provide treatment. That is, they affect the state of the patient, for example, by infusing medication. Medical device systems, considered together with the patient and caregivers, represent an important class of cyber-physical systems. Patient safety is the primary concern in such systems, yet reasoning about patient safety is very difficult because of insufficient understanding of the dynamics of human body response to treatment. Human errors, another important source of patient safety problems, are also difficult to reason about in the framework of conventional embedded system development.

It is natural to view a clinical scenario as a control system, in which the patient is a plant, bedside monitors are sensors and infusion pumps are actuators. Traditionally, caregivers perform the role of the controller in such a system. This means that the caregiver needs to continuously monitor all sensor devices and apply appropriate treatment. The large number of devices to monitor and control makes the job of the caregiver very difficult. On top of that, a caregiver is typically responsible for several patients. An emergency may divert the caregiver's attention elsewhere, making him or her miss an important event. As a result, patient safety may suffer. Multiple such occurrences are documented in the clinical literature.

The human caregiver will always play an indispensable part of most clinical scenarios. However, in many cases automatic controllers can play important roles, reducing the burden on the caregiver and avoiding the possibility of human errors. A number of such cases have been documented in the ASTM standard for the Integrated Clinical Environment [1]. The standard has been developed by the Medical Device Plug-and-Play Interoperability program at the Center for Integration of Medicine & Innovative Technology of the Massachusetts General Hospital (mdpnp.org) [2]. Although many medical devices today have network interfaces and can send sensed data across the network, few can be controlled remotely. Vendors of medical equipment continue to stay away from closed-loop scenarios.

The rationale is that it is difficult to reason about patient safety in closed-loop scenarios. Such reasoning is necessary for medical devices, which have to be approved for use by

*Research is supported in part by the National Science Foundation grants CNS-0834524 and CNS-0930647. POC: Insup Lee, lee@cis.upenn.edu

government regulators who assess their safety and effectiveness. A particular challenge arises from the complexity of a complex interplay between the continuous dynamics of the patient reaction to treatment and the discrete nature of the controller and communication network. The dynamics of the patient body is not well understood and exhibits parametric uncertainty and high variability between different patients.

To overcome this difficulty, we explore a model-driven approach that allows us to prove safety properties of devices on the modeling level and ensure that abstract models used in the verification process are sound with respect to the actual dynamics of the system.

In order to prove properties of the system, we model it using the formalism of *timed automata* [3] in the UPPAAL tool [4]. The model abstracts away continuous dynamics of the system, replacing it with timing constraints. The values for the timing constraints are obtained from the continuous dynamics of system components using the detailed model in Matlab.

Both the abstract, formal model and the detailed, informal model are needed in the process of verification, validation, and regulatory approval of closed-loop medical device systems. On the one hand, formal models allow us to exhaustively explore the possible behaviors of the system and prove its safety. On the other hand, detailed models allow us to use high-fidelity simulation that take real system dynamics into account. Both kinds of results can be used to make the case for regulatory approval.

The main contribution of the paper is the methodology for the analysis of safety properties of closed-loop medical device systems. While we do not solve the problem in its entirety, we demonstrate, using a case study, how the methodology can be applied to a system of clinical importance. In the case study, we prove that the system is safe under a set of assumptions. We then demonstrate that a violation of these assumptions – for example, by a more realistic fault model – can make the system unsafe. Finally, we propose a solution to restore the safety of the system under the new fault model.

The rest of the paper is organized as follows. Section 2 introduces the medical case study and presents and the system architecture of the case study that we used to describe our approach. Section 3 describes the UPPAAL models of the system and safety properties we have verified. Section 4 explains the use of Matlab models of the system to analyze safety with no failure assumption. It then explains modifications to the system to deal with failures and provides an argument that the new system is fail-safe. The last section summarizes the paper and identifies the future work.

2. CASE STUDY

In this section, we describe a case study that represents one of the MD PnP interoperability clinical scenarios [1]. In the past, we have built a demonstration of this scenario [5]. Several variants of this system have been presented at the American Society of Anesthesiologists annual meeting in 2007, where it won first place in the scientific exhibits, at the 2008 HIMMS (Healthcare Information and Management Systems Society) Congress, and at the 2008 CIMIT Innovation Congress.

We have built the system according to the Integrated Clinical Environment (ICE) architecture, developed by the MD PnP project. Figure 1 shows the main components of an ICE-compliant system. The patient and caregiver are the human elements of the system. The Supervisor is the computer system that runs the control algorithm. Medical devices are connected, through adapters where necessary, to the Network Controller, which keeps track of connected devices and their capabilities. The Data Logger records pertinent network traffic for later forensic analysis and external networks such as the hospital information system are connected through an external interface. In our case study, we did not model the data logger since it does not affect our runtime safety analysis.

2.1 Clinical Use Case

The selected scenario involves a patient connected to a patient-controlled analgesia (PCA) infusion pump. PCA infusion pumps are commonly used to deliver opioids for pain management, for instance after surgery. Patients have very different reactions to the medications and require very different dosages and delivery schedules. PCA pumps give the patient a button to press to request a dose when they decide they want it rather than using a schedule fixed by a caregiver. Some patients may decide they prefer a higher level of pain to the nausea the drugs may cause and can press the button less often, while patients who need a higher dose can press it more often.

A major problem with opioid medications in general is that an excessive dose, or overdose, can cause respiratory failure. A properly programmed PCA system should not allow an overdose because it is programmed with limits on how many doses it will deliver, regardless of how often the button is pushed. However, this safety mechanism is not sufficient to protect all patients. Some patients still receive overdoses if the pump is misprogrammed, if the pump programmer overestimates the maximum dose a patient can receive, if the wrong concentration of drug is loaded into the pump, or if someone other than the patient presses the button (PCA-by-proxy), among other causes. PCA infusion pumps are currently involved in a large number of adverse events, and existing safeguards such as drug libraries and programmable limits are not adequate to address all the scenarios seen in clinical practice [6].

The system we are considering aims to improve patient safety in such a scenario by introducing a supervisor that monitors patient data for the early signs of respiratory failure and can stop the infusion and sound an alarm if the

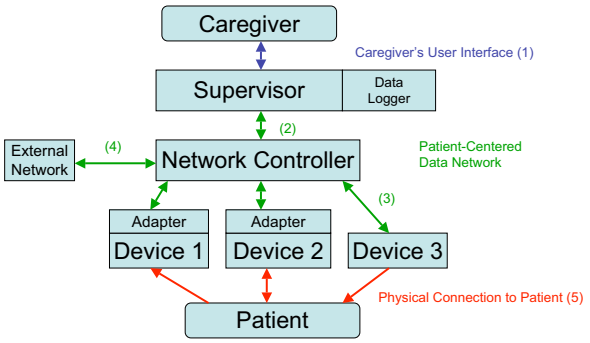

Figure 1: MD PnP Architecture with patient-in-the-loop automatic control

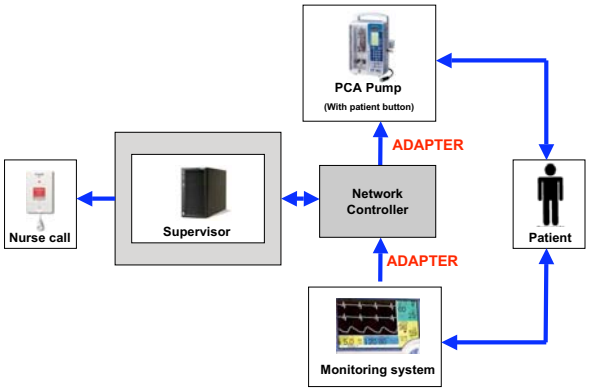

Figure 2: Hardware for PCA Demo System

patient experiences an adverse event. We use a pulse oximeter device that receives physiological signals from a clip on the patient's finger and processes them to calculate heart rate and SpO_2 outputs, where SpO_2 is the measure of blood oxygenation. Note that, at the time of writing, there are no PCA pumps on the market that are capable of being remotely controlled. In the demonstration system, we used the PCA pump prototype we have built in the past during the Generic Infusion Pump project [7].

2.2 System Architecture

Figure 2 shows the components of the PCA safety system, and Figure 3 is a photograph of a demo system built using these components. The system components are described below.

Figure 4 shows the devices and essential data flow in this control loop. The variables in the system are listed in Table 1. The pulse oximeter receives physiological signals from the patient and processes them to produce heart rate and SpO_2 outputs. The Supervisor gets these outputs and makes a control decision, possibly sending a stop signal to the PCA Pump. The PCA pump delivers a drug to the patient at its programmed rate unless it is stopped by the Supervisor. The patient model gets the drug rate as an input and calculates the level of drug in the patient's body. This in turn influences the physiological output signals through a drug absorption function.

Figure 3: PCA Demo System

source	description	name
pulse oximeter	signal processing time output HR and SpO_2 values	t_{po} hr, SpO_2
patient model	drug level drug absorption function output physiological signals	dl $f(dl)$ wf_1, wf_2
supervisor	algorithm processing time	t_{sup}
pca pump	pump stop delay infusion rate	t_{stop} rate

Table 1: Variables for critical timing loop

PCA Infusion Pump. Patients using a PCA pump are usually also attached to patient monitors that record the patient's EKG, blood pressure, respiratory rate, and SpO_2. These monitors sound alarms if the values they measure are outside thresholds set by the caregivers, but they do not stop the infusion. Thus, the patients continue to receive more of an overdose while the caregiver responds, assesses the patient, decides whether there is a real problem, and finally stops the pump.

The pump in our case study operates in the following way. Before operation, the pump is programmed by the caregiver, who sets the normal rate of infusion, the increased rate of a bolus, and bolus duration. Some PCA pumps also can be programmed to limit the total amount of drug to be infused. Once programmed and started, the pump delivers the drug at the normal rate until it is stopped or the bolus button is pressed. From that moment, it delivers drug at the bolus rate for the specified duration and then returns to the normal rate.

The pump is equipped with a number of built-in sensors that detect internal malfunctions such as the presence of air in the tubes that deliver the drug. When a problem is detected, the pump is stopped. We do not consider such malfunctions in this case study and do not represent the built-in alarm mechanism.

Finally, the pump is equipped with a network interface, which allows the pump to transmit its status across the network to other devices such as the logger. For the purpose of our scenario, we assume that the network interface allows the pump to accept control signals. A *stop* control signal will set the current infusion rate to zero, while the *start* signal will set the normal infusion rate (regardless of the state of the pump before it was stopped).

Pulse Oximeter. In this study, we look at using SpO_2 and heart rate measurements as the basis for a physiologic closed-loop control system that can stop the PCA pump and halt the dose of opioid while sounding an alarm if respiratory distress is detected. Both of these measurements can be produced by a device called pulse oximeter. This device is equipped with a finger clip sensor that shines two wavelenghts of light through the patient's finger. The measured light intensity reflects the blood oxygen content, which can change rapidly.

last output value	new window size
97 - 100	10
94 - 96	8
90 - 93	7
85 - 89	6
< 85	4

Table 2: Sliding Window Size for Pulse Oximeter

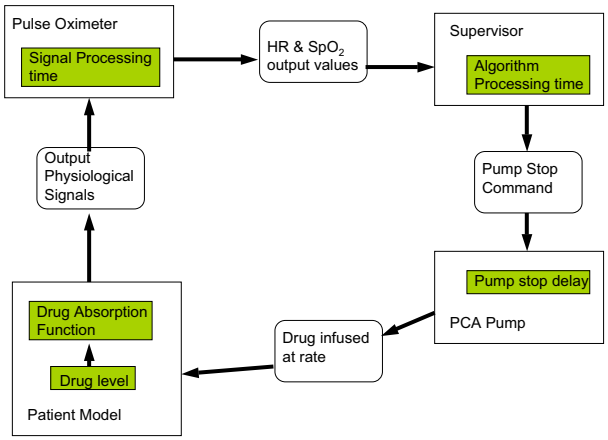

Figure 4: PCA System Control Loop

The pulse oximeter samples the patient's SpO_2 at regular intervals, processes them, and outputs an averaged result [8]. It calculates the average using a variable-sized sliding window. The window size varies with the last output value. The reason for changing the window size is that smaller sample size gives faster, but potentially less accurate results. When SpO_2 values are low, quick response is more important than filtering out transient noise. When SpO_2 is high, increasing the window size helps to filter out transient low values at the expense of less frequent updates. Since the samples are at regular intervals and a varying number of samples are used to calculate the output, the output is updated irregularly. The size of the sliding window that we used in the case study is determined using a simple table shown in Table 2.2. Note that this table does not reflect the details of any real implementation but rather attempts to capture the essential behavior of a typical pulse oximeter.

Patient Model. We use a simple patient model, where the patient state is characterized by the current drug level. The state space is partitioned into regions. The patient can be in pain (under-medicated), pain-controlled (adequate medication), or over-medicated. If the patient is over-medicated to the point that he or she starts experiencing respiratory distress, we consider it an overdose. We refer to the overdose condition as the *Critical* region. Any treatment needs to make sure that the patient stays out of the critical region, and we use this requirement as the main safety property of the system that needs to be ensured. In this case study, we defined the boundary of the *Critical* region in terms of the patient SpO_2 and heart rate and set it to $H_2 = 70\%$ for SpO_2 (and $H_2 = 11.5$ $beats/min$ for heart rate), a clear indication of respiratory failure.

Our model represents the instantaneous level of medication in the patient's body as a single variable. This variable is linked to the patient's heart rate and SpO_2 by the drug absorption function, which represents how the patient reacts to the dose received over time. Some patients react very quickly to a dose of drug, while others react more slowly. By adjusting this function, we can tune the model to different patient types.

Caregiver Model. The caregiver in this system programs the PCA pump and reacts to alarms. The control system is closed loop, so no intervention by the caregiver is necessary to stop the infusion when a problem is detected. The caregiver can react to restart the system if it has stopped in reaction to a false alarm, or when a problem such as a slipped patient sensor is fixed.

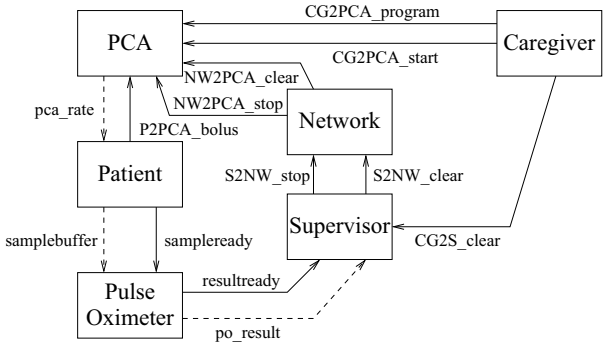

Figure 5: Communication structure of the UPPAAL model

Supervisor. The supervisor is a running program in the system that communicates with other devices and executes clinical application scripts (CAS's). A CAS is a script that implements a particular clinical use case. The clinical application in this case study is to control the loop shown in Figure 4. The Supervisor receives the patient's heart rate and SpO_2 measurements from the pulse oximeter and uses this information to decide whether the PCA infusion pump should be allowed to run or immediately stopped.

In the case study, we designed a simple control algorithm for the supervisor, in which the decision to stop the pump is made as soon as the patient heart rate or SpO_2 readings fall below a fixed threshold. The choice of threshold needs to ensure that the patient does not enter the *Critical* region despite the delay in detecting the problem and delivering the control signal to the pump. For the case study, we defined the threshold as $H_1 = 90\%$ for the SpO_2 and $H_1 = 57$ $beats/min$ for heart rate. Values below these thresholds typically indicate "a clinical concern" ([9], p. 45), meaning that a caregiver needs to be notified. The supervisor notifies the caregiver when the threshold is crossed, as it sends the message to stop the pump. Values between H_1 and H_2 are thus referred as the *Alarming* region. The width of the alarming region is denoted $\Delta H =\mid H_2 - H_1 \mid$.

3. VERIFICATION AND VALIDATION OF COMPONENTS AND SYSTEM IN UPPAAL

The structure of the UPPAAL model follows the architecture of the system. For each component in Figure 2, the model includes a separate automaton. The automata communicate using synchronization channels and shared variables. Figure 5 shows network of automata and communication between them. Solid arrows represent communication channels and dashed arrows represent shared variables.

3.1 UPPAAL Component Models

The PCA automaton, which represents the pump, is shown in Figure 6. When the pump is operational, it is either in the state `running`, with the shared variable `pca_rate` set to default rate, or in the state `bolusing`, when `pca_rate` is increased by the bolus rate. Both rates are specified as pa-

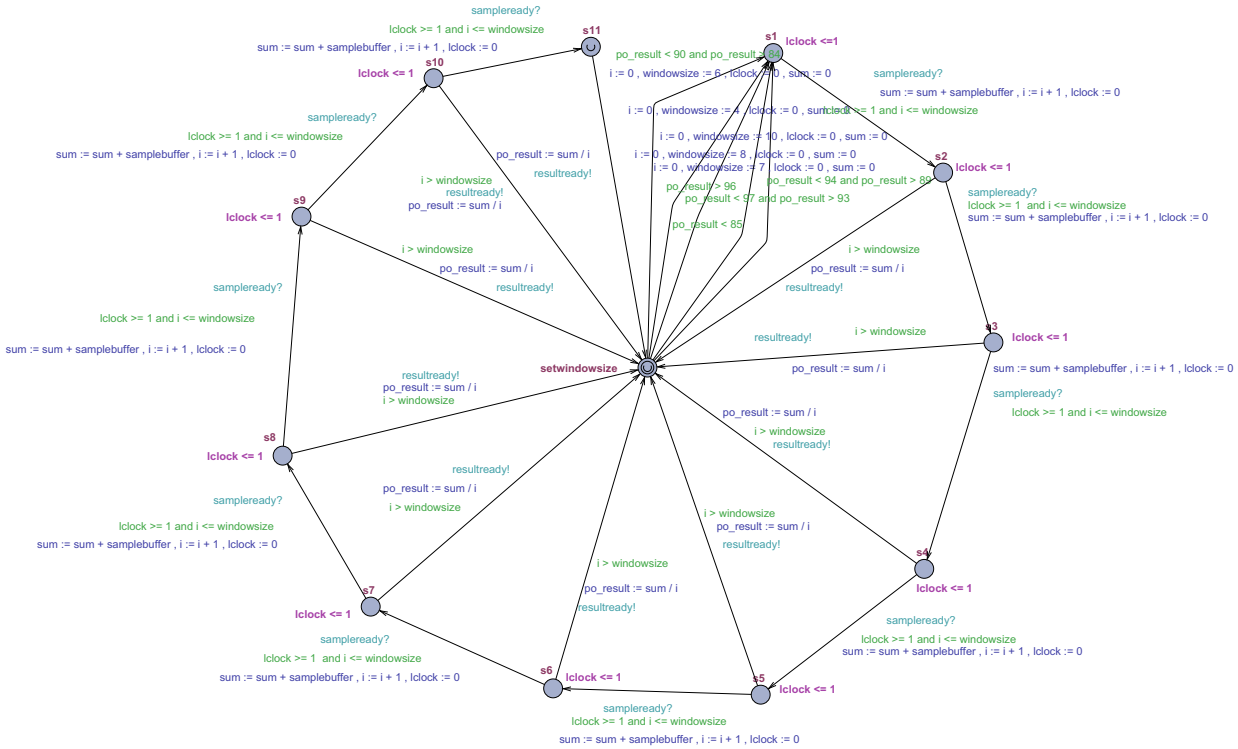

Figure 7: Timed automaton for the pulse oximeter

rameters of the model. The pump can be bolusing for a fixed duration given by the value of the `bolus_time` parameter. The pump transitions to the `bolusing` state upon the signal received from the patient only if it is in the `running` state; in all other states, the signal is ignored. From either `running` or `bolusing` state, the pump can move to a stopped state (`Rstopped` or `Bstopped`, respectively) upon a signal from the network.

The PO automaton, which represents the pulse oximeter, is shown in Figure 7. The operation of the automaton proceeds in rounds. Each round begins by setting the window size for the round based on the last sampled value. Then, the automaton collects the number of samples to fill the window. Samples are obtained periodically with the interval of 1 time unit, which corresponds to 100 ms. Finally, the result is stored in the `po_result` variable and delivered to the supervisor using the `resultready` channel.

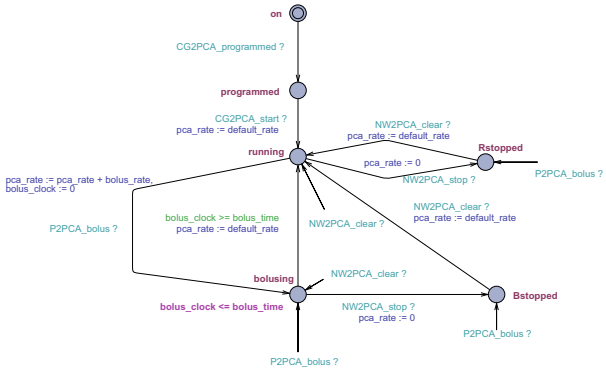

Figure 6: Timed automaton for the PCA pump

The Supervisor automaton, shown in Figure 8, implements the simple control algorithm. Upon receiving a SpO_2 reading from the pulse oximeter, the supervisor compares it with the pre-defined threshold value and, if the result is too low, sends the stop message to the pump across the network. The model also incorporates a delay, which represents the worst-case execution time of the supervisor algorithm. Then, once the caregiver resolves the problem, the supervisor sends another message to restart the pump. For simplicity of the presentation, the Supervisor automaton only deals with SpO_2, not heart rates.

The Patient automaton, shown in Figure 9, periodically updates the drug level based on the flow rate of the pump and drug absorption rate. At any time, it can deliver a sample as the function of the current drug level.

The network automaton in Figure 10 implements a two-place buffer, which means that there may be two network messages in transit. The stop message may be dropped by the network, if the boolean parameter `drop` is set to true. We do not model dropping of the restart message, since the loss of these messages does not affect the safety of the patient. If messages are not dropped, they are delivered by the network in order. The caregiver automaton, not shown here, contains one state and can send any of the messages at any time. A more detailed model of the caregiver may include data-dependent behaviors, for example, the clear signal may be sent only if the SpO_2 reading is high enough. However, any

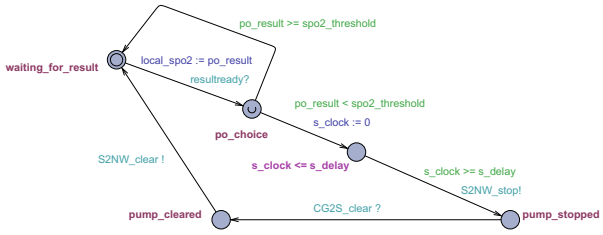

Figure 8: Timed automaton for the supervisor

other model will have fewer behaviors than the caregiver model used here, and thus the safety property will still hold.

3.2 Verifying PCA System Safety Properties

The main safety property that needs to be verified on the UPPAAL model is whether or not the patient can enter the *Critical* region, where SpO_2 and heart rate are low enough to indicate a respiratory arrest. Before verifying safety, however, we perform several auxiliary checks to ensure sanity of the model.

We express properties we verify in the subset of the Computational Tree Logic (CTL) [10] used by UPPAAL. The main temporal operators of this logic are $A\Box\phi$, which means that ϕ is satisfied in every state along every execution path from the current state, and $A\diamond\phi$, meaning that ϕ is satisfied eventually along every path.

The first sanity check is the absence of deadlocks in the model. Another sanity check is that once the SpO_2 level goes below the pain threshold, it eventually goes up. This property is captured by the temporal logic formula

$$A\Box(samplebuffer < pain\_thresh \Rightarrow A \diamond samplebuffer \geq pain\_thresh). \quad (1)$$

Note that the property is defined in terms of the true SpO_2 level as defined by the patient model, not the sensor reading obtained by the supervisor. Intuitively, this property should hold, because the normal infusion rate is lower than the drug absorption rate and, once the patient stops requesting new boluses and the last bolus infusion is over, drug level will start decreasing and thus SpO_2 and heart rate levels should increase, until they reach pain threshold again. Finally, we check that the pump is stopped if the patient ever enters the *alarming* region. Formally,

$$A\Box(samplebuffer < alarm\_thresh \Rightarrow A \diamond (PCA.Rstopped \vee PCA.Bstopped)). \quad (2)$$

We consider this property to be a sanity check rather than a safety requirement, because wrong parameters of the model

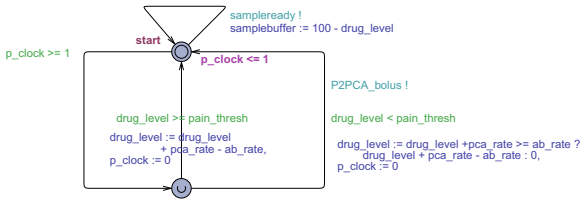

Figure 9: Timed automaton for the patient

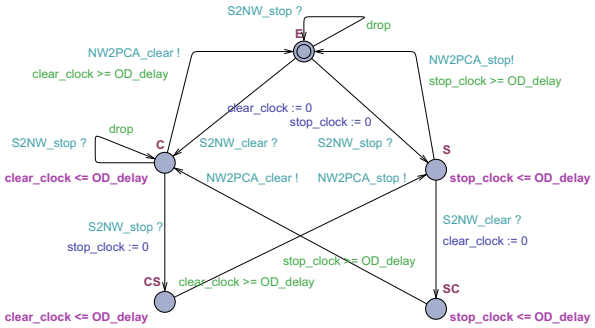

Figure 10: Timed automaton for the network

– for example, too short bolus duration or too high drug absorption rate – can make the system appear safe (that is, SpO_2 level never goes too low), but it would be safe for the wrong reason. All sanity checks were passed by the UPPAAL model described above when no dropped messages are allowed. Clearly, property (2) does not hold if messages can be dropped.

Finally, we turn to checking the main safety property. With the threshold for the *Critical* region set to 70%, the property $A\Box(samplebuffer \geq critical)$ is satisfied if the stop message cannot be dropped. However, if losing messages is enabled in the network automaton, the property is not satisfied.

4. MATLAB / SIMULINK MODELING OF SYSTEM DYNAMICS

Figure 11 presents the Simulink model of the PCA pump system. The overall structure of the Simulink model follows that of the model shown in Figure 2. The Simulink models are used to capture the dynamics of the PCA infusion pump, pulse oximeter, patient model, and supervisor, described in Section 2. They are used to define the notion of the safe, critical, alarming regions precisely. In addition, using the overall structures and timing properties of the components, this section shows when the system is safe and when the system is not safe. Also, this section describes how to cope with failures of network while maintaining the safety of the overall system.

4.1 Simulink Models

Figure 12 presents the *Patient*'s model, where effects of the drug flow are depicted using the drug absorption function. The Patient Model's dynamics are modeled as a first-order continuous system with predefined values for pole and zero, which is a simplified version of the function presented in [11]. In addition, its Heart Rate (HR) and SpO_2 level were extracted from the drug level using linear mapping. We have simulated the *Patient*'s behavior when the drug is repeatedly delivered for 10 minutes followed with 10 minutes pause. Figure 13 presents obtained changes in HR and SpO_2 levels.

PO, designed similarly as described in Section 3, monitors the *Patient*'s HR and SpO_2 level and informs the *Supervisor* about these values. Using the *Patient*'s readings we define three different regions shown in Figure 14 as follows:

- The *Safe* region is defined as the region where the *Pa-*

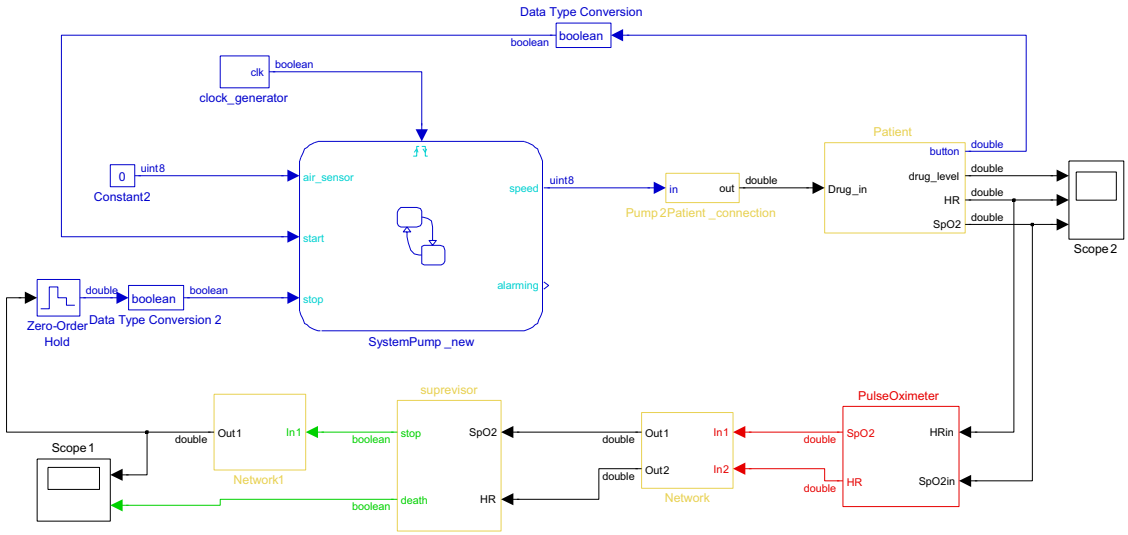

Figure 11: Modeling the system in Simulink

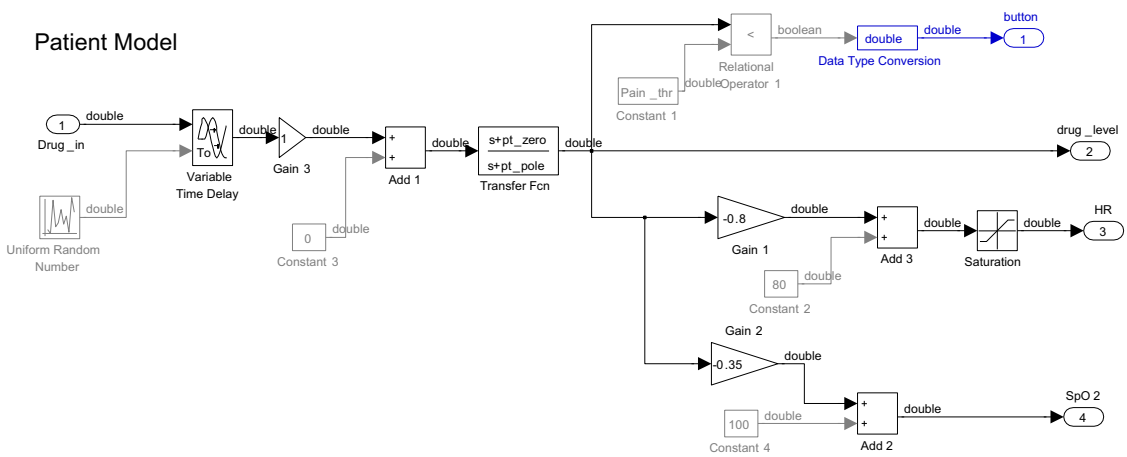

Figure 12: Patient model in Simulink

tient's readings are below some predefined threshold values that guarantee that *Patient*'s vitals are not endangered

- The *Critical* region where *Patient*'s life is in danger or there is a chance that irreparable damage can occur.
- The *Alarming* region defined as the region where *Patient*'s vitals are not endangered but there is a reasonable concern that the *Patient* can be forced to the *Critical* region.

4.2 Analysis of System Safety Properties

For the presented system we considered safety requirement that the PCA pump will always be stopped before *Patient*'s *Critical* region is reached. The system will satisfy the requirement if following condition is met:

$$t_{POdel} + t_{net} + t_{Sup} + t_{net} + t_{Pump} + t_{P2PO} + t_{pi} \leq t_{crit} \quad (3)$$

where:

- t_{POdel} - worst case delay caused by PO: it can be calculated from the PO specification, see Table 2.2,
- t_{net} - worst case network delay: the value depends on used network protocol,
- t_{Sup} - worst case delay introduced by the *Supervisor*'s procedure: it can be calculated from the *Supervisor* model,
- t_{Pump} - worst case delay introduced by the *PCA pump*: the value can be calculated from the pump model,
- t_{P2PO} - worst case latency from the moment when command is sent from the *PCA pump* until the drug starts (or stops) flowing: it depends on type of connection used,
- t_{pi} - worst case Patient Inertia, a time that elapses before drug injected in body affects the *Patient*: it can be calculated from the pharmacological drug description, and

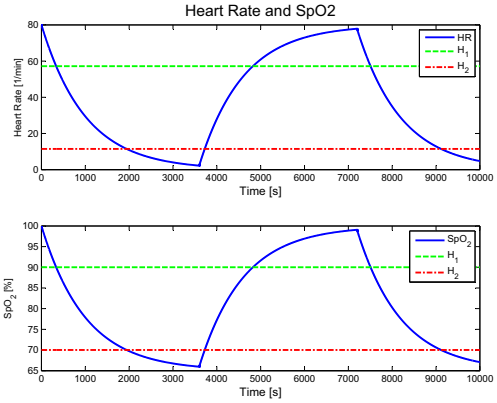

Figure 13: *Patient*'s response on the pump activities

- t_{crit} - *Patient*'s critical time, a shortest time that *Patient* spends in the *Alarming* region before it enters *Critical* region: this parameter can be calculated from the *Patient* model as described below.

If Equation 3 is satisfied we can guarantee that the *Supervisor* will be able to determine that the *Patient* have entered the *Alarming* zone and stop *PCA pump* before the *Patient* switches from *Alarming* to the *Critical* zone. The values for previously mentioned time constants are shown in Table 4.2. Here, we want to emphasize that an assumption is used that the network latency t_{net} is bounded by $0.5s$.

The main problem in determining whether the safety condition is met is to find the value for t_{crit}. If a mathematical model for the *Patient* is known this value can be analytically determined. In our model, since we modeled the *Patient* as a first order continuous system with time constant $\alpha > 0$, when the drug flow is on, HR and SpO_2 level will decrease with time as function $e^{-\alpha t}$. For example, SpO_2 level will have the form $C_{min} + Ae^{-\alpha t}$ where C_{min} and A are constants. To calculate t_{crit} consider t_1 and t_2, time instances when the *Patient* enters the *Alarming* and *Critical* regions respectively. If the *Patient* is continuously pushed toward the *Critical* region (which in this case means that the pump is continuously delivering drug to the *Patient*) then $t_{crit} = t_2 - t_1$. Since $H_i = C_{min} + Ae^{-\alpha t_i}, i \in \{1, 2\}$, t_{crit} can be calculated as

$$t_{crit} = \frac{1}{\alpha} \log \frac{H_1 - C_{min}}{H_2 - C_{min}}.$$

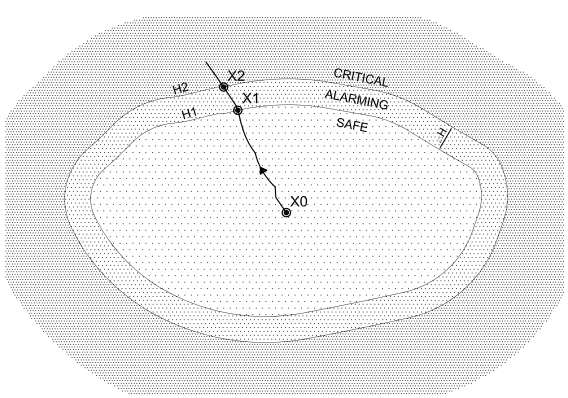

Figure 14: Regions of *Patient*'s conditions

In our model the time constant α is equal to $0.001s^{-1}$ meaning that if the drug flow is stopped the patient's drug level will be decreased to half of the initial level in approximately 11.5 minutes (690s). Thus, t_{crit} is a couple of orders of magnitude bigger than sum of all other timing parameters. For $H_1 = 90\%$ and $H_2 = 70\%$ (SpO_2 levels) and C_{min}, we have $t_{crit} \approx 1609s \approx 26.8min$. Therefore, our system always satisfies the safety requirement if the assumption that all messages are delivered is valid.

time delay	value
t_{POdel}	$1s$
t_{net}	$0.5s$
t_{Sup}	$0.2s$
t_{Pump}	$0.1s$
t_{P2PO}	$2s$
t_{pi}	$10s$

Table 3: Worst case delays

If we were given a complex patient model in Simulink, from which it is not possible to analytically determine t_{crit}, in order to obtain it the simulation of the *Patient*'s behavior can be used (see Figure 13). Of course simulation results depend on input signals and model's initial state, so we assume that worst-case scenario is known. Otherwise, approaches similar to one from [12] can be used.

An important question that needs to be addressed is the consistency between the two models. Since the UPPAAL model of the controller uses significantly simplified patient dynamics, there is the possibility that verification results obtained on the UPPAAL model would not apply to the more detailed model and thus the system itself. An important consistency check is the value of t_{crit}, the time it takes for the patient to be overdosed. In the UPPAAL model, the value of t_{crit} may be different (we denote it t^U_{crit}), but it has to be no greater than the value obtained above. In our case, t^U_{crit} turns out to be 20 seconds, an order of magnitude smaller than t_{crit}. Clearly, the UPPAAL model overestimates the rate of change for the drug level in the patient body. And, since the system has been proven safe in this case, it would also be safe in the more realistic case.

4.3 Failures and Fail-Safe PCA system

We have seen in Section 3 that the system does not satisfy its safety property if network messages can be lost. Also it can be noticed from the Equation 3 if any of the delays on the left side of the equation is significantly increased, the condition would not be satisfied. Increase in any of the delays can be caused by a component or network failures, which would result in an open-loop system. To provide safety assurance, we have to take into account realistic situations where network failures occur or *PO* accidentally gets detached from the Patient. Therefore, *Supervisor*'s control algorithm and *PCA pump*'s designs have to guarantee the system's open-loop stability. For our case study this means that even if the *Supervisor* does not receive the right values for HR and/or SpO_2 or the pump does not receive the command to disable drug flow, we have to guarantee that the patient will not enter the *Critical* region. Open-loop stability implies that even if the patient keeps pressing the button no drug flow will be enabled if there is a possibility that the amount of drug inserted can harm the patient.

One way to design the system that complies with the open-loop stability is to design the pump that receives activation

command from the *Supervisor* (not the *Patient*) along with the duration of the drug flow. When the *Patient* presses the button if the *Supervisor* is informed about current HR and SpO_2 values it can determine duration of the pump's activation (Δt_{safe}) that guarantees the patient's safety. An additionally imposed condition is that the *Supervisor* will disregard if the button is pressed during t_{del} units of time after the pump is stopped. Here, t_{del} takes into account all the delays in the loop and is defined as follows:

$$t_{del} = t_{POdel} + t_{net} + t_{Sup} + t_{net} + t_{Pump} + t_{P2PO} + t_{pi}$$

The imposed condition ensures that the last drug delivery will take full effect before the next *button pressed* command is sent to the *Supervisor*. In addition, it implies that the drug level function will reach its local maximum before HR and SpO_2 measurements sent to the *Supervisor* are obtained. All constituents of t_{del}, except the network delay, can be calculated as described in the previous Section. We assume that the underlying real-time network provides a guaranteed bound on the network delay, as described in [5].

To calculate Δt_{safe} we define following parameters:

- dl_{cur} - last received drug level
- Δdl - worst case increase in drug level due to the system latencies while the pump is off
- $\Delta dl(d, \Delta t)$ - maximal drug level increase due to drug flow of the duration Δt when the initial drug level is d
- Hdl_2 - *Critical* region threshold for the drug level

Therefore to satisfy the safety requirement, Δt_{safe} has to meet the condition

$$dl_{cur} + \Delta dl + \Delta dl(dl_{cur}, \Delta t_{safe}) \leq Hdl_2 \quad (4)$$

Since a linear dependencies between drug level and HR and SpO_2 measurements is assumed, dl_{cur} can be easily calculated from the latest measurements. Requirement that after the pump is stopped, the *Supervisor* will disregard if the button is pressed for t_{del} time units, implies that when the button pressed command is accepted $\Delta dl \leq 0$, since the drug level will reach its local maximum before the measurements are taken. Thus, the new safety condition can be obtained from:

$$\Delta dl(dl_{cur}, \Delta t_{safe}) \leq Hdl_2 - dl_{cur} \quad (5)$$

Again if mathematical model of the *Patient*'s dynamics is available, Δt_{safe} can be either analytically or numerically calculated from the previous condition. In our model, when the pump is on, the drug level function has form $dl_{max}(1 - e^{-\alpha t})$, where dl_{max} is the maximal (saturation) level for the drug absorption. Thus Δt_{safe} can be calculated as

$$\Delta t_{safe} = \frac{1}{\alpha} \log \frac{dl_{max} - dl_{cur}}{dl_{max} - Hdl_2} \quad (6)$$

Obviously if $dl_{cur} < Hdl_1$, where Hdl_1 is the drug level *Alarming* region threshold, Δt_{safe} will be greater than t_{crit}. For example in the Simulink model where $dl_{max} = 100$, $Hdl_2 = 85.71$ and $dl_{cur} = 20 < 28.57 = Hdl_1$ (which corresponds to $H_2 = 90\%$ and $H_1 = 70\%$ for SpO_2 *Critical* and *Alarming* regions thresholds respectively) we obtain $\Delta t_{safe} \approx 1723s > t_{crit}$. Similarly if the patient is already in the *Alarming* region ($dl_{cur} \geq Hdl_1$) then $\Delta t_{safe} \leq t_{crit}$.

Similarly as for safety condition described in Equation 3, more complex Simulink *Patient* model can be used for black-box testing. In this case after setting the initial condition to dl_{cur} simulation can be used to determine Δt_{safe} for each initial point dl_{cur}.

The described system can guarantee open-loop safety since, even if some (or all) of the messages are dropped, the patient would never enter the *Critical* region. The reason is that the pump activation command also contains the duration of the drug flow. If the flow duration is calculated properly, the pump will stop before safety requirement is breached.

To guarantee open-loop safety we made an assumption that the *Supervisor* is able to determine whether received measurements are valid. This is a reasonable assumption because modern *PO*s send an invalid code in cases when probes are detached from the patients body, which is the main reason why *PO*s do not obtain valid values. Note that in this work we do not consider failures where components behavior differs from the behavior described by their models. For example, if the pump is active longer then requested, or if *PO* does not send an invalid code when it is not able to obtain valid measurements.

Of course the presented solution is not the only way to guarantee open-loop safety. An alternative approach which requires minimum change to existing PCA infusion pumps is to have the *Supervisor* instruct the pump the maximum amount of drugs can be injected.

Using current measurements similarly as in Equation 6, the *Supervisor* can calculate the maximum allowed drug dosage as follows:

$$\max drug_{safe} = \frac{1}{\alpha} \log \frac{dl_{max} - dl_{cur}}{dl_{max} - Hdl_2} \cdot df,$$

where df presents the drug flow when the *PCA pump* is on. The infusion pump is programmed to inject at most $\max drug_{safe}$ until the *Supervisor* sends a revised maximum allowed dosage. This system is inherently fail-safe since the worst consequence of failed network, *Supervisor*, or *PO* is no drug injection.

5. RELATED WORK

Formal methods have traditionally been used for verification of time-critical and safety-critical embedded systems [13]. Until recently these methods have not been used for medical device certification. The authors in [7] presented the use of Extended Finite State Machines for model checking of the Computer Automated Resuscitation A medical device. Formal techniques have also been applied to improve medical device protocols [14] and safety [15]. However, the authors either used a simplified patient model or did not model the patient at all.

Continuous monitoring of the blood oxygenation of patients receiving PCA infusions has been done in the past and even commercially implemented. The Alaris 8210 SpO2 Module connects to the Alaris 8000 pump controller and adds the ability to pause infusions based on a target SpO_2. Our approach shows how a similar system could be designed and validated. In particular, while the available commercial system is provided as a tightly-integrated system from a single vendor, our approach could be used to design and validate systems based on devices from multiple sources as long as the timing and other necessary information is available.

6. DISCUSSION

One of the biggest problems in designing physiologically closed-loop control for medical scenarios is the lack of appropriate plant models (in this case, patient bodies). There is a vast body of physiological knowledge in the medical literature. For the PCA scenario, pharmacokinetic models of drug absorption are known (e.g., [11]), and there is statistical data on the effects of the drug on vital signs. However, we are not aware of a unified model that captures the whole process and is appropriate for the control-theoretic study of the closed loop dynamics. Constructing such a model is beyond the scope of this paper but is an important avenue of future research. Once an appropriate model is in place, supervisory adaptive control technqiues [16, 17] can be used to address parametric uncertainty and adapt to the continuously fluctuating parameters due to the changes in the patient condition.

7. CONCLUSION

We have presented a model-driven approach to design and validate closed-loop medical device systems. The approach combines simulation-based validation of a continuous-time system model in Matlab with formal verification of a more abstract model using timed automata and the UPPAAL tool. The key to keeping the two models consistent is to derive timing parameters of the system from the Matlab model and use these constants in the UPPAAL model.

We offer a case study of the proposed approach using a clinically relevant system. The dynamics system we use is relatively simple. This choice is made on purpose, to better present the steps in our approach. With more complicated dynamics, some of the steps become more difficult; in particular, derivation of timing parameters may require more sophisticated methods and further research may be required. For the case study, we have shown that the system is safe under no failure assumptions. We identify how to deal with some of the failures that manifest as unbounded delays. The proposed method is based on the well-known notion of timed lease used in fault-tolerant distributed systems. We believe that such a technique can be applied to other tightly integrated medical systems in which fail safe is essential.

Although the presented case study is relatively simple, we believe that our approach allows us to construct safety cases for regulatory approval of closed-loop medical systems.

8. REFERENCES

[1] ASTM International. *STAM F2761-2009. 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*, 2009.

[2] J.M. Goldman, R.A. Schrenker, J.L. Jackson, and S.F. Whitehead. Plug-and-play in the operating room of the future. *Biomedical Instrumentation and Technology*, 39(3):194–199, 2005.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] G. Behrmann, A. David, and K.G. Larsen. A tutorial on uppaal. In *Formal Methods for the Design of Real-Time Systems (revised lectures)*, volume 3185 of *LNCS*, pages 200–237, 2004.

[5] S. Fischmeister, J. M. Goldman, I. Lee, and R. Trausmuth. Plug-and-play for medical devices: Experiences from a case study. *Biomedical Instrumentation & Technology*, 43(4):313–317, 2009.

[6] T. K. Nuckols, A. G. Bower, S. M. Paddock, L. H. Hilborne, P. Wallace, J. M. Rothschild, A. Griffin, R. J. Fairbanks, B. Carlson, R. J. Panzer, and R. H. Brook. Programmable infusion pumps in ICUs: An analysis of corresponding adverse drug events. *Journal of General Internal Medicine*, 23(Supplement 1):41–45, January 2008.

[7] D. Arney, R. Jetley, P. Jones, I. Lee, and O. Sokolsky. Formal methods based development of a PCA infusion pump reference model: Generic Infusion Pump (GIP) project. In *HCMDSS-MDPNP '07: Proceedings of the 2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, pages 23–33, Washington, DC, USA, 2007. IEEE Computer Society.

[8] V. Chan and S. Underwood. A single-chip pulsoximeter design using the MSP430. Technical Report SLAA274, Texas Instruments, November 2005.

[9] P. Jevon and B. Ewens, editors. *Monitoring the Critically Ill Patient*. Wiley-Blackwell, 2nd edition, 2007.

[10] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, volume 131 of *LNCS*, pages 52–71, 1981.

[11] J. X. Mazoit, K. Butscher, and K. Samii. Morphine in postoperative patients: Pharmacokinetics and pharmacodynamics of metabolites. *Anesthesia and Analgesia*, 105(1):70–78, 2007.

[12] Rajeev Alur, Aditya Kanade, S. Ramesh, and K. C. Shashidhar. Symbolic analysis for improving simulation coverage of simulink/stateflow models. In *EMSOFT '08: Proceedings of the 8th ACM international conference on Embedded software*, pages 89–98, New York, NY, USA, 2008. ACM.

[13] E. M. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Comput. Surv.*, 28(4), 1996.

[14] R. Alur, D. Arney, E. L. Gunter, I. Lee, J. Lee, W. Nam, F. Pearce, S. Van Albert, and J. Zhou. Formal specifications and analysis of the computer-assisted resuscitation algorithm (CARA) Infusion Pump Control System. *Intl. J. Software Tools for Technology Transfer*, 5(4):308–319, 2004.

[15] A. ten Teije, M. Marcos, M. Balser, J. van Croonenborg, C. Duelli, F. van Harmelen, P. Lucas, S. Miksch, W. Reif, K. Rosenbrand, and A. Seyfang. Improving Medical Protocols by Formal Methods. *Artif. Intell. Med.*, 36(3):193–209, 2006.

[16] A. S. Morse. Supervisory control of families of linear set-point controllers – Part 1: Exact matching. *IEEE Transactions on Automatic Control*, pages 1413–1431, October 1996.

[17] A. S. Morse. Supervisory control of families of linear set-point controllers – Part 2: Robustness. *IEEE Transactions on Automatic Control*, pages 1500–1515, November 1997.