A Modal Specification Approach for On-Demand Medical Systems[*]

Andrew L. King, Lu Feng[**], Oleg Sokolsky, and Insup Lee

Department of Computer & Information Science, University of Pennsylvania
{kingand,lufeng,sokolsky,lee}@cis.upenn.edu

Abstract. The on-demand approach, where systems are assembled from components by lay users, has seen success in the consumer electronics industry. Currently, there is growing demand for on-demand capabilities in medical systems so caregivers can create larger medical systems from smaller medical devices. Unlike consumer electronics, medical systems pose challenges for the on-demand approach due to attributes such as device complexity, device variability and safety requirements. In this paper, we propose a formal specification language for on-demand (medical) systems. Our approach is based on the formalism of Modal I/O Automata, which allows system designers to express complex device requirements and can be used to reason about safety and liveness properties of on-demand medical systems directly from their specifications. We illustrate the applicability of our approach through a case study of a closed-loop patient controlled analgesia system.

1 Introduction

An on-demand system is any system assembled by a lay user out of components that have not been previously tested together. An example on-demand system is a home entertainment system: A typical home theater is composed of speakers, an audio/visual receiver, a content source (such as a DVD player or streaming video device), and television. Assembly of these on-demand systems is facilitated by a 'plug-and-play' capability in the components themselves; in theory each component conforms to a well defined standard (*e.g.,* USB, HDMI, Bluetooth) which then ensures that the components properly compose to form a functioning system. The standards typically define a small set of rigid component classes. For example, the USB standard defines around 20 classes for different component types such as mass storage (external hard drives), human interface device (keyboards and mice), audio, and video. The feature sets for each class are fixed (and relatively simple).

[*] Research is supported in part by NSF grants CNS-1035715 and IIS-1231547.
[**] Lu Feng is supported by James S. McDonnell Foundation 21st Century Science Initiative - Postdoctoral Program in Complexity Science/Complex Systems - Fellowship Award.

Recently, there has been interest in on-demand medical systems where health care workers can assemble larger medical systems out of smaller medical devices at the bedside in order to provide better therapy for their patients [25]. These on-demand systems would be used to provide better physiologic alarms (by combining data streams from multiple medical devices) and closed loop control (using physiologic sensors to drive actuators). While there is demand from the medical community, critical care medical systems have two major attributes which pose engineering challenges for the on-demand approach.

First, medical devices tend to be more **complex**. Unlike consumer electronics, critical care medical devices are very complex and variable, even among devices designed for a similar purpose (*e.g.,* infusion pumps). Often, this complexity and variability is the result of different ways device manufacturers have chosen to mitigate certain safety hazards. The complexity and variability means it is difficult to capture the range of behavior of a single device class in a standard similar to USB where the features and behavior of each class are fully enumerated beforehand. Second, on-demand medical systems serve a **safety critical** purpose; if the composite system malfunctions or is implemented incorrectly injury or death could result. Traditional safety critical systems such as aircraft, nuclear power plants and standalone medical devices are evaluated for safety before they are delivered to the user. The state of the art in safety assesment is to consider the completely assembled system as a whole. In on-demand medical systems this would not be possible because each system instance may be assembled by combining devices that have never been tested together. There must be some mechanism in place to analyze the behavior of *all* instantiations of a on-demand system in order to ensure that any instantiation only exhibits safe behavior.

There have been a number of high-level proposals for how to achieve safe on-demand medical systems [20,11,25,5]. These proposals all involve separating system functionality between *interoperable* componenets: *coordination applications* (apps), medical devices, and a Medical Application Platform (MAP). In these proposals, each type of system component would be regulated, certified, and then obtained by the health-care organization separately [12]. We now provide a brief overview of the role and use of each system component type.

Apps are software programs that provide the coordination algorithm for a specific clinical scenario (*i.e.,* smart alarms, closed-loop control of devices, *etc.*). In addition to executable code, these apps contain *device requirements* declarations: a formal model of the medical devices they need to operate correctly. These apps would be validated and verified against their requirements specification before they are marketed. Symmetrically, the interoperable medical devices carry a self-descriptive model, known as a *capabilities* specification. Each medical device would be certified that it conforms to its specification before it is marketed and sold to end users.

The MAP provides a trusted base: It executes the coordination apps and facilitates the assembly of the on-demand system. When a user connects a medical device to the network, that medical device will transmit its capabilities specification to the MAP. Likewise, when a user attempts to launch an app, the MAP analyzes the app's requirements specification and the connected devices' capabilities specifications. If the devices do not have the required capabilities, the MAP will prevent the app from running and notify the user. This functionality is critical to the assembly of on-demand medical systems because it is the foundation for any safety or effectiveness claim; in theory only systems which exhibit the behavior captured by the app (and its associated requirements specification) will be instantiated. This enables various stakeholders (*e.g.,* app developers and regulatory agencies) to verify and validate the behavior of all possible instantiations of an on-demand system by checking the behavior of the application against its requirements specification.

Finally, each of these components would implement an *interoperability standard.* The standard would specify allowed network transport protocols, how medical and system information is encoded on the network, provide a basic means to establish an interconnection between components, and expose logical interfaces for the transmission of data or commands. Current interoperability standards, such as IEEE-11073 POC [16], IEEE-11073 PHD [9,8] and Health Level 7 (HL7) [10] focus mainly on *data* interoperability (*i.e.,* they provide a mechanism for the exchange of data). Notably absent in current standards is the ability to address the reactive behavior of various medical systems. This means that current standards are largely unsuitable for interoperable medical systems where multiple devices are coordinating to provide autonomous delivery of care.

Our ability to reason about an on-demand medical system *a priori* (*i.e.,* before it is instantiated) depends on how the app requirements and device capabilities are specified. There are three major goals that must be met by a suitable specification language:

G1 The language must enable us to automatically relate app device requirements specifications to device capabilities specifications: properties which must hold for apps composed with their requirements specification must hold for any ad hoc system where the app is composed with compatible devices.

G2 The language must enable app developers to explicitly specify variability in required device behavior: if all safety properties are satisfied with a highly variable device requirements specification it means the app is compatible with a larger set of medical devices.

G3 The language must be expressive enough to specify arbitrarily complex and reactive behavior.

In this paper, we propose a modal specification language for on-demand systems which addresses **G1**, **G2** and **G3**. This formalism could be layered onto existing standards to enable the specification of device behavior. Syntactically, it is a simple, state-based language inspired by Alur and Henzinger's *Reactive Modules* formalism [1]. Semantically, each module defined in the language is equivalent to a modal I/O automaton (MIOA) [24], a formalism that extends labeled transitions systems with a may/must modality on transitions and an input/output/internal distinction on action labels.

MIOAs are well suited for reasoning about on-demand systems for the following reasons. First, the may/must distinction of transitions is useful for specifying the behavioral variability of devices: must transitions denote required behavior and may transitions represent allowed behavior. Thus, we can use MIOAs to reason about all possible instantiations of an on-demand system based on the specification. Second, we show that the *weak modal refinement* relation of MIOAs preserves both *safety* (nothing bad happens) and *liveness* (something good eventually happens) properties. The guarantee of safety and liveness properties are essential for on-demand medical systems, *e.g.,*

– "the patient's SpO_2 level should never be lower than 95" (safety)
– "the laser must be completely deactivated to allow the flow of the oxygen concentrate from the ventilator" (liveness)

Third, the compositionality of MIOAs allows us to easily verify the behavior of component-based on-demand systems by verifying their specifications. For example, if we find a medical device that refines an app's requirements specification, then we can claim that the system created by composing the app with that device will satisfy all safety and liveness properties satisfied by the composition of that app and its requirements specification.

We have prototyped our approach using the MIO Workbench tool [6] and the PRISM model checker [22], and applied it to a few case studies. In this paper, we report on the application of our approach to the case study of a closed loop patient controlled analgesia system.

The rest of this paper is organized as follows: in Section 2 we describe one possible application of on-demand medical systems as a motivating example. In Section 3 we describe our proposed specification language and the underlying MIOA semantics. Section 4 contains a case study where we apply language to specify an on-demand medical system and then verify properties of that system. We conclude the paper, discuss current weaknesses of the approach, and propose directions for future work in Section 5.

2 Motivating Example

In this section we describe Patient Controlled Analgesia (PCA), the hazards of PCA, and how a closed-loop system could be used to mitigate those hazards. The purpose of the example itself is twofold. First it illustrates how the functionality of the closed-loop system can be divided between an app and medical devices. Second, it allows us to show variability among the same class of medical device and how that variability can affect the safety of an on-demand system.

After trauma (*e.g.,* invasive surgery) patients convalescing in an ICU are often placed on PCA therapy for pain management. During PCA therapy, patients are attached to an infusion pump loaded with a painkiller (*e.g.,* an opiod). When the patient desires additional pain-relief, they press a trigger which causes the pump to deliver a bolus of medication. While PCA lets patients manage their own pain-levels effectively [15] it also creates an opportunity for overdose. Opiod

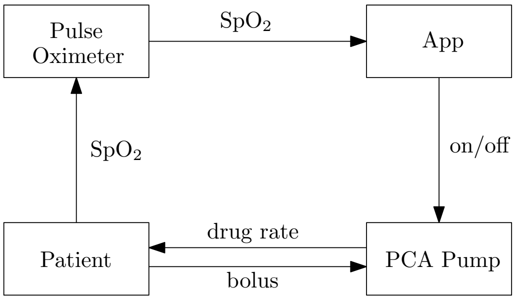


Fig. 1. A closed-loop PCA system

overdose can result in respiratory depression, which in turn can result in injury or death [17,27,13]. One possible way to mitigate the hazard posed by PCA is to 'close the loop': analyze data from sensors attached to the patient in real-time to determine if the patient is nearing respiratory distress and, if so, disable the pump [26,19].

As shown in Figure 1, the system consists of four components: a pulse oximeter, an app, a PCA infusion pump and a patient. The system operates in a closed-loop fashion: the pulse oximeter keeps monitoring the patient's SpO_2 value (measure of blood oxygenation), and the app controls the PCA pump based on the SpO_2 value read from the pulse oximeter (the app would stop the pump if the detected SpO_2 value is lower than 95). When the PCA pump is turned on, it delivers drug to the patient with a normal infusion rate pre-programmed by the caregiver; however, if it receives a bolus request from the patient, then a higher drug dose will be supplied, unless the pump is stopped by the app.

This application has one main safety property: the infusion rate of the pump should always be 0 (*i.e.,* the pump is off) whenever the patient has an SpO_2 below a certain threshold (*e.g.,* $SpO_2 < 95$). The satisfaction of this property depends on both the algorithm implemented by the app and the behavior of the PCA pump the app is managing. Figure 2 illustrates the behavior of three different types of PCA pumps as state machines. Figure 2a represents a simple infusion pump that infuses while it receives the *on* signal. Technically speaking, the pump of Figure 2a is not a PCA pump because it doesn't provide any mechanism for the patient to request a bolus. Figure 2b represents a PCA pump that will infuse at a rate of 1 while it receives the *on* signal and it will infuse at a rate of 2 when the patient requests a bolus, even if it is receiving the *off.* Finally, Figure 2c represents an even more complex PCA pump. This pump will autonomously disable itself under a number of conditions in order to mitigate several hazards associated with infusion pumps [4]. For example, if the pump detects air bubbles in the infusion line, it will halt infusion and raise an alarm in order to prevent an air-embolism.

So, which pump should we choose to use? If we plug any of these three pumps into the system, will the patient's safety be guaranteed? And is there an easy way to verify the effectiveness of the PCA system? These questions will be answered

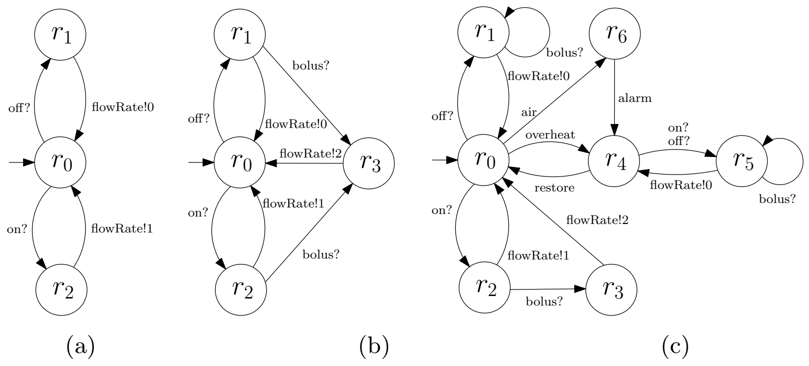


Fig. 2. MIOAs for three different PCA pump devices

in Section 4 by applying our modal specification approach, which is explained in the next section.

3 A Specification Language for On-Demand Systems

In this section, we propose a simple, state-based specification language for describing the requirements of on-demand systems. This language can be applied, for example, by app developers to specify the desirable behavior of compatible medical devices. The language is based on Alur and Henzinger's *Reactive Modules* formalism [1], with the extension of transition modality (*i.e.,* distinguishing *must* and *may* transitions). We first define the syntax of our proposed language in Section 3.1, and then give its semantics in Section 3.2.

3.1 Syntax

We define each component of an on-demand system as a module M, which is a tuple $\big((in_M, out_M, int_M), var_M, (must_M, may_M)\big)$ where (in_M, out_M, int_M) is the signature of M representing sets of *input*, *output* and *internal* actions, var_M is a set of state variables, and $(must_M, may_M)$ are sets of *must* and *may* transitions.

A module communicates with the external environment via input and output actions in the CSP style [14]; that is, an input (resp. output) action, denoted by $c?v$ (resp. $c!v$), enables the module to receive (resp. send) message v over a named channel c. The message v, which for example can be an expression about some previous inputs or a valuation of some local variable of the module, must be typed. Sometimes, message v is omitted from an input/output action, denoted by $c?$ or $c!$, with the interpretation that there is only a single possible message can be transmitted through channel c. The set of internal actions int_M, representing internal events of module M, are not observable to the external environment.

module M_1

input: a?**integer[0..1]**, b?;
output: c!;
internal: τ;

s : [0..3] **init** 0;

[a?v] $(s = 0) \xrightarrow{\text{must}} (s' = 1)$;
[c!] $(s = 2) \xrightarrow{\text{must}} (s' = 3)$;
[τ] $(s = 3) \xrightarrow{\text{must}} (s' = 3)$;

[τ] $(s = 0) \xrightarrow{\text{may}} (s' = 2)$;
[b?] $(s = 1) \xrightarrow{\text{may}} (s' = 3)$;

endmodule

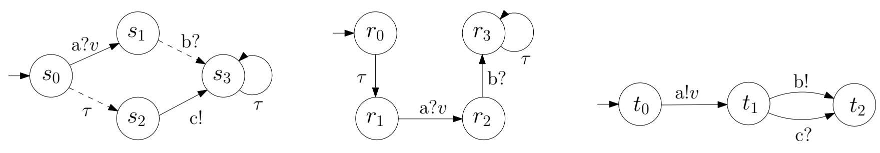
Fig. 3. An example module specification

The variable set var_M defines the local state space of module M. A state variable $s \in var_M$ can be either a Boolean value, or an integer within a predefined finite range. We suppose that each variable s has an initial value $\overline{s}$.

The behavior of module M is defined by the set of must/may transitions $(must_M, may_M)$. Each transition $t \in must_M \cup may_M$ takes the form (a, g, m, u), comprising an action label a, a guard g, a modality label m and an update u. The action $a \in in_M \cup out_M \cup int_M$ can be either an input/output action synchronizing with the external environment, or an internal event occurring within the module. The guard g is a predicate over the state variables var_M, determining if transition t is enabled. The modality label $m \in \{must, may\}$ indicates if the transition is required (*i.e.,* must occur) in all implementations or if it is allowed (*i.e.,* may occur) in any implementation. The update u describes the effect of transition t on state variables; more specifically, $u = (s'_1 = expr_1) \wedge \cdots (s'_n = expr_n)$ where s'_i denotes the updated value of state variable $s_i \in var_M$, and $expr_i$ is an expression in terms of the state variables.

Example 1. Figure 3 shows an example module M_1 described in our proposed specification language. The description is split into four parts, defining actions, state variables, as well as must and may transitions of the module. The module M_1 has two input actions: "a?**integer[1..2]**" that receives a integer value within the range $\{1, 2\}$ via channel "a", and "b?" that receives an input via channel "b" (the message is omitted). The module may send an output through action "c!", and has a single internal action τ.

We also see that M_1 has a single interger-valued variable s with the range $\{0, \ldots, 3\}$ and an initial value 0. There are four transitions in M_1. Each transition $t = (a, g, m, u)$ is written in a line "[a] $g \xrightarrow{m} u$;". For example, line "[a?v] $(s = 0) \xrightarrow{\text{must}} (s' = 1)$;" represents a "must" transition with action "a?v", guard $(s = 0)$ and update $(s' = 1)$.



(a) $\mathcal{M}_1$ (b) $\mathcal{M}'_1$ (c) $\mathcal{M}_2$

Fig. 4. Three example MIOAs

3.2 Semantics

The semantics of our specification language is based on Modal I/O Automata [24] which in turn are extensions of Modal Transition Systems [23]. A modal I/O automaton (MIOA) can be considered as a (nondeterministic) state transition system with an input/output/internal distinction on action labels and a must/may distinction on its transition relations.

Definition 1 (MIOA). *A modal I/O automaton P is a tuple $(S_P, \overline{s}_P, in_P, out_P, int_P, \rightarrow_{\Box P}, \rightarrow_{\Diamond P})$ where S_P is a finite set of states, $\overline{s}_P \in S_P$ is an initial state, in_P, out_P and int_P are disjoint sets of input, output and internal actions, $\rightarrow_{\Box P} \subseteq S_P \times act_P \times S_P$ is the must transition relation describing required behavior, and $\rightarrow_{\Diamond P} \subseteq S_P \times act_P \times S_P$ is the may transition relation describing allowed behavior ($act_P = in_P \cup out_P \cup int_P$).*

The mapping from a module $M = \big((in_M, out_M, int_M), var_M, (must_M, may_M)\big)$ described in our proposed specification language to a MIOA P is straightforward. We define the state space S_P of P to be the set of all valuations of the state variables in var_M. The initial state $\overline{s}_P$ is given by the initial values of variables in var_M. The action sets $in_P = in_M$, $out_P = out_M$, and $int_P = int_M$. And each transition $t \in must_M$ (resp. $t \in may_M$) maps to a transition $p \xrightarrow{a}_{\Box P} p'$ (resp. $p \xrightarrow{a}_{\Diamond P} p'$) in P, where p and p' are states given by the guard and update of t, respectively.

Example 2. The module M_1 described in Figure 3 maps to the MIOA $\mathcal{M}_1$ shown in Figure 4a. There are four states $\{s_0, s_1, s_2, s_3\}$ in $\mathcal{M}_1$, each of which maps to a valuation of variable $s = i$ for $i \in \{0, \ldots, 3\}$ in M_1 . The initial state of $\mathcal{M}_1$ is s_0, indicated by an incoming arrow in Figure 4a. The must transitions are drawn in solid arrows, while the may transitions are drawn in dashed arrows.

In this paper, we consider only *syntactically consistent* MIOA where $\rightarrow_{\Box P} \subseteq \rightarrow_{\Diamond P}$, *i.e.,* every required transition is also allowed. If the must and may transition relations of a MIOA P coincide, denoted $\rightarrow_{\Box P} = \rightarrow_{\Diamond P}$, then we call P an *implementation.* An abstract MIOA with nonempty must and may transition relations specifies a set of concrete implementations: a must transition asks that any legal implementation must include that transition, while a may transition indicates that implementations are allowed (but not required) to have that transition. Formally, the relation between an abstract specification MIOA and a concrete implementation MIOA is captured by *refinements.* There are many different types of refinement relations between MIOAs [24,7]. In our setting, we adopt the *weak modal refinement* relation [7], which ensures that the observable behavior of an implementation (*e.g.,* actual medical device) refines the specification (*e.g.,* app requirements).

We need the notion of *weak transitions* to reason about the observable behavior of MIOAs. Given an input/output action a of a MIOA P, there exists a weak must transition between states p and p', denoted by $p\ (\xrightarrow{a}_{\Box P})^*\ p'$, iff there exist a pair of states $p_1, p_2 \in S_P$ such that $p\ (\xrightarrow{\tau}_{\Box P})^*\ p_1 \xrightarrow{a}_{\Box P} p_2\ (\xrightarrow{\tau}_{\Box P})^* p'$, where τ denotes any arbitrary internal action and $p\ (\xrightarrow{\tau}_{\Box P})^*\ p_1$ represents finitely many (zero or more) transitions from p to p_1 labelled with internal actions. The notion of weak may transitions $p\ (\xrightarrow{a}_{\Diamond P}\ )^*\ p'$ can be defined analogously.

Definition 2 (Weak Modal Refinement). *Given two MIOAs P and Q with $in_P = in_Q$ and $out_P = out_Q$, a relation $R \subseteq S_P \times S_Q$ is called a* weak modal refinement *for P and Q iff for all $(p, q) \in R$ and $a \in act_P \cup act_Q$ it holds that:*

- *if $q \xrightarrow{a}_{\Box Q} q'$ then there exists $p' \in S_P$ such that $p\ (\xrightarrow{\hat{a}}_{\Box P})^*\ p'$ and $(p', q') \in R$,*
- *if $p \xrightarrow{a}_{\Diamond P}\ p'$ then there exists $q' \in S_Q$ such that $q\ (\xrightarrow{\hat{a}}_{\Diamond Q}\ )^*\ q'$ and $(p', q') \in R$,*

where $(\xrightarrow{\hat{a}}_{\Box P})^* = (\xrightarrow{a}_{\Box P})^*$ if $a \in int_P \cup out_P$, and $(\xrightarrow{\hat{a}}_{\Box P})^* = (\xrightarrow{\tau}_{\Box P})^*$ otherwise. If there exists a refinement relation R such that $(\overline{s}_P, \overline{s}_Q) \in R$, then we claim that P weakly modally refines Q, denoted by $P \leq^*_m Q$.

The weak modal refinement relation defined above allows implementations to contain different (*i.e.,* unspecified) internal behavior as long as the internal behavior does not prevent the implementation from performing required external behavior. The refinement also prevents implementations from performing 'extra' external behavior as long as the extra behavior is specified in the signature of the specification. It does allow implementations to introduce new external behavior if that behavior is an action-label not specified in the specification; in this case, the transitions labled with that action are treated as τ transitions.

Example 3. The MIOA $\mathcal{M}'_1$ shown in Figure 4b weakly modally refines the MIOA $\mathcal{M}_1$ shown in Figure 4a, under relation $R = \{(r_0, s_0), (r_1, s_0), (r_2, s_1), (r_3, s_3)\}$. Note that the required transition $s_2 \xrightarrow{c!}_{\Box} s_3$ in $\mathcal{M}_1$ does not have a mapping in $\mathcal{M}'_1$, because $\mathcal{M}'_1$ does not contain a refinement of the allowed transition $s_0 \xrightarrow{\tau}_{\Diamond}\ s_2$ in $\mathcal{M}_1$ such that no refinement of state s_2 is reachable in $\mathcal{M}'_1$.

Recall that, in our proposed specification language, a module (medical device) can comminucate with the external environment via sending/receiving messages. We now introduce a binary *composition* operator [24] to reason about the message passing. We say that two MIOAs P_1 and P_2 are *composable* iff the overlapping of their actions only occur on complementary types, *i.e.,* $(in_1 \cup int_1) \cap (in_2 \cup int_2) = \emptyset$ and $(out_1 \cup int_1) \cap (out_2 \cup int_2) = \emptyset$.

Definition 3 (Composition). *The composition of two composeable MIOAs P_1 and P_2 is given by a MIOA $P_1 \otimes P_2 = (S, \overline{s}, in, out, int, \rightarrow_{\Box}, \rightarrow_{\Diamond})$, where the state*

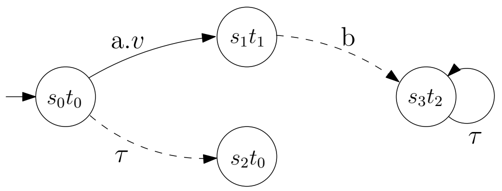


Fig. 5. A MIOA composition $\mathcal{M}_1 \otimes \mathcal{M}_2$.

space $S = S_1 \times S_2$, the initial state $\overline{s} = (\overline{s}_1, \overline{s}_2)$, $in = (in_1 \backslash out_2) \cup (in_2 \backslash out_1)$, $out = (out_1 \backslash in_2) \cup (out_2 \backslash in_1)$, $int = int_1 \cup int_2 \cup (in_1 \cap out_2) \cup (in_2 \cap out_1)$, and the transition relations are given by the following rules (for $\gamma \in \{\Box, \Diamond\}$):

$$\frac{p_1 \overset{a!}{\rightarrow}_\gamma p_1' \quad p_2 \overset{a?}{\rightarrow}_\gamma p_2'}{p_1 \otimes p_2 \overset{a}{\rightarrow}_\gamma p_1' \otimes p_2'} \qquad \frac{p_1 \overset{a?}{\rightarrow}_\gamma p_1' \quad p_2 \overset{a!}{\rightarrow}_\gamma p_2'}{p_1 \otimes p_2 \overset{a}{\rightarrow}_\gamma p_1' \otimes p_2'}$$

$$\frac{p_1 \overset{a}{\rightarrow}_\gamma p_1' \quad a \notin act_2}{p_1 \otimes p_2 \overset{a}{\rightarrow}_\gamma p_1' \otimes p_2} \qquad \frac{a \notin act_1 \quad p_2 \overset{a}{\rightarrow}_\gamma p_2'}{p_1 \otimes p_2 \overset{a}{\rightarrow}_\gamma p_1 \otimes p_2'}$$

Example 4. Figure 5 shows the composition of two MIOAs $\mathcal{M}_1$ (Figure 4a) and $\mathcal{M}_2$ (Figure 4c). $\mathcal{M}_1$ expects an input action "a?v" in state s_0 while $\mathcal{M}_2$ sends an output action "a!v" in state t_0, these two transitions are composed into a transition $s_0t_0 \xrightarrow{\text{a}.v}_\Box s_1t_1$ in $\mathcal{M}_1 \otimes \mathcal{M}_2$. Similarly, the may transition $s_1 \xrightarrow{\text{b?}}_\Diamond s_3$ in $\mathcal{M}_1$ composes with the must transition $t_1 \xrightarrow{\text{b?}}_\Box t_2$ in $\mathcal{M}_1$, resulting in a may transition $s_1t_1 \overset{\text{b}}{\rightarrow}_\Diamond s_3t_2$ in the product. Since τ is an internal action of $\mathcal{M}_1$, the τ-labelled transitions of $\mathcal{M}_1$ compose with atomic transitions (self-loops) of $\mathcal{M}_2$, see for example, $s_0t_0 \overset{\tau}{\rightarrow}_\Diamond s_2t_0$. Note that s_2t_0 is a deadlock state.

A motivation of our work is to design a specification language that makes verifying properties of medical devices easy. In particular, we are interested in two kinds of properties: *safety* (something bad will never happen) and *liveness* (something good will eventually happen). Formally, a safety property ϕ_{safe} is defined as a liner-time property over a set of actions such that any infinite word w where ϕ_{safe} does not hold contains a bad prefix (*i.e.,* a finite prefix w' where the bad thing has happened); a liveness property ϕ_{live} (over a set of actions) is an linear-time property such that each finite word can be extended to an infinite word that satisfies ϕ_{live}.

Given a MIOA P and a safety or liveness property ϕ over act_P, we define two satisfaction relations:

– $P \models_\Box \phi$, under-approximates "all refinements of P satisfy ϕ";
– $P \models_\Diamond \phi$, over-approximates "some refinement of P satisfies ϕ".

To verify property ϕ on P, we need to prove that $P \models_\Box \phi$ is true; and to refute property ϕ, we need to establish that $P \models_\Box \neg\phi$ holds.

We prove that weak modal refinement preserves the verification of safety and liveness properties as follows.

Lemma 1. *Let P and Q be two MIOAs such that $P \leq_m^* Q$. Given a safety property ϕ_{safe} over actions $in_P \cup out_P$, if $Q \models_\Box \phi_{safe}$, then $P \models_\Box \phi_{safe}$.*

Proof. For the sake of contradiction, suppose $Q \models_\Box \phi_{safe}$ and $P \not\models_\Box \phi_{safe}$. The latter means that there may exist some path ρ in P containing a bad prefix. For simplicity, assume that the bad behavior is represented by a single word a. We have a path $\rho = p_0 \rightarrow \cdots \rightarrow p_{n-1} \xrightarrow{a}_{\Diamond P} p_n$. Based on $P \leq_m^* Q$ and Definition 2, there exists a pair of states q and q' in Q such that $(p_{n-1}, q) \in R$, $(p_n, q') \in R$ and $q\ (\xrightarrow{\tau}_{\Diamond Q}\ )^*\ q_{i-1} \xrightarrow{a}_{\Diamond Q}\ q_i\ (\xrightarrow{\tau}_{\Diamond Q}\ )^* q'$. Therefore, there may exist a path in Q containing the bad word a, which is a contradiction with $Q \models_\Box \phi_{safe}$. □

Lemma 2. *Let P and Q be two MIOAs such that $P \leq_m^* Q$. Given a liveness property ϕ_{live} over actions $in_P \cup out_P$, if $Q \models_\Box \phi_{live}$, then $P \models_\Box \phi_{live}$.*

Proof. Since $Q \models_\Box \phi_{live}$, every (infinite) path in Q must eventually reach the good condition. For simplicity, assume that the good condition is represented by a single word a. Then, there must exists a pair of states q_{n-1} and q_n in any (infinite) path ρ of Q such that $\rho = q_0 \rightarrow \cdots \rightarrow q_{n-1} \xrightarrow{a}_{\Box Q} q_n \rightarrow \cdots$. Based on $P \leq_m^* Q$ and Definition 2, for each pair of q_{n-1} and q_n, there must exists a pair of corresponding states p, p' in P such that $(p, q_{n-1}) \in R$, $(p', q_n) \in R$ and $p\ (\xrightarrow{\tau}_{\Box P})^*\ p_{i-1} \xrightarrow{a}_{\Box P} p_i\ (\xrightarrow{\tau}_{\Box P})^* p'$. Therefore, every infinite path in P must eventually reach the good condition, so that we have $P \models_\Box \phi_{live}$. □

Another nice feature about weak modal refinement is the following compositionality result:

Theorem 1 (Compositionality [2]). *Let P_1,P_1' and P_2 be MIOAs (P_1 and Q are composable). If $P_1' \leq_m^* P_1$, then $P_1' \otimes P_2 \leq_m^* P_1 \otimes P_2$.*

Based on the above theorem and Lemmas 1 and 2, we can verify safety and liveness properties on medical devices without composing the (large) complete systems. For example, let P_1 be the specification for some medical device, P_1' be an implementation of the actual device, P_2 be the external environment (*e.g.,* app, patient) and ϕ be the desirable safety or liveness property. When designing the specification, the app developers make sure that $P_1 \otimes P_2 \models_\Box \phi$. We only need to check whether the device P_1' weakly modally refines the specification P_1. If $P_1' \leq_m^* P_1$ holds, then we can claim that $P_1' \otimes P_2 \models_\Box \phi$ without actually verifying the composed system $P_1' \otimes P_2$.

4 Case Study

Now we apply our proposed specification approach to analyze the closed-loop PCA example from Section 2. We model each system component (*i.e.,* app, patient, Pulse Oximeter and PCA pump) as a MIOA. To answer the question that, from the three PCA pumps shown in Figure 2, which one should we choose

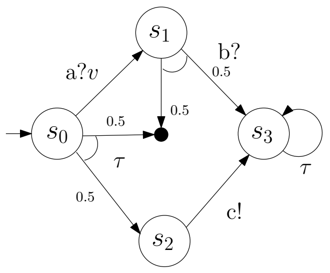


Fig. 6. Translation of the MIOA in Figure 4a for PRISM

to use, we use the MIO Workbench tool [6] to check weather the MIOA of a pump weakly modally refines the MIOA of the pump specification. If the refinement relation holds, then the pump is good in the sense that the PCA system should be able to guarantee the required safety property; otherwise, the pump is not safe to use.

To demonstrate that our approach can help to preserve system properties, we also use the (probabilistic) model checker PRISM [22] to verify the required safety property of composed PCA systems because of PRISM's relative maturity and ease of use. PRISM does not actually support the verification of MIOAs. We instead translate MIOAs as probabilistic automata (PAs): each must transition becomes a transition with probability 1 and each may transition becomes a transition with probability 0.5 [1]. For example, Figure 6 shows the probabilistic translation of the MIOA in Figure 4a; the may transition from s_0 to s_2 now becomes a transition with probability 0.5 (the black dot is the sink state). If PRISM verifies that certain property is *true* with probability 1, then the property "must" be satisfied by the on-demand medical system; and if the verification result is a real value p such that $0 < p < 1$, then the system "may" satisfy the property.[2]

4.1 Modelling

Figure 7 shows the detailed model for the pulse oximeter, which is a two-state modal I/O automaton synchronizing with the patient model via a must transition labelled with the input action "toSensor?SpO_2" and then immediately sending the data to the app via another must transition with the output action "toApp!SpO_2". Note that, in both actions, $SpO_2 \in [0, 100]$ is an integer value transmitted over the input/output channel.

[1] To make probabilistic distributions full, we complement each may transition with a transition (with probability 0.5) to a sink state; however, the verification result would exclude all the paths leading to the sink state.

[2] While an evaluation of scalability is beyond the scope of this paper, PRISM implements a number of efficient model-checking algorithms for probabilistic systems and can scale to models with at least 10^7 states.

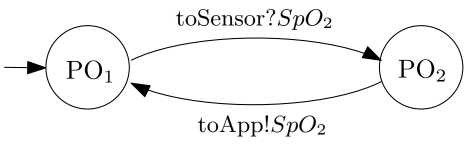


Fig. 7. MIOA for the pulse oximeter

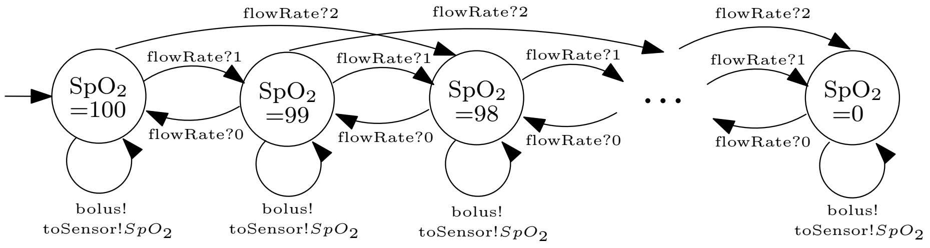


Fig. 8. MIOA for the patient dynamics

We adopt a simple patient model which only considers the patient's discrete SpO_2 measurements.[3] As shown in Figure 8, each state of the MIOA for the patient model represents a single SpO_2 value, ranging from 0 to 100. With an initial value of 100, the patient's SpO_2 measurement decreases by 1 or 2 upon receiving drug from the PCA pump at a normal infusion rate ("flowRate?1") or a bolus rate ("flowRate?2"), respectively. On the other hand, the SpO_2 value increases by 1 if the PCA pump stops ("flowRate?0"), modeling the restoration of the patient's vital sign as the drug concentration reduces. At any point, the patient has the option of pressing the button to request one more dose from the PCA pump (denoted by the output action "bolus!"). The patient's SpO_2 level is constantly monitored by the pulse oximeter via synchronizing over "toSensor!SpO_2".

The behavior of the control app is illustrated in Figure 9 using our proposed specification language[4]. The app has an input action "toApp?" carrying integer values SpO_2 sent by the pulse oximeter, and two output actions "on!" and "off!" which control the PCA pump. There are two state variables a and v. Initially we assume $a = 0$ and $v = 100$. If an input action "toApp?SpO_2" is detected, the app updates a as 1 and sets v with the received SpO_2 value. If $v > 95$, meaning that the patient's vitals are not endangered, then the app outputs an "on!" signal to the PCA pump for allowing drug delivery; otherwise, an "off!" signal is sent to stop the PCA pump.

Figure 10a describes the specification for the PCA pump, which should be provided by the app developers, and Figure 10b shows the corresponding MIOA. Under this specification, if the pump is enabled by the "on?" command from the

[3] See [26] for an example of how continuous patient dynamics can be related to a discrete model.

[4] The corresponding MIOA has more than 200 states and thus is too large to be drawn here.

module *app*

input: toApp?**integer[0..100]**;
output: on!, off!;

a : [0..1] **init** 0;
v : [0..100] **init** 100;

[toApp?SpO_2] $(a = 0) \xrightarrow{\text{must}} (a' = 1) \wedge (v' = SpO_2)$;
[on!] $(a = 1) \wedge (v > 95) \xrightarrow{\text{must}} (a' = 0)$;
[off!] $(a = 1) \wedge (v \leq 95) \xrightarrow{\text{must}} (a' = 0)$;

endmodule

Fig. 9. App that controls the PCA pump

module *pumpSpec*

input: on?, off?, bolus?;
output: flowRate!**integer[0..2]**;
internal: τ;

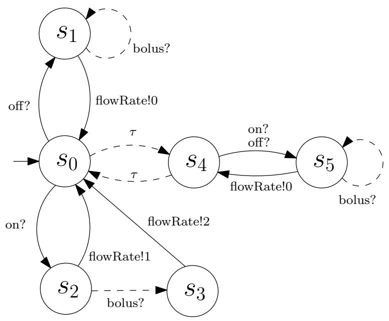
s : [0..5] **init** 0;

[off?] $(s = 0) \xrightarrow{\text{must}} (s' = 1)$;
[on?] $(s = 0) \xrightarrow{\text{must}} (s' = 2)$;
[on?] $(s = 4) \xrightarrow{\text{must}} (s' = 5)$;
[off?] $(s = 4) \xrightarrow{\text{must}} (s' = 5)$;
[flowRate!0] $(s = 1) \xrightarrow{\text{must}} (s' = 0)$;
[flowRate!0] $(s = 5) \xrightarrow{\text{must}} (s' = 4)$;
[flowRate!1] $(s = 2) \xrightarrow{\text{must}} (s' = 0)$;
[flowRate!2] $(s = 3) \xrightarrow{\text{must}} (s' = 0)$;

[bolus?] $(s = 1) \xrightarrow{\text{may}} (s' = 1)$;
[bolus?] $(s = 2) \xrightarrow{\text{may}} (s' = 3)$;
[bolus?] $(s = 5) \xrightarrow{\text{may}} (s' = 5)$;
[tau] $(s = 0) \xrightarrow{\text{may}} (s' = 4)$;
[tau] $(s = 4) \xrightarrow{\text{may}} (s' = 0)$;

endmodule

(a)



(b)

Fig. 10. The PCA pump specification and its corresponding MIOA

app, then it can deliver drug to the patient at a normal infusion rate "flowRate!1"; however, if it is disabled by the "off?" command, then no drug will be delivered ("flowRate!0"). Some pump may allow receiving "bolus?" request from the patient, which are described as "may" transitions in the specification (dashed lines in Figure 10b). To avoid the overdose of drug, the "bolus?" request is only effective (*i.e.,* the pump delivers drug at a higher rate "flowRate!2") when the pump is enabled by the app. A pump may shut itself down due to various reasons and may also restore from the shutdown; the specification allow these behavior by describing them "may" transitions labelled with an internal action τ. When a pump is shutdown, it would not respond to any command from the app/patient and no drug will be delivered.

4.2 Analysis

Figure 2 shows three PCA pump devices with different functionality. Device (a) has the basic functions of receiving commands from the app and delivering drug to the patient accordingly. Device (b) has the additional function of adjusting the drug infusion rate based on patients' bolus requests. Device (c) is the most sophisticated equipment: apart from the functions of receiving app commands (bolus requests) and delivering drug, it can detect air bubbles in the infusion line and issue alarm, and also protect itself from overheating; if any of these hazards is detected, the pump will shutdown automatically until the device is restored.

Our experiments indicate that devices (a) and (c) meet the specification requirements, because their corresponding MIOAs (Figures 2a and 2c) are both weak refinements of the specification's MIOA (Figure 10b). However, device (b) does not meet the specification: the path

$$r_0 \xrightarrow{\text{off?}} r_1 \xrightarrow{\text{bolus?}} r_3 \xrightarrow{\text{flowRate!2}} r_0$$

in the device's MIOA (Figure 2b) does not have a mapping (allowed path) in the specification's MIOA.

We verified the safety property "patient's SpO_2 level should always be above 94" on the PCA systems composed using these three devices. It is not surprising to find that both systems of devices (a) and (c) satisfy the safety property, but the system of device (b) violates the property. The violation is due to the fact that device (b) always admits patients' bolus request even if the app has issued the "off" command.

5 Conclusion and Discussion

In this paper we have described an approach to specify on-demand systems and applied that approach to an example application from the medical domain. This approach uses a specification language with the semantics of MIOA. We showed that weak modal refinement can be used to check the compatibility between app requirements and device capabilites by proving that weak modal refinement preserves both safety and liveness properties. This enables medical systems developers to express complex medical device behavioral requirements, explicitly specify allowed variability and reason about the behavior of on-demand systems *a priori*.

While we provided a case study as a proof-of-concept for the approach, there are many open questions concerning the engineering, safety and application of on-demand medical systems in a critical care setting. First we note that there are several areas where our proposed specification language and associated semantics would need to be extended to make the approach more applicable to real medical systems. For example, the action labels in our language are only tagged with simple data-types. Often, medical device actions relate directly to an interaction with the physical world (*e.g.,* an infusion pump *infusing* a drug). It would be useful if action labels could also be tagged with *physical* types which would denote the physical interaction of the action. Furthermore, our approach only supports reasoning about discrete time systems. Real medical devices exhibit continuous time behavior, and the ability to capture real-time behavior is critical if we want to apply our approach to real medical systems. For example, the on-demand medical systems described in [3,26,18] all rely on 'timeout' behavior in the medical devices to guarantee system safety in the presence of inter-device communications failures. Additionally, many medical devices exhibit continuous behavior in terms of their interactions with the patient. For example, infusion pumps deliver drugs continuously according to 'trumpet' curves [28]. It is not known to what fidelity a specification will need to capture device behavior: Is continuous time plus discrete behavior enough, or is a fully hybrid specification required? The answer to this question will likely depend on the types of on-demand systems clinicians will want to employee.

Second, the approach described in this paper requires that medical devices comply with their specifications. In theory this seems reasonable, but in practice it may not be possible to have total confidence in a device's compliance with its specification. For example, a device may be non-compliant due to an uncaught systematic defect (*i.e.,* a design or implementation error), uncaught manufacturing errors, or unaccounted for environmental interference (*e.g.,* artificial light interfering with a physiologic sensor). Should the specification approach be extended to capture the possibility of these errors? If so, what types of errors and faults should be captured in the specification versus left to other risk management mechanisms?

Finally, medical devices can interact with each other indirectly through the patient (*e.g.,* a pulse-oximeter attached to a patient on supplementary oxygen will sometimes give abnormally high readings). Should on-demand systems check for these types of interactions automatically? Or should we rely on the medical caregiver to determine which combinations of medical devices are appropriate to use in a given situation? The answer is not clear; though at first it may appear that automatic interaction checking could improve the safety of the system, in practice doing thisis very difficult due to the fact we currently lack a detailed understanding of human physiology. This in turn can result in overly conservative or sensitive checks. For example, modern computerized physician order entry (CPOE) systems automatically check if a doctor prescribes drugs that *may* interact adversely. In many cases hospitals have worked with vendors to disable these checks because these checks are too conservative (*i.e.,* generate an overwhelming number of alerts) and the caregivers are competent enough to know how the drugs will interact and whether or not the risk is justified for a particular patient [21]. The answer to this question will depend on the complexity of on-demand medical systems, our level of understanding of patient physiology, and the fidelity of care those system are intended to deliver.

References

1. Alur, R., Henzinger, T.A.: Reactive modules. Formal Methods in System Design 15(1), 7–48 (1999)
2. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: 20 years of modal and mixed specifications. European Association for Theoretical Computer Science. Bulletin (95) (2008)
3. Arney, D., Goldman, J.M., Whitehead, S.F., Lee, I.: Synchronizing an x-ray and anesthesia machine ventilator: A medical device interoperability case study (2009)
4. Arney, D., Jetley, R., Jones, P., Lee, I., Sokolsky, O.: Formal methods based development of a pca infusion pump reference model: Generic infusion pump (gip) project. In: Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability, HCMDSS-MDPnP, pp. 23–33. IEEE (2007)
5. Medical devices and medical systems - essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ice), `http://enterprise.astm.org/filtrexx40.cgi?+REDLINE_PAGES/F2761.htm`
6. Bauer, S.S., Mayer, P., Legay, A.: Mio workbench: A tool for compositional design with modal input/output interfaces. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 418–421. Springer, Heidelberg (2011)
7. Bauer, S.S., Mayer, P., Schroeder, A., Hennicker, R.: On weak modal compatibility, refinement, and the MIO workbench. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 175–189. Springer, Heidelberg (2010)
8. Carroll, R., Cnossen, R., Schnell, M., Simons, D.: Continua: An interoperable personal healthcare ecosystem. IEEE Pervasive Computing 6(4), 90–94 (2007)
9. Clarke, M., Bogia, D., Hassing, K., Steubesand, L., Chan, T., Ayyagari, D.: Developing a standard for personal health devices based on 11073. In: 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS 2007, pp. 6174–6176 (2007)
10. Dolin, R.H., Alschuler, L., Boyer, S., Beebe, C., Behlen, F.M., Biron, P.V., Shvo, A.S.: Hl7 clinical document architecture, release 2. Journal of the American Medical Informatics Association 13(1), 30–39 (2006)
11. Hatcliff, J., King, A., Lee, I., Macdonald, A., Fernando, A., Robkin, M., Vasserman, E., Weininger, S., Goldman, J.M.: Rationale and architecture principles for medical application platforms. In: IEEE/ACM Third International Conference on Cyber-Physical Systems, ICCPS 2012, pp. 3–12. IEEE Computer Society, Washington, DC (2012), `http://dx.doi.org/10.1109/ICCPS.2012.9`
12. Hatcliff, J., Vasserman, E., Weininger, S., Goldman, J.: An overview of regulatory and trust issues for the integrated clinical environment. In: Proceedings of HCMDSS 2011 (2011)
13. Hicks, R.W., Sikirica, V., Nelson, W., Schein, J.R., Cousins, D.D.: Medication errors involving patient-controlled analgesia. American Journal of Health-System Pharmacy 65(5), 429–440 (2008)
14. Hoare, C.A.R.: Communicating sequential processes. Prentice-Hall, Inc., Upper Saddle River (1985)
15. Hudcova, J., McNicol, E., Quah, C., Lau, J., Carr, D.B.: Patient controlled intravenous opioid analgesia versus conventional opioid analgesia for postoperative pain control: A quantitative systematic review. Acute Pain 7(3), 115–132 (2005)
16. Iso/ieee 11073 committee, `http://standards.ieee.org/findstds/standard/11073-10103-2012.html`

17. Joint Commission: Sentinel event alert issue 33: Patient controlled analgesia by proxy (December 2004), `http://www.jointcommission.org/sentinelevents/sentineleventalert/`
18. Kim, C., Sun, M., Mohan, S., Yun, H., Sha, L., Abdelzaher, T.F.: A framework for the safe interoperability of medical devices in the presence of network failures. In: Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems, pp. 149–158. ACM (2010)
19. King, A., Arney, D., Lee, I., Sokolsky, O., Hatcliff, J., Procter, S.: Prototyping closed loop physiologic control with the medical device coordination framework. In: Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care, pp. 1–11. ACM (2010)
20. King, A., Procter, S., Andresen, D., Hatcliff, J., Warren, S., Spees, W., Jetley, R., Jones, P., Weininger, S.: An open test bed for medical device integration and coordination. In: Proceedings of the 31st International Conference on Software Engineering (2009)
21. Kuperman, G.J., Bobb, A., Payne, T.H., Avery, A.J., Gandhi, T.K., Burns, G., Classen, D.C., Bates, D.W.: Medication-related clinical decision support in computerized provider order entry systems: A review. Journal of the American Medical Informatics Association 14(1), 29–40 (2007), `http://www.sciencedirect.com/science/article/pii/S106750270600209X`
22. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
23. Larsen, K.G., Thomsen, B.: A modal process logic. In: Proceedings of the Third Annual Symposium on Logic in Computer Science, LICS 1988, pp. 203–210 (1988)
24. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O automata for interface and product line theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007)
25. Medical device "plug-and-play" interoperability program (2008), `http://mdpnp.org/`
26. Pajic, M., Mangharam, R., Sokolsky, O., Arney, D., Goldman, J., Lee, I.: Model-driven safety analysis of closed-loop medical systems. IEEE Transactions on Industrial Informatics (2013)
27. Paul, J.E., sawhney, M., Beattie, W.S., McLean, R.F.: Critical incidents amongst 10033 acute pain patients. Canadian Journal of Anesthesiology 51, A22 (2004)
28. Voss, G.I., Butterfield, R.D.: 27.1 performance criteria for intravenous infusion devices. Clinical Engineering, 415 (2003)