A Medical Device Safety Supervision over Wireless

Cheolgi Kim[1], Mu Sun[2], Heechul Yun[3], and Lui Sha[4]

[1] University of Illinois, Urbana, IL 61801, USA cheolgi@illinois.edu
[2] University of Illinois, Urbana, IL 61801, USA musun@illinois.edu
[3] University of Illinois, Urbana, IL 61801, USA heechul@illinois.edu
[4] University of Illinois, Urbana, IL 61801, USA lrs@illinois.edu

Summary. Interoperability of medical devices is a growing need in modern healthcare systems, not just for convenience, but also to preclude potential human errors during medical procedures. Caregivers, as end users, strongly prefer the use of wireless networks for such interconnections between clinical devices due to its seamless connectivity and ease of use/maintenance. In [KSM+10], we introduced a Network-Aware Safety Supervisior framework to integrate medical devices into clinical supervisory systems using finite state machine (FSM). In this paper, we simplify FSM into Boolean Logic to minimize safety logic overhead and introduce a generic method, called pre-verified safety control (PVSC) framework to integrate medical devices into clinical management systems using wireless technologies that have their safety properties verified in a formal manner. Our method provides *(i)* a PVSC safety layer that automatically generates the safety engine to guarantee given safety requirements and *(ii)* an abstracted application development environment so that applications can be developed independent of underlying complications of wireless communication. To mitigate negative effects due to packet losses, the PVSC framework employs a pipelined "pre-planning" of the device controls. The key motivation of the work in this paper is to preserve safety and the application development environment, as is, even after adding unreliable communication media, such as wireless, along with a pre-planning mechanism.

1 Introduction

The medical device supervision paradigm recently has a significant transformation in patient monitoring and administration. The importance of medication automation has been increased because of the high cost of health care and high rate of medical accidents. a recent report [Gra07] suggested that of the 284,798 deaths that occurred among patients who developed one or more patient safety incidents between 2003 – 2005, 247,662 were potentially preventable. Hence, there is a need to increase the use of devices that can *automaticlly* check for safety constraints.

J. Hong et al. (eds.), *Reliable and Autonomous Computational Science*,
Autonomic Systems, DOI 10.1007/978-3-0348-0031-0_2,

However, automation of medication has another potential problem. The reliability of automation relies on software. Because of its complexity, making software bug free is known to be very challenging work. Since the reliability in such medical system is directly related to human life, the bug in such system is even more critical. Thereby, most medical devices today operate in a stand-alone fashion in the era of communication to avoid complexity.

Therefore, we need safety assurance software framework to connect medical devices over networks to decouple safety from the medical devices. Recent initiatives have been launched to increase *interoperability* of medical devices to reduce medical accidents caused by human errors [Hig09]. One such initiative is the work on the *Integrated Clinical Environment* (ICE) draft standard [Gol08]. The ICE standard aims to provide integration of data and devices to enable *real-time* decision support and safety interlocks, thus ensuring patient safety.

Wireless technology has recently been proposed, and used, as a communication mechanism to implement medical device interoperability [BH08]. Most of the developments though, have been in the area of medical sensors [cod] (pulse oximeters, EKG monitors, *etc.*) and also in critical systems such as infusion pumps and pacemakers. The latter set of devices, though comprising of safety-critical aspects, mostly limit themselves to using wireless technology for transmission of non-critical data and for offline analysis.

The use of wireless technology in real-time safety-critical applications is complicated by various factors. First, the fact that wireless technologies cannot provide *guarantees* that transmitted packets will actually be delivered, *on time*. If some control messages between two devices are lost en route then it could endanger the life of a patient. Second, the *safety* of the physical system must be guaranteed in a rigorous manner *in spite* of the failure of the wireless communication mechanism. Third, when using wireless technology in conjunction with medical devices, the quality of medical service must be good enough; a few packet losses should not incur a discontinuity of the medical procedure. Finally, the framework should incorporate *abstraction* to aid in the process of the development of clinical applications. The complexity of using wireless technology as the underlying communication mechanism should not propagate to the upper layers where clinical control applications are being designed and developed. If the actual communication mechanism is not abstracted away then system architects cannot concentrate on the design of the clinical application logic, thus degrading the overall system.

To address these issues, we present a new *Pre-Verified Safety Control* (PVSC) framework. We proposed such a framework called NASS framework [KSM+10] in the form of finite state machine (FSM), but PVSC simplifies the approach by employing Boolean logic instead of FSM. PVSC proposes a reliable control messaging framework and a system-wide protocol for wireless communication that can be used in conjunction with *integrated clinical control systems* (*ICCSs*), where the medical device controls are performed through the closed loop, but the safety of the patient can be secured in an

open loop situation. The application logic have no knowledge of the complications of using wireless communication handled by the framework. The basic idea is to deliver messages that set the pre-verified safety controls for a long period of time in the future. Hence, even if messages are lost, the system will be in a predictable state. Furthermore, verifying safety under the general asynchronous environment of wireless communication is non-trivial. An unverifiable system cannot be considered a safe one. Thus, we simplify the analysis by imposing a synchronous control messaging system structure on the system. Furthermore, we describe the standard preverified software architecture blocks that can be automatically generated and deployed for the medical safety. To the best of our knowledge, this is the first such framework that systematically provides the ability to use wireless communication that *preserves safety* and is grounded in *formal guarantees*. The main contributions of the PVSC framework are:

1. *Preserving Safety*: Safety requirements written for an *ideal* clinical environment (with robust communication scheme) are *automatically* transformed to the safety validation engine for PVSC framework in wireless networks, thus preserving safety. These are explained in further detail in Section 4.
2. *Abstraction*: Clinical control applications built on top of the PVSC framework can be developed independent of the underlying complications of wireless communication. The PVSC framework hides these complications from the application logic, as explained in Section 3.

The degradation of the quality of service caused by packet drops is unavoidable, but, the PVSC framework is able to minimize such losses. We are also able to achieve a decoupling between the safety and the application logic. Hence, the clinical application development life-cycle becomes shorter while still preserving the safety and real-time constraints.

This paper is organized as follows. Section 2 presents the background, benefits and overview of PVSC using an example. Section 3 shows how to hide the complications of wireless communications from the application logic. The safety preserving method by automatic generation of safety validation engine is presented in Section 4. Section 5 briefly introduces our prototype system, and Section 6 presents the related work while Section 7 concludes the paper.

2 Airway-Laser Interlock Example

Laser cutters are often used in operating rooms these days. During surgery to the airway[5], if such a laser is used, then there is the potential for fatal burns to the patient if the concentration of oxygen in the airway is high. Hence,

[5]The passage by which air reaches a person's lung

during such surgery, it is recommended that the oxygen pump connected to the ventilator[6] be blocked. Unfortunately, this guideline is sometimes overlooked, and serious, perhaps fatal, injuries occur. The 2006 American Society of Anesthesiologists meeting estimated that roughly 100 such fires occur each year in U.S. hospitals, causing roughly 20 serious injuries and one or two deaths [Mar07].

To prevent such problems, an *airway-laser interlock* must be enforced by use of an automated ICCS that maintains mutual exclusion between the activities of a ventilator and an airway laser. Hence, whenever an 'airway laser' is used, *(1)* the air path from the oxygen concentrate at the ventilator must be minimized and *(2)* the oxygen proportion in the airway must be guaranteed to be lower than a predetermined safety threshold. On the other hand, the safety of the patient should not be compromised due to the reduced oxygen flow. A pulse oximeter[7] is used to measure the SpO_2 (Saturated Peripheral O_2) during this surgical procedure to track the amount of oxygen in the blood. If the SpO_2 readings are below a predefined "watch level," then the oximeter produces an alarm. The surgeon can then stop using the laser instrument and restore the flow of oxygen between the oxygen concentrate and the ventilator, *manually*. If the medical personnel ignore such alarms and the SpO_2 reaches the lower threshold and is in danger of dropping further, then an automated controller should disable the use of the laser and restart the flow of oxygen again. Hence, the safety requirements of the airway laser interlock are:

S1 The airway laser and the oxygen concentrate should not be activated together.
S2 The airway laser should not be activated if the proportion of oxygen in the airway is higher than a predetermined threshold (*e.g.* 25% [Gol09]).
S3 If patient's SpO_2 is lower than the lower threshold value, then the oxygen concentrate must be activated through a ventilator.

2.1 PVSC Framework Overview

Let us recall the airway-laser example to better understand the control flow of the PVSC framework. The operational sequence of the system in a *round* is as follows: ① at the end of the previous round acknowledgement packets from participating devices (a PCA pump, an oximeter and a capnometer) have been received by the ICCS manager and delivered up to the virtual devices. These virtual devices update their state information based on the information contained in the packets. According to our measurement results, ② the *patient state estimator* is not only able to gauge the patient's current states but also μ future states of control rounds. The patient state estimator generates two kinds of future patient states: strict worst case states for the safety layer, and

[6] An appliance for artificial respiration
[7] An instrument measuring the oxygenated hemoglobin in the blood

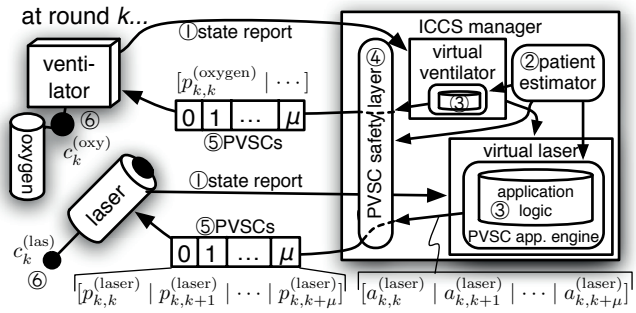

Fig. 1. Control flow example of airway laser interlock

realistic states for the application logic. How to generate them is the out of scope of this paper.

Then, ③ each virtual actuator, virtual airway laser in this example, prepares *application-planned controls* for each rounds k (current round) to $k+\mu$ where μ denotes the number of future rounds for which the framework provides back-up plans, *i.e.*, $[a_{k,k}^{(\text{laser})}, a_{k,k+1}^{(\text{laser})}, \cdots a_{k,k+\mu}^{(\text{laser})}]$. $a_{k,l}^{(\text{laser})}$ denotes an *application-planned control* for the usage allowance of the airway laser for the future (or current) round k generated at round l. The number of future rounds decided at each round, μ, is supposed to be given by system designer based on the system communication characteristics and the system requirements. *Application-planned controls* are generated by the PVSC application engine that exploits the application logic in which the airway laser's logical operations are implemented. Recall that application logic is not aware of the planning mechanism of the framework. It is developed so as to operate round-by-round in monotonic time sequence. However, in order to provide back-up plans, the PVSC application engine has to virtually perform setting back a clock at each round. How the PVSC application engine provides an abstracted view of the system to the application logic is discussed in Section 3.

Once such *application-planned controls* are generated ④ the PVSC safety layer verifies their safety properties. The safety-checked sequence of controls are called *pre-verified safety controls*, denoted by $[p_{k,k}^{(\text{laser})}, p_{k,k+1}^{(\text{laser})}, \cdots, p_{k,k+\mu}^{(\text{laser})}]$ to represent the usage allowance of the airway laser. If the application-planned controls satisfy the safety invariants they are delivered to the device without modification by the safety layer. Otherwise, the layer modifies the necessary control value(s) to ensure safety. The details of how to produce the PVSC safety layer from the invariants and how to enforce safety into the application-planned controls are presented in Section 4. A command packet delivers these pre-verified safety controls to a device⑤, the airway-laser or the ventilator in this example. Then, ⑥ the device executes these safety controls thus maximizing the effectiveness of the system for each round. The selection mechanism is explained in Section 3 along with the PVSC application engine.

3 Provision of Abstraction to Application Logic

When the PVSC application engine produces application-planned controls exploiting the application logic, there are two main issues to be addressed. The first one is to hide the planning mechanism from the application logic.

To support the planning mechanism, the application engine has to set back a clock at the beginning of each round. When the PVSC application engine generates back-up plans at round k, it produces the plans for the rounds from k to $(k+\mu)$. At the next round, the plan generation starts from round $(k+1)$, setting back a clock from round $(k+\mu)$ to round $(k+1)$. However, it should be hidden from the application logic for the simpler development environment.

To enable setting back a clock, the PVSC application engine takes a snapshot of the active object of the application logic whenever a control for a round is generated. At round k, the engine generates $a_{k,k}^{(\text{laser})}$ as the first application-planned control. Then, it takes a snapshot of the running algorithm, and requests the next round control, $a_{k,k+1}^{(\text{laser})}$ by running application logic again to virtually have the next round. The snapshot of the algorithm after $a_{k,k+1}^{(\text{laser})}$ generation is taken, too. This iteration is repeated until $a_{k,k+\mu}^{(\text{laser})}$ is generated. Because the engine has the full snapshots of the application logic, setting back a clock is obviously realized by recalling an appropriate snapshot. Each snapshot is kept until it becomes useless (i.e. after projected execution round is expired).

Another issue is to ensure *a single flow of the logic* executed by a device. Suppose that a medicine infusion is controlled by an ICCS. The infusion pump always applies the most recently received pre-verified safety control for the infusion, and the application logic proceeds based on the most up-to-date snapshot of the system. Infusion is supposed to be controlled by occasional pumping events rather than a continuous infusion rate. At round 1, the application logic generates application-planned controls, planning the next infusion of the medicine at round 6. This plan is repeated until round 4 by the logic. However, the application logic changes its mind at round 5 to make the next infusion of the medicine at round 7 instead of round 6. Thus, all application-planned controls generated from round 6 are changed to have this new plan. However, the infusion pump infused the medicine at round 6 because it has not received the updated decision due to the series of packet losses. Then, it finally received the new plan at round 7, and infused the medicine again according to the updated control. Consequently, the infusion was doubled unintentionally because the pump followed the application-planned controls without checking if the series of decisions are from *a single flow of the logic*.

Algorithm 3 and Algorithm 4 are the pseudo codes for the PVSC application engine and the device control selection to fulfill the single flow requirement, respectively. An application-planned control, a pre-verified safety control and an executed control are denoted by $a_{j,k}^{(\cdot)}$, $p_{j,k}^{(\cdot)}$ and $c_k^{(\cdot)}$, respectively, where j is the plan generation round, k is (projected) execution round

Algorithm 3 Pseudo code of PVSC application engine

Require: Round k has just started

Require: $p^{(\cdot)}_{x,k-1}$ is the pre-validated safety control executed by device at round $(k-1)$.

Require: $s^{(\cdot)}_{x,y}$ is application logic snapshot for round-y execution, generated at round x.

Ensure: application-planned controls, $[a^{(\cdot)}_{k,k}, \cdots, a^{(\cdot)}_{k,k+\mu}]$, are ready for PVSC safety layer

if ack. packet at round $(k-1)$ is delivered **then**
  $s := s^{(\cdot)}_{x,k-1}$
else $s := s^{(\cdot)}_{k-1,k-1}$
end if
$a^{(\cdot)}_{k,k} := s$.nextRound();
$s^{(\text{mor})}_{k,k} := s$.snapshot();
for $i = 1$ to μ **do**
  $a^{(\cdot)}_{k,k+i} := s^{(\cdot)}_{k,k+i-1}$.nextRound();
  $s^{(\cdot)}_{k,k+i} := s^{(\cdot)}_{k,k+i-1}$.snapshot();
end for

Algorithm 4 Pseudo code in a device for PSVC selection

Require: Timer for command packet is expired at round k, $p^{(\cdot)}_{x,k-1}$ was executed at last round

Ensure: Ack. packet returns x to notify $p^{(\cdot)}_{x,k}$ is executed.

if command packet is delivered and $p^{(\cdot)}_{k,k}$ is generated from the flow of $p^{(\cdot)}_{x,k-1}$ **then**
  execute $p^{(\cdot)}_{k,k}$, which is $c^{(\cdot)}_k := p^{(\cdot)}_{k,k}$
  $x := k$
else execute $p^{(\cdot)}_{x,k}$, which is $c^{(\cdot)}_k := p^{(\cdot)}_{x,k}$
end if

and $(\cdot)$ is reserved for a control identification. For more intuitive explanation, let us trace an example execution flow presented in Fig. 2. Each sequence of application-planned controls generated at the same round is grouped by a looped curve, and the controls for the same round executions are vertically aligned by rounds. μ is two, and all controls satisfy the safety requirements, such that $p^{(\cdot)}_{j,k} = a^{(\cdot)}_{j,k}$. The bold circled application-planned controls are the controls that the device ran at the execution round.

At round 1, the device executed $a^{(\cdot)}_{1,1}$, after the successful reception of command packet. The acknowledgement packet of round 1 notifies the execution of $a^{(\cdot)}_{1,1}$ back to the PVSC application engine. Thus at round 2, the PVSC application engine generates the sequence of application-planned controls from

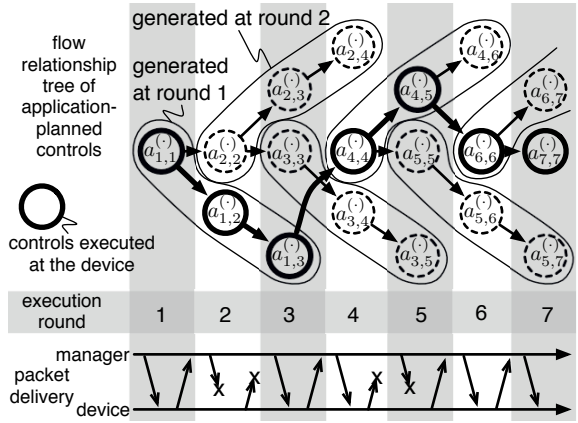

Fig. 2. An example behavior of the PVSC application engine and the related device

the snapshot of $a^{(\cdot)}_{1,1}$. The command packet delivery was failed, so the device executed $a^{(\cdot)}_{1,2}$. Unfortunately, this consequence of the command packet failure also failed to be notified back through the acknowledgement packet. As a result, the PVSC engine was not aware of the application-planned control which the device had executed at round 2. As a result, $a^{(\cdot)}_{3,3}$ was generated from $a^{(\cdot)}_{2,2}$ as a default action, which does not match the real executed control, $a^{(\cdot)}_{1,2}$. Because of this information gap between the manager and the device, the device reject applying the up-to-date control, $a^{(\cdot)}_{3,3}$ and used the old one, $a^{(\cdot)}_{1,3}$ for the single flow requirement. This circumstance was informed back to the manager through the acknowledgement. So the manager produced application-planned control from $a^{(\cdot)}_{1,3}$ for the next round. Now, the information gap between the manager and the device was filled up. By having this sequence of operations, a single flow of application logic is fulfilled over wireless communications.

4 Auto Generation of Safety Verifier

To understand the proposed verifier, readers are referred to [KSM+10] for detail of NASS framework. In this section, FSM of NASS framework is transformed into Boolean logic. The section describes and justifies the safety verifier.

Control information, planned by application designers and generated by the NASS application engine, flows down to the NASS safety layer for the safety assurance. As mentioned previously, the safety requirements are expressed as Boolean-invariants. The NASS safety layer that guarantees the safety of the controls, regardless of packet losses or network disconnections, is *automatically* generated from these invariants.

Consider the safety requirements from **S1**–**S3** given in Section 2. These requirements can be formally expressed by safety invariants as

$$c_k^{(\text{laser})} \wedge c_k^{(\text{oxygen})} = \mathit{false} \tag{1}$$

$$c_k^{(\text{laser})} \Rightarrow \text{low}_k^{(\text{oxygen})} \tag{2}$$

$$\neg\text{safe}_k^{(\text{SpO}_2)} \Rightarrow c_k^{(\text{oxygen})} \tag{3}$$

where $c_k^{(\text{laser})}$ is the allowance of the laser activation $c_k^{(\text{oxygen})}$ is the enabling of the oxygen concentrate pathway $\text{low}_k^{(\text{oxygen})}$ is the Boolean indicator if the concentration of oxygen in the airway is lower than the given threshold, and $\text{safe}_k^{(\text{SpO}_2)}$ is the Boolean indicator if patient's SpO_2 level is higher than the warning level at round k Eq. (1), (2) and (3) represent **S1**, **S2** and **S3**, respectively.

4.1 Safety Invariants

The safety invariants are composed of a set of control propositions A and a set of environmental event propositions E. In the airway-laser example, we have $c_k^{(\text{oxygen})}, c_k^{(\text{laser})} \in A$ and $\text{low}_k^{(\text{oxygen})}, \text{safe}_k^{(\text{SpO}_2)} \in E$. We abstract Boolean propositions from continuous values; *e.g.* $\text{low}_k^{(\text{oxygen})}$ and $\text{safe}_k^{(\text{SpO}_2)}$ are abstracted from continuous values of oxygen concentration, and measured SpO_2.

It is well known from logic that a set of Boolean sentences can be transformed into one sentence in conjunctive normal form (CNF). For example, CNF of Eq. (1)–(3) is given by

$$\begin{aligned}\phi_{\text{laser}}(A,E) \triangleq &\left(\neg c_k^{(\text{laser})} \vee \neg c_k^{(\text{oxygen})}\right) \\ &\wedge \left(\neg c_k^{(\text{laser})} \vee \text{low}_k^{(\text{oxygen})}\right) \wedge \left(c_k^{(\text{oxygen})} \vee \text{safe}_k^{(\text{SpO}_2)}\right)\end{aligned} \tag{4}$$

Let ϕ denote the sentence in CNF transformed from safety invariants. A sentence ϕ in CNF can be represented by a conjunction set (Φ) having disjunctions as the elements. The safety sentence ϕ in CNF, and its conjunction set Φ has the relationship of

$$\phi(A,E) \Leftrightarrow \bigwedge_{\theta \in \Phi(A,E)} \theta(A,E).$$

We have the conjunction powerset of ϕ_{laser} given in Eq. (4):

$$\begin{aligned}\Phi_{\text{laser}}(A,E) = \Big\{&\neg c_k^{(\text{laser})} \vee \neg c_k^{(\text{oxygen})}, \neg c_k^{(\text{laser})} \vee \text{low}_k^{(\text{oxygen})}, \\ &c_k^{(\text{oxygen})} \vee \text{safe}_k^{(\text{SpO}_2)}\Big\}\end{aligned} \tag{5}$$

4.2 Finding an Open-Loop Safe State

The first step of the safety verifier generation is to find the open-loop safe states of the system. If a system does not have an open-loop safe state, the

safety cannot be guaranteed when the network of the system is totally disconnected. Finding open-loop safe states is finding the *safety bias* of each control. In the airway laser example, the safety bias of airway laser $c_k^{(\text{laser})}$ is *false* because no disjunction in Φ_{laser} has $c_k^{(\text{laser})}$ from Eq. (5), which means no disjunction becomes more restrictive by the assignment of $c_k^{(\text{laser})} = \mathit{false}$. However, the oxygen concentrate cannot have safety bias from conjunction set Φ_{laser} because disjunction $c_k^{(\text{oxygen})} \vee \text{safe}_k^{(\text{SpO}_2)}$ has $c_k^{(\text{oxygen})}$ while another disjunction $\neg c_k^{(\text{laser})} \vee \neg c_k^{(\text{oxygen})}$ has $\neg c_k^{(\text{oxygen})}$. Suppose that we have $c_k^{(\text{laser})} = \mathit{false}$. Then, disjunction $\neg c_k^{(\text{laser})} \vee \neg c_k^{(\text{oxygen})}$ is cleared by $c_k^{(\text{laser})}$. Then, the *conditional safety bias* of $c_k^{(\text{oxygen})}$ becomes *true*, depending on the assignment of $c_k^{(\text{laser})} = \mathit{false}$. We call it a *conditional safety bias* and $c_k^{(\text{oxygen})}$ is called to be *safety-biased depending on* $c_k^{(\text{laser})}$.

If one control proposition, $c_k^{(\text{a})}$ is safety-biased depending on another control proposition, $c_k^{(\text{b})}$, $c_k^{(\text{b})}$ must be decided first and then $c_k^{(\text{a})}$ should follow. Suppose that $c_k^{(\text{laser})}$ is decided first to the contrary of the above claim. If $c_k^{(\text{laser})}$ is assigned to be *false*, $c_k^{(\text{oxygen})}$'s safety bias is in trouble. no control decision of $c_k^{(\text{oxygen})}$ guarantees the safety of ϕ_{laser}. Thus, to decide $c_k^{(\text{laser})}$ first, it has to look ahead the disjunctions related to the depending control $c_k^{(\text{oxygen})}$ in run time. Otherwise, $c_k^{(\text{laser})}$ must be assigned *false* for the global system safety, which is a too restrictive choice. This looking ahead is not required if $c_k^{(\text{oxygen})}$ is decided first according to the order of the dependency.

Algorithm 5 presents the algorithm finding the open-loop safe state, $s_{\text{safe}}^{(\alpha)}$ of each control proposition, which is the safety-bias, and the control decision order by finding the safety-bias dependencies. L is the list of control propositions in the control decision order as an output of the algorithm. Because, the dependencies between control propositions does not generate a single decision order of controls, we recommend the designers to input preferred control decision order manually.

The **repeat-until** loop finds the safety-bias of controls, and their dependencies. If there is circular dependencies among control propositions, their safety-biases cannot be found and the loop is terminated. An example of a circular dependency is:

$$\phi_{\text{circular}} \triangleq (c_k^{(1)} \vee \neg c_k^{(2)} \vee e_k^{(1)}) \wedge (\neg c_k^{(1)} \vee c_k^{(2)} \vee e_k^{(1)}) \tag{6}$$

where $c_k^{(1)}$ and $c_k^{(2)}$ are control propositions, and $e_k^{(1)}$ is a environment proposition. In this case, the algorithm can declare a failure, or optionally the user can enforce the algorithm to find any possible open-loop safe state by having exhaustive search. By the enforcement, the open-loop safe states for ϕ_{circular} will be found to be either $c_k^{(1)} = \mathit{true}, c_k^{(2)} = \mathit{true}$ or $c_k^{(1)} = \mathit{false}, c_k^{(2)} = \mathit{false}$. However, we do not recommend the enforced search for the open-loop safe state. When there are circular dependencies among control variables, the con-

Algorithm 5 Find_Safety_Bias_Relationship(Φ, A)

Ensure: L(out): list of controls in control decision order
Ensure: $\Phi \neq \varnothing$ or procedure fails
  repeat
    for all $\alpha \in A$ { A in preferred decision order} **do**
      $\Phi_{\text{old}} = \Phi$
      $\Theta_\alpha := \{\theta \in \Phi \mid \alpha \in \theta\}; \Theta_{\neg\alpha} := \{\theta \in \Phi \mid \neg\alpha \in \theta\}$
      if $\Theta_{\neg\alpha} = \varnothing$ **then**
        $s_{\text{safe}}^{(\alpha)} := \mathit{true}$
        $L := [\alpha; L]; A := A \setminus \{\alpha\}; \Phi := \Phi \setminus \Theta_\alpha$
      else if $\Theta_\alpha = \varnothing$ **then**
        $s_{\text{safe}}^{(\alpha)} := \mathit{false}$
        $L := [\alpha; L]; A := A \setminus \{\alpha\}; \Phi := \Phi \setminus \Theta_{\neg\alpha}$
      end if
    end for
  until $\Phi = \Phi_{\text{old}}$
  if $\Phi \neq \varnothing$ **then**
    Declare failure or
    Exhaustively search for the open-loop safe states
  end if

trol decision order has an inverse dependency. As we have seen in the example of inverse dependency of the safety-bias between $c_k^{(\text{laser})}$ and $c_k^{(\text{oxygen})}$, the inversion requires traverse of inverse dependencies, or pessimistic decision of the controls. Because the dependency traversal is NP-hard because analyzing all disjunctions related to the circular dependency is identical to the SAT problem. To have practical run-time algorithm, our safety verifier takes the pessimism. (Recall that $c_k^{(\text{laser})} = \mathit{false}$ is the solution of pessimism for inverse dependency.) Because this pessimism may not be what the user wants, we recommend 'declaring failure' when the loop has not found the full safety biases. If a failure occur, the user can review the invariants and fix them. In our experiences, all medical interlocks having open-loop safe states have successfully completed with this algorithm.

4.3 Safety Engine Generation

When the NASS safety layer generates pre-verified safety controls, it has to ensure the safety in all possible scenarios. The generated controls at each round can be executed altogether by the connected devices. But sometimes, the controls that are generated at the various rounds can be executed by the devices at a round. Therefore, the safety verifier of the NASS safety layer has to ensure that the generated pre-verified safety controls be always safe when they are mixed with previously generated pre-verified safety controls that are possibly effective in the devices. If this joined safety is ensured with the

previously generated controls. By mathematical induction to the execution round, the safety of NASS execution is ensured.[8]

From Algorithm 5, the controls have the decision precedence. For the convenience of the analysis, $c_k^{(d)}$, $p_{j,k}^{(d)}$ and $a_{j,k}^{(d)}$ denotes the executed control, pre-verified safety control and application-planned control of the control decided in d-th precedence in the safety engine. *E.g.* $c_k^{(1)} \triangleq c_k^{(\text{oxygen})}$ and $c_k^{(2)} \triangleq c_k^{(\text{laser})}$. The number of devices is denoted by n.

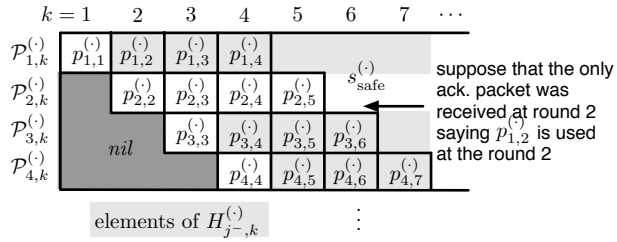

Fig. 3. Semantic of $\mathcal{P}_{j,k}^{(d)}$ and $H_{j^-,k}^{(d)}$

The pre-verified safety controls for a control d, $p_{j,k}^{(d)}$, is explicitly defined only when $k - \mu \leq j \leq k$. We extend this values of $p_{j,k}^{(d)}$ to be defined over every combination of j and k with $\mathcal{P}_{j,k}^{(d)}$, such that

$$\mathcal{P}_{j,k}^{(d)} \triangleq \begin{cases} nil & \text{if } j > k, \\ p_{j,k}^{(\text{d})} & \text{if } j \leq k \leq j + \mu, \\ s_{\text{safe}}^{(\text{d})} & \text{if } k > j + \mu \end{cases} \tag{7}$$

where *nil* is the empty command as depicted in Fig. 3.

Recall that the acknowledgement packet has the round information of the most recently executed NASS. See the example in Fig. 3. The manager received acknowledgement packet at round 2 saying that $p_{1,2}^{(d)}$ is executed by the device at round 2. Then, the latest effective generation round that the manager knows is round 1. Let β_j^d denote the latest effective generation round. Then, the set of the effective NASS at the beginning of round j, $H_{j^-,k}^{(d)}$ is bounded by

$$H_{j^-,k}^{(d)} \subset \bigcup_{i=\beta_{j-1}^{(d)}}^{j-1} \mathcal{P}_{i,k}^{(d)} \tag{8}$$

β_j^d for $j > 1$ in this example is 1. but acknowledgement delivery round is 2. Then we know that all NASS generated at round 2 is not effective because

[8] As the basis of induction, the safety of all rounds are ensured at the beginning of operations by all devices being in the open-loop safe states forever.

the device informed that it used NASS generated at round 1 even after round 2 command packet is delivered. From this intuition, we have

$$H^{(d)}_{j^-,k} = \mathcal{P}^{(d)}_{\beta^{(d)}_{j-1,k},k} \cup \bigcup_{i=\gamma^{(d)}_{j-1}+1}^{j-1} \mathcal{P}^{(d)}_{i,k} \tag{9}$$

where $\gamma^{(d)}_j$ is the most recent round at which an acknowledgement packet was received about control (d). While $H^{(d)}_{j^-,k}$ denotes the set of the still effective NASSs before the NASS generation at round j, $H^{(d)}_{j^+,k}$ denotes the set of the still effective NASSs after the generation, such that

$$H^{(d)}_{j^+,k} = H^{(d)}_{j^-,k} \cup \mathcal{P}^{(d)}_{j,k} \tag{10}$$

Consequently, the safety verifier must ensure that any control proposition combination of $\left(H^{(1)}_{j^+,k} \times H^{(2)}_{j^+,k} \times \cdots H^{(n)}_{j^+,k}\right)$ satisfies safety statement ϕ. Let us define another statement, $\hat{\phi}$ to check if the whole combinations satisfy such that

$$\hat{\phi}\left(\left(S^{(d_1)}, S^{(d_2)}, \cdots, S^{(d_n)}\right), E\right) \Leftrightarrow \left(\forall x \in \left(S^{(d_1)} \times S^{(d_2)} \times \cdots \times S^{(d_n)}\right), \quad \phi(x, E)\right) \tag{11}$$

where $S^{(d_d)}$ denotes a non-empty set of Boolean values that can be applied for control (d). $\hat{\phi}$ receives the tuple of sets of Boolean values rather than the tuple of Boolean values, for the control propositions. $\hat{\phi}$ is defined by

$$\hat{\phi}\left(\left(S^{(d_1)}, S^{(d_2)}, \cdots, S^{(d_n)}\right), E\right) \triangleq \bigwedge_{\theta \in \phi} \mathit{worst}\left(\theta, \left(S^{(d_1)}, S^{(d_2)}, \cdots, S^{(d_n)}\right), E\right) \tag{12}$$

where

$$\begin{aligned} &\mathit{worst}\left(\theta, \left(S^{(d_1)}, S^{(d_2)}, \cdots, S^{(d_n)}\right), E\right) \triangleq \theta(\cdot, E) \\ &\vee \left(\left(\forall a^{(d)} \in \theta, \mathit{false} \notin S^{(d_d)}\right) \wedge \left(\forall \neg a^{(d)} \in \theta, \mathit{true} \notin S^{(d_d)}\right)\right) \end{aligned} \tag{13}$$

The definition of $\hat{\phi}$ in Eq. (13) obviously derives Eq. (11) because $\hat{\phi}$ checks if there is a combination of control propositions killing any disjunction, and that combination violates ϕ if there is one.

The safety verifier algorithm is presented in Algorithm 6. The algorithm employs the function $\mathit{safety\text{-}check}^{(d)}_{j,k}\left(a^{(d)}_{j,k}\right)$ presented in the middle of proof by Eq. (19)

Algorithm 6 The safety verifier

Require: all effectiveness control decisions, $a_{j,j}^{(1)} \cdots a_{j,j+\mu}^{(n)}$
  for $k = j$ to $j + \mu$ **do**
    for all $a_{j,k}^{(d)}$ as effectiveness controls in order **do**
      if *safety-check*$_{j,k}^{(d)}(a_{j,k}^{(d)}) = true$ {from Eq. (19)} **then**
        $p_{j,k}^{(d)} = a_{j,k}^{(d)}$
      else $p_{j,k}^{(d)} = \neg a_{j,k}^{(d)}$
      end if
    end for
  end for

Lemma 1 *Algorithm 6 always generates pre-verified safety controls guaranteeing the safety invariants.*

Proof. The proof is by mathematical induction.
Basis:

$$\hat{\phi}\Big(\big(H_{k-\mu-1^+,k}^{(1)}, H_{k-\mu-1^+,k}^{(2)}, \cdots, H_{k-\mu-1^+,k}^{(n)}\big), \hat{E}_{k-\mu-1,k}\Big) = true \tag{14}$$

where $\hat{E}_{j,k}$ is the worst case estimations of environmental propositions provided by patient state estimator. The basis is satisfied because $H_{k-\mu+1^+,k}^{(d)} = \{s_{\text{safe}}^{(\text{d})}\}$ from $\mathcal{P}_{m,k}^{(d)} = s_{\text{safe}}^{(\text{d})}$ if $m < k-\mu$. The safety invariants are always safe if all controls are open-loop safe state regardless of the environmental variables.
Inductive step:
For inductive step, we prove:
if the following holds:

$$\hat{\phi}\left(\big(H_{j-1^+,k}^{(1)}, H_{j-1^+,k}^{(2)}, \cdots, H_{j-1^+,k}^{(n)}\big), \hat{E}_{j-1,k}\right) = true, \tag{15}$$

the following also holds:

$$\hat{\phi}\left(\big(H_{j^+,k}^{(1)}, H_{j^+,k}^{(2)}, \cdots, H_{j^+,k}^{(n)}\big), \hat{E}_{j,k}\right) = true. \tag{16}$$

As a intermediate step from Eq. (15) to Eq. (16), we have

$$\hat{\phi}\left(\big(H_{j^-,k}^{(1)}, H_{j^-,k}^{(2)}, \cdots, H_{j^-,k}^{(n)}\big), \hat{E}_{j,k}\right) = true \tag{17}$$

because we have $H_{j^-,k}^{(d)} \subset H_{j-1^+,k}^{(d)}$ for every device (d) from the fact that an incoming acknowledgement packet gives more information on the device side, and can drop some previously effective NASSs, but cannot add additional

effective NASSs. Moreover, $\hat{E}_{,j-1,k}$ is always more pessimistic than $\hat{E}_{,j,k}$ because of the nature of strict worst-cases.

Algorithm 6 must produce the series of $[p_{j,k}^{(1)}, p_{j,k}^{(2)}, \cdots, p_{j,k}^{(\mathrm{n})}]$ making Eq. (16) be satisfied from the assumption of Eq. (17). The proof of the correctness of the NASS generation is also shown by mathematical induction. The correctness of Basis is given by Eq. (17) because it is the initial state before control $p_{j,k}^{(1)}$ has not been produced yet. In the algorithm, control variables are verified one-by-one in the given dependency order. As an induction step, we assume that all of the NASS of the previously decided devices before device (d), satisfy the safety invariants such that

$$\hat{\phi}\Big(\Big(\big(H_{j^-,k}^{(1)} \cup p_{j,k}^{(1)}\big), \big(H_{j^-,k}^{(2)} \cup p_{j,k}^{(2)}\big), \cdots, \big(H_{j^-,k}^{(d-1)} \cup p_{j,k}^{(\mathrm{d\text{-}1})}\big), H_{j^-,k}^{(d)}, H_{j^-,k}^{(d+1)}, \cdots, H_{j^-,k}^{(n)}\Big), \hat{E}_{j,k}\Big) = \mathit{true}. \quad (18)$$

Algorithm 6 checks the following function for $a_{j,d}^{(d)}$:

$$\begin{aligned} \mathit{safety\text{-}check}_{j,k}^{(d)}\big(a_{j,k}^{(d)}\big) \triangleq \hat{\phi}\Big(\Big(\big(H_{j^-,k}^{(1)} \cup p_{j,k}^{(1)}\big), \big(H_{j^-,k}^{(2)} \cup p_{j,k}^{(2)}\big), \\ \cdots, \big(H_{j^-,k}^{(d-1)} \cup p_{j,k}^{(\mathrm{d\text{-}1})}\big), a_{j,k}^{(d)}, H_{j^-,k}^{(d+1)}, \cdots, H_{j^-,k}^{(n)}\Big), \hat{E}_{j,k}\Big) \end{aligned} \quad (19)$$

and the decision of the function $\mathit{safety\text{-}check}_{j,k}^{(d)}\big(a_{j,k}^{(d)}\big)$ is projected to the pre-verified safety control, $p_{j,k}^{(d)}$ such that

$$p_{j,k}^{(\mathrm{d})} = \begin{cases} a_{j,k}^{(d)} & \text{if } \mathit{safety\text{-}check}_{j,k}^{(d)}\big(a_{j,k}^{(d)}\big) = \mathit{true} \\ \neg a_{j,k}^{(\mathrm{d})} & \text{otherwise.} \end{cases} \quad (20)$$

In Eq. (17), the first case is trivial because the fact that the function $\mathit{safety\text{-}check}_{j,k}^{(d)}\big(a_{j,k}^{(d)}\big)$ is *true* is that the effectiveness control, $a_{j,k}^{(\mathrm{d})}$ is safe with any possible combinations of other controls decided until the decision of $p_{j,k}^{(\mathrm{d})}$. The point is if $\neg a_{j,k}^{(\mathrm{d})}$ can be applied for the safety assurance when $\mathit{safety\text{-}check}_{j,k}^{(d)}$ rejects $a_{j,k}^{(\mathrm{d})}$.

If $a_{j,k}^{(\mathrm{d})} \in H_{j^-,k}^{(d)}$, $a_{j,k}^{(\mathrm{d})}$ passes $\mathit{safety\text{-}check}_{j,k}^{(d)}(a_{j,k}^{(\mathrm{d})})$ because of Eq. (18). If $a_{j,k}^{(\mathrm{d})} \notin H_{j^-,k}^{(d)}$, it may not pass the filter. However, $H_{j^-,k}^{(d)} = \{\neg a_{j,k}^{(\mathrm{d})}\}$ is derived from $a_{j,k}^{(\mathrm{d})} \notin H_{j^-,k}^{(d)}$ because $H_{j^-,k}^{(d)}$ is not an empty set, but a set with Boolean values. Thus, from Eq. (18), and $H_{j^-,k}^{(d)} = \{\neg a_{j,k}^{(\mathrm{d})}\}$, $\neg a_{j,k}^{(\mathrm{d})}$ satisfy the filter if $a_{j,k}^{(\mathrm{d})}$ does not. Consequently, the verifier of Eq. (20) always generates pre-verified safety controls satisfying the safety.

After the last control, $p_{j,k}^{(\mathrm{n})}$ is decided, we have

$$\hat{\phi}\Big(\Big(\big(H_{j^-,k}^{(1)} \cup p_{j,k}^{(1)}\big), \big(H_{j^-,k}^{(2)} \cup p_{j,k}^{(2)}\big), \cdots, \big(H_{j^-,k}^{(n)} \cup p_{j,k}^{(\mathrm{n})}\big)\Big), \hat{E}_{j,k}\Big) = \mathit{true} \quad (21)$$

by induction. Because $H^{(d)}_{j^+,k} = H^{(d)}_{j^-,k} \cup p^{(\mathrm{d})}_{j,k}$, we are aware that Eq. (16) holds. □

The devices in the real world experience some delay in actuation, so the states do not immediately change as our current representation shows. We have extended our algorithm to cover the case of time delays in actuation also.

S1' Between the activities of airway laser and oxygen concentrate, 1 s guard time is required.

The safety invariant for **S1'** is given by

$$c_k^{(\mathrm{oxygen})} \wedge c_l^{(\mathrm{laser})} = \mathit{false} \quad \text{if } |k-l| < \Delta_{1\mathrm{s}} \tag{22}$$

where $\Delta_{1\mathrm{s}}$ is the number of rounds per second. This kind of delay dependent safety invariant can employ our safety engine as long as the pre-verified period is longer than the time duration required to be checked. Compensating for delay requires a minor change in the filter function by expanding the potential states of a device at round j to also include states from all previous rounds between $j - \Delta_{1\mathrm{s}}$ and j.

4.4 Case Study of Safety Layer Operations

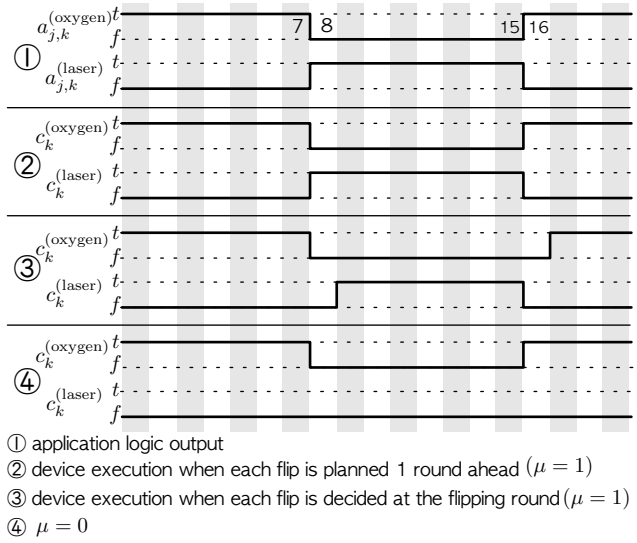

Fig. 4. Example operational sequence of airway-laser interlock case

We have simulated the original airway-laser interlock case given in Section 2. Fig. 4 shows the simulated results of the airway-laser example sequences. There is no packet loss and all environments are estimated to be safe

for all instances in the simulation. The sequence ① presents the application logic output. The application logic wants to turn on the laser at round 8 and to turn off the laser at round 16. For these operations, it wants to pause the oxygen concentrate for the same duration of the laser activation.

Suppose that $\mu = 1$ and all these application decisions are planned one round ahead (② of Fig. 4). Then, at round 7, the application logic generates the application-planned controls for round 8 to be $a_{7,8}^{(\text{oxygen})} = \mathit{false}$, $a_{7,8}^{(\text{laser})} = \mathit{true}$. After safety layer processing, the NASSs for round 8 become $p_{7,8}^{(\text{oxygen})} = \mathit{false}$, $p_{7,8}^{(\text{laser})} = \mathit{false}$. The reason why $p_{7,8}^{(\text{laser})} = \mathit{false}$ is that $c_8^{(\text{oxygen})}$ can become *true*, which is $s_{\text{safe}}^{(\text{oxygen})}$, if all command packets until round 8 are dropped. However, the following packet exchanges were successful in reality, and the planned controls at round 8 are the same as the previous round. Then the NASSs become $p_{8,8}^{(\text{oxygen})} = \mathit{false}$, $p_{8,8}^{(\text{laser})} = \mathit{true}$ because $s_{\text{safe}}^{(\text{oxygen})}$ is not an option for round 8 any more. Then, the devices flip their states together at round 8.

If the application decides the state flipping at the flipping rounds, the behaviors of the devices are like sequence ③. In this case, the application-planned controls and the NASSs at round 7 are $a_{7,8}^{(\text{oxygen})} = p_{7,8}^{(\text{oxygen})} = \mathit{true}$ and $a_{7,8}^{(\text{laser})} = p_{7,8}^{(\text{laser})} = \mathit{false}$ without being aware of flipping of the next round at round 7. At round 8, the application changes its mind and generates the plan, such as $a_{8,8}^{(\text{oxygen})} = \mathit{false}$ and $a_{8,8}^{(\text{laser})} = \mathit{true}$. However, the safety engine disallows $a_{8,8}^{(\text{laser})}$ because $p_{7,8}^{(\text{oxygen})} = \mathit{true}$, which can be employed by the device at round 8, making conflict. Thus, the laser is allowed after the oxygen concentrate is cleared. Even though the safety engine delays the operations, it is the best for the guaranteed safety.

5 Prototype System

We have implemented a prototype of the PVSC framework in our medical tested. Our testebed consists of three computers connected through a private IEEE 802.11g wireless network in a non–line-of-sight configuration in 5 m distances to demonstrate operating room scenario. One computer served as the ICCS manager and the others demonstrate device controls, sometimes connected to experimental medical devices such as infusion pumps. We have implemented the synchronous communication layer in conjunction with the PVSC framework. The round duration is set to 100 ms and command packets are supposed to come in 40 ms from the beginning of a round. Acknowledgement packets must be received before the end of a round. μ was set to be 20. The packet drop rate was 0.24% during 9 hours of experiments. The maximum number of consecutive rounds with series of packet drops was 11 (*i.e.* for 1.1 s) but mostly less than 5. The safety of the system was ensured as designed. The software environment is Java2 Runtime Environment 1.5.0 and we em-

ployed [Cri89] for clock synchronization between the ICCS manager and the devices. The application logic is developed as using robust communications.

Even though the packet drop rate was quite low, some noticeable consecutive rounds had series of packet losses. If we use *the time difference between the control generation and the execution* as the quality of service metric, we found that the PVSC framework has served quite optimized quality of service to the devices adapting itself. At the most of rounds (99.76%) when the channel condition was good, the devices apply the controls made at the same round. Only when there are packet losses, the devices used some aged controls. And that ages of controls were identical to the number of consecutive rounds without delivered packets.

6 Related Work

Our work is motivated by the on going initiative of Medical Device Plug-and-Play [Gol09]. We proposed a framework, called NASS framework [KSM+10], for the same purpose of this paper in the form of finite state machine (FSM), but PVSC simplifies the approach by employing Boolean logic instead of FSM. This initiative has been quite active for the past ten years resulting in the ISO/IEEE 11073 family of communications standards [SFWS07], healthcare alliances such as Continua, and specification alliances such as Bluetooth and ZigBee.

Although wireless technology is readily available inside current medical devices [BH08], few have been used for control. The techniques of deploying safe interoperability in medical devices (wired or wireless) have only recently gained momentum. Arney et. al demonstrated synchronization techniques of medical devices with an X-ray and a ventilator [AJJ+07]. Fischmeister et. al. applied their work for the Network Code Machine [FSL07] to be deployed in medical systems providing verifiable real-time performance demonstrated in HIMSS '08. Software architectures for communications in medical plug-and-play systems have also been explored by King et. al. [Ke08] using publish-subscribe architectures for dynamic information flow. Currently, much of the work for medical plug-and-play focus on estabilishing dynamic connectivity of devices, device-to-device synchronization, and ensuring fair access to a communication medium not focusing on enforcing generic types of safety constraints with the addition of wireless.

On the other end of the spectrum, medical device safety has been a very prevalent issue dating back to the infamous incidents in the 80's involving the Therac 25 radiation therapy machines [LT93]. Since then, much work has been done to apply formal methods to medical devices analysis [AJJ+07, RC04, AAG+04]. The use of formal methods may even start to influence actual medical device review procedures [JIJ06]. However, much of the formal analysis work has been done on individual devices without any

interoperable behavior between a network of devices. We have taken the initiative to move forward in this direction.

7 Concluding Remarks

It is important to provide the best convenience to caregivers for patient safety in a health-care environment; providing wireless connectivity for medical devices is one of them. We presented the pre-verified safety control framework interconnecting and managing medical devices through wireless with guaranteed safety and high quality of service. Even though this framework is based on a non-trivial pipeline planning method, all of the complications are hidden from the application logic by the application engine. The provable safety verifier is automatically generated from the given safety invariants in Boolean logic.

From our prototype, we found the application logic and the end devices are well protected from external flaws and complications by applying the NASS framework in Boolean logic. Furthermore, the quality of the communication support was also decent.

References

[AAG+04] Rajeev Alur, David Arney, Elsa L. Gunter, Insup Lee, Jaime Lee, Wonhong Nam, Frederick Pearce, Steve Van Albert, and Jiaxiang Zhou. Formal specifications and analysis of the computer-assisted resuscitation algorithm (CARA) Infusion Pump Control System. *International Journal in Software Tools for Technology Transfer*, 4, 2004.

[AJJ+07] David Arney, Raoul Jetley, Paul Jones, Insup Lee, and Oleg Sokolsky. Formal Methods Based Development of a PCA Infusion Pump Reference Model: Generic Infusion Pump (GIP) Project. In *Proc. of Joint Workshop on HCMDSS-MDPnP*, Jun. 2007.

[BH08] Steven D. Baker and David H. Hoglund. Medical-grade, mission-critical wireless networks. *IEEE Engineering in Medicine and Biology Magazine*, Mar./Apr. 2008.

[cod] CodeBlue: Wireless Sensors for Medical Care. http://fiji.eecs.harvard.edu/CodeBlue.

[Cri89] Flavin Cristian. A Probabilistic Approach to Distributed Clock Synchronization. In *Proc. of ICDCS*, Jun. 1989.

[FSL07] Sebastian Fischmeister, Oleg Sokolsky, and Insup Lee. A Verifiable Language for Programming Real-Time Communication Schedules. *IEEE Trans. on Comp.*, 56(11), Nov. 2007.

[Gol08] Julian Goldmann. Medical Devices and Medical Systems — Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE) — Part 1. draft ASTM TC F29.21 N 21, Sep. 2008.

[Gol09] Julian Goldmann. Medical Device Interoperability to Enable System Solutions at the Sharp Edge of Healthcare Delivery. White House Homeland Security Council Biodefense Directorate Conference presentation, Apr. 2009.

[Gra07] Health Grades. Fourth Annual Patient Safety in American Hospitals Study, 2007.

[Hig09] High Confidence Software and Systems Coordinating Group. High-Confidence Medical Devices: Cyber-Physical Systems for 21st Century Health Care. A Research and Development Needs Report, NITRD, Feb. 2009.

[JIJ06] Raoul Jetley, S. Purushothaman Iyer, and Paul L. Jones. A Formal Methods Approach to Medical Device Review. *IEEE Computer*, 39(4), Apr. 2006.

[Ke08] Andrew King and et. al. An Open Test Bed for Medical Device Integration and Coordination. Technical report, Kansas State University, 2008.

[KSM+10] Cheolgi Kim, Mu Sun, Sibin Mohan, Heechul Yun, Lui Sha, and Tarek F. Abdelzaher. A framework for the safe interoperability of medical devices in the presence of network failures. In *Proc. of ICCPS*, pages 149 – 158, April 2010.

[LT93] Nancy Leveson and Clark Turner. An Investigation of the Therac-25 Accidents. *IEEE Computer*, 26(7):18–41, July 1993.

[Mar07] Adam Marcus. Once a Tech Fantasy, Plug-and-Play OR Edges Closer to Reality. *Anesthesiology News*, 33(1), Jan. 2007.

[RC04] Arnab Ray and Rance Cleaveland. Unit verification: the CARA experience. *International Journal on Software Tools for Technology Transfer*, 2004.

[SFWS07] L. Schmitt, T. Falck, F. Wartena, and D. Simons. Novel ISO/IEEE 11073 Standards for Personal Telehealth Systems Interoperability. In *Proc. of Joint Workshop on HCMDSS-MDPnP*, 2007.

http://www.springer.com/978-3-0348-0030-3

Reliable and Autonomous Computational Science
International Conference, RACS 2010, Atlanta, GA, USA,
October 27–30, 2010
Shin, S.Y.; Gantenbein, R.; Kuo, T.-W.; Hong, J. (Eds.)
2010, XVI, 410p., Softcover
ISBN: 978-3-0348-0030-3
A product of Birkhäuser Basel