A Framework for the Safe Interoperability of Medical Devices in the Presence of Network Failures

Cheolgi Kim
University of Illinois
201 N Goodwin Ave.
Urbana, IL 61801, USA
cheolgi@illinois.edu

Mu Sun
University of Illinois
201 N Goodwin Ave.
Urbana, IL 61801, USA
musun@illinois.edu

Sibin Mohan
University of Illinois
201 N Goodwin Ave.
Urbana, IL 61801, USA
sibin@illinois.edu

Heechul Yun
University of Illinois
201 N Goodwin Ave.
Urbana, IL 61801, USA
heechul@illinois.edu

Lui Sha
University of Illinois
201 N Goodwin Ave.
Urbana, IL 61801, USA
lrs@illinois.edu

Tarek F. Abdelzaher
University of Illinois
201 N Goodwin Ave.
Urbana, IL 61801, USA
zaher@cs.uiuc.edu

ABSTRACT

There exists a growing need for automated interoperability among medical devices in modern healthcare systems. This requirement is not just for convenience, but to prevent the possibility of errors due to the complexity of interactions between the devices and human operators. Hence, a system supporting such interoperability is supposed to provide the means to interconnect distributed medial devices in an open space, so must be designed to account for network failures. In this paper, we introduce a generic framework, the *Network-Aware Supervisory System* (NASS) to integrate medical devices into such a clinical interoperability system that uses real networks. It provides a development environment, in which medical-device supervisory logic can be developed based on the assumptions of an ideal, robust network. A case study shows that the NASS framework provides the same procedural effectiveness as the original logic based on the ideal network model but with protection against real-world network failures.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Reliability, availability, and serviceability*; C.2.4 [**Computer—Communication Networks**]: Distributed Systems—*Distributed applications*

1. INTRODUCTION

The medical field has seen significant transformations of late with the advent of a whole plethora of medical devices that aid in the process of diagnosis, patient monitoring and even administering medication. Most such devices have real-time constraints as failure to operate in a timely manner could result in patient harm or death. Furthermore, in many cases, safety of a medical system depends on the global distributed state across multiple devices. For example, *an accidental burn caused during an airway surgery* happens when an airway surgery laser is activated before the oxygen concentration is turned down from an oxygen supply. The need for medical personnel (human operators) to enforce these interoperability of medical devices means there is high probability of error. The 2006 American Society of Anesthesiologists estimated that roughly 100 such fires occur each year in U.S. hospitals causing roughly 20 serious injuries and even one or two deaths [18]. In general, preventable accidents in hospitals occur at alarming frequencies. Between the years of 2003 – 2005, 247,662 safety-related incidents were potentially preventable [10].

To improve safety in medical environments, there is a need to develop interoperable medical devices that can *automatically* operate with safety interlocks *i.e. without* a human in the loop. Recent initiatives have been launched to increase *interoperability* among medical devices to reduce accidents caused by human errors [11]. One such effort is the work on the *Integrated Clinical Environment* (ICE) draft standard [9] published in 2009 by ASTM. The ICE standard aims to provide standardized integration of data and devices to enable *real-time* control decision support and safety interlocks, thus ensuring patient safety. The Medical Device Plug-and-Play (MD PnP) Interoperability program initiated by CIMIT (Center for Integration of Medicine & Innovative Technology) covers a wide range of necessary efforts in an *Integrated Clinical Supervisory Systems (ICSS)*. This includes such efforts as eliciting high-level surgical scenarios to understanding regulatory pathways for patient-centric networked medical devices and even prototype development [3, 8, 13].

As a major safety-core component of the ICSS, we focus on developing safety supervisory system for surgical procedures. Surgical procedures are risky since many times caregivers must intentionally put a patient temporarily into a speculative state to perform a medical operation: administering anesthesia, cutting tissues, temporarily stopping life support devices, etc. Design of these safety-critical supervisory systems are further complicated in a medical environment by:

1. *real-time operation*: surgical procedures clearly requires timely operation.
2. *inter-device dependencies*: certain device operations must be performed in a certain order, *e.g.* turning off the oxygen before starting the laser.
3. *unreliable communication*: ICCS cannot assume that the underlying network that connects the medical devices is reliable. For wired systems, people may trip over the physical wires. Wireless networks are even worse: communication signals are subject to background noise, self fading, etc. [25].

We take into account all of these considerations into design of a provably safe ICSS.

In this paper, we describe a safe ICSS design that can be used to monitor the procedure of the medical operation and prohibit an action that can threaten the safety of the patient or enforce actions that protect the patient. For example, when an airway-laser activation is requested an ICSS can pause the oxygen flow and activate the laser with some latency to ensure that the oxygen that was previously pumped in is completely exhausted before the laser is turned on. This supervision by an ICSS can avert accidents caused by human errors by enforcing checks against medical protocols.

We propose a new *Network-Aware Safety Supervision* (NASS) framework to deal with communication failures in an ICSS. One of the main goals of our work is to delegate the burden of handling connection failures from the application logic to an underlying system framework. The NASS framework has the following characteristics:

1. *Basic Model:* NASS provides a simple high level abstraction for designing for *surgical-procedure supervisory systems* with an ideal robust network (*basic supervisory system model*)
2. *Network-Aware Framework:* Using the *basic model*, NASS creates a deployable ICSS that can *safely operate* in the presence of unreliable networks and failures.
3. *Effectiveness:* If no packet losses occur, the system executing on the network-aware framework is supposed to behave logically equivalent to the basic model as long as the behavior is surely safe.

This means that by using NASS, we obtain the benefits of a simple development environment (ideal synchronous communication), but with safety guarantees for an actual deployed system in a real medical environment (asynchronous communication with failures). Furthermore, we can obtain expected behavior when the physical network is reliable enough.

To the best of our knowledge, this is the first time that a framework for the *correct and safe* operation of medical devices that inter-operate with each other, in the presence of communication failures, has been developed. There have been frameworks that utilize supervisions for diagnosing faults in discrete-event systems (DES) but they are difficult to be directly applied for the supervision of medical devices. This happens because a new supervisor must be designed and analyzed on every configuration change, no matter how minor to apply such an approach [16, 24] .

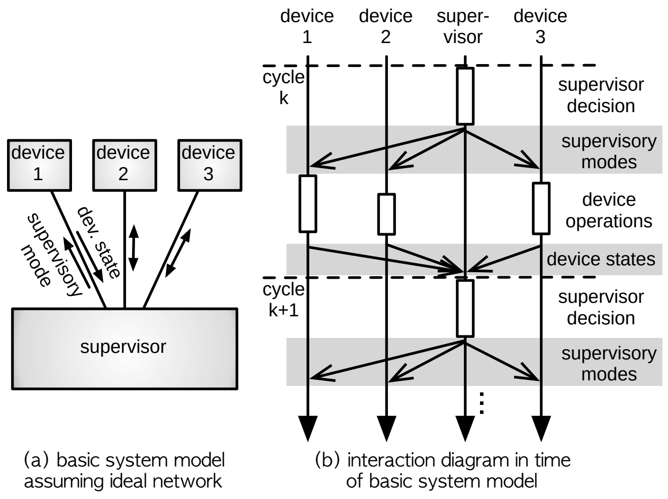

Figure 1: Architecture overview of basic system model

The rest of this paper is structured as follows: Section 2 presents the formal representation of the basic supervisory system model. Section 3 shows how the NASS framework provides a safe ICSS in the presence of network problems is described. A case study that addresses the problems with the airway-laser surgery example is presented in Section 4. Related work is discussed in Section 5 and Section 6 concludes the paper.

2. BASIC SYSTEM MODEL WITH IDEAL NETWORK

The Basic Supervision System Model (or just *basic model*) describes an ideal medical device supervisory logic during surgical procedures. It assumes no failure in the supervisory logic or the network. Through NASS, system developers only need to specify supervisory system in terms of this basic model. The network exception handling are automatically done by the lower levels of the NASS framework. In this section, we describe the details of the basic system model (Fig. 1(a)).

The basic model has one central supervisor, connecting all required medical devices through a star network. For each connected device, the supervisor decides a *supervisory mode*, and the device will follow this mode of operation. A supervisory mode for a device declares a safe region of operations for the device. These regions need not be disjoint; sometimes the region of one mode can be a subset of the region of another mode. For example, the airway laser can have the supervisory modes of ⟨allow-usage⟩ and ⟨deactivated⟩. When the supervisory mode of the laser is ⟨allow-usage⟩, the airway laser can be either activated or deactivated. When the supervisory mode is ⟨deactivated⟩, only the laser being deactivated is allowed.

The supervisory modes of all the connected devices are periodically issued by the supervisor. Once each device receives the supervisory mode, it replies with its *device state* (device status, measured patient state, physician input, etc.). Using all the device-state replies, the supervisor predicts the state of the patient, and decides the next supervisory modes of operation for all the devices. The period of these handshakes between the supervisor and the devices are called a *supervision cycle* (or just *cycle*). The basic model assumes all handshakes can be completed within one cycle (Fig. 1(b)).

The supervisor must generate supervisory modes that enforce the safety of the system, while the devices are responsible for operating within the declared operational region of the given mode in each cycle. The safety of the ICSS then follows for correctly performing these two responsibilities.

We use a centralized architecture with periodic synchronous communication model for simplicity of the development. Distributed components and asynchronous communication are major sources of complexity in system development [1]. The basic model eliminates the need to consider these complexities for actual system developers. Of course, these ideal network assumptions are not valid in practice, but this will not compromise safety because of the safety layers provided by the NASS framework described in Section 3.

2.1 Basic Model Formalism

Device model. The definition of device model is about how to define the supervisory modes, the device states, and their relationships. Let $\mathcal{D} = \{d_1, d_2, \cdots, d_n\}$ denote the set of the devices. For each $d \in \mathcal{D}$, $d = (\mathit{Modes}^{(d)}, \bullet\!\!\xrightarrow[\mathit{Modes}]{}{}^{(d)}, \mathit{mode}_0^{(d)}, \mathit{States}^{(d)}, \models_{\mathit{Allows}}^{(d)})$. $\mathit{Modes}^{(d)}$ is the set of device modes directed by the supervisor; $\bullet\!\!\xrightarrow[\mathit{Modes}]{}{}^{(d)} \subseteq \mathit{Modes}^{(d)} \times \mathit{Modes}^{(d)}$ is a relation denoting the set of the valid mode transitions between supervisory modes from one cycle to the next one where notation of $\mathit{mode}_a^{(d)} \bullet\!\!\xrightarrow[\mathit{Modes}]{} \mathit{mode}_b^{(d)}$ means that the mode transition from $\mathit{mode}_a^{(d)}$ to $\mathit{mode}_b^{(d)}$ is valid; $\mathit{mode}_0^{(d)} \in \mathit{Modes}^{(d)}$ is the initial supervisory mode; $\mathit{States}^{(d)}$ is the set of device states; and $\models_{\mathit{Allows}}^{(d)} \subseteq \mathit{Modes}^{(d)} \times \mathit{States}^{(d)}$ define the operational region of the *States* when *mode* is given, where notation of $\mathit{mode}_a^{(d)} \models_{\mathit{Allows}} \mathit{state}_b^{(d)}$ means that $\mathit{state}_b^{(d)}$ is in the operational region of $\mathit{mode}_a^{(d)}$. For hereon forward, we will drop the superscript for device d when it is clear from context.

Recall that the device must be made to ensure that each device state $\mathit{state} \in \mathit{States}$ is allowed in supervisory mode *mode* if and only if $\mathit{mode} \models_{\mathit{Allows}} \mathit{state}$. It is responsibility of the device developer.

Supervisor model. When a new cycle starts, the supervisor manages all the information of the previous cycle, including the supervisory modes and the device states of the devices. The supervisor updates the patient state estimated inside, and decides the supervisory modes for the active devices.

For convenience, let S^* denote the Cartesian product of all the $\mathcal{D}$ indexed sets of S for convenience. That is:

$$S^* = \prod_{d \in \mathcal{D}} S_d.$$

For example, global mode of operation for all devices is $\mathit{Modes}^* = \mathit{Modes}^{(d_1)} \times \mathit{Modes}^{(d_1)} \times \cdots \times \mathit{Modes}^{(d_n)}$ for $\mathcal{D} = \{d_1, d_2, \cdots, d_n\}$.

The supervisor is defined by a tuple

$$\mathcal{S} = (\mathcal{P}, \mathit{Modes}^*, \mathit{States}^*, \xmapsto[\mathcal{P}]{}, \xmapsto[\mathit{Spv.}]{}, \mathit{Safe}^*) \tag{1}$$

$\mathcal{P}$ is the set of the patient states; Modes^* is the set of *global supervisory modes* of all devices; States^* is the set of the possible acknowledgements from all devices; $\xmapsto[\mathcal{P}]{}: \mathcal{P} \times \mathit{States}^* \rightarrow \mathcal{P}$ is the transition function of the patient state with regard to the replied device states; $\xmapsto[\mathit{Spv.}]{}: \mathcal{P} \times \mathit{States}^* \times \mathit{Modes}^* \rightarrow \mathit{Modes}^*$ is the compositional state transition function of the supervisory modes of devices (notice that the transition of $\xmapsto[\mathit{Spv.}]{}$ is deterministic because the patient context is available, while the valid mode transition within each device, $\bullet\!\!\xrightarrow[\mathit{Modes}]{}$, is not), and $\mathit{Safe}^* \subseteq \mathcal{P} \times \mathit{Modes}^*$ defines the safety of the system.

The procedure of computing the next supervisory states of the devices is as follows. The device states collected at cycle $(k-1)$, which is given by $(\mathit{state}_{k-1}^{(d_1)}, \mathit{state}_{k-1}^{(d_2)}, \cdots, \mathit{state}_{k-1}^{(d_n)}) \in \mathit{States}^*$, has new patient state information and the updated physician input. Based on them, the supervisor updates its patient state of cycle k, such that

$$p_{k-1} \xmapsto[\mathcal{P}]{(\mathit{state}_{k-1}^{(d_1)}, \mathit{state}_{k-1}^{(d_2)}, \cdots, \mathit{state}_{k-1}^{(d_n)})} p_k \tag{2}$$

where $p_k \in \mathcal{P}$ is the patient state at cycle k. The state of the surgical procedure is a part of the patient state, so is updated together by information in an acknowledgement, too. Then finally, it computes the supervisory mode from the updated patient state and the device states, such that

$$\begin{aligned}&\left(\mathit{mode}_{k-1}^{(d_1)}, \cdots, \mathit{mode}_{k-1}^{(d_n)}\right)\\ &\quad \xmapsto[\mathit{Spv.}]{p_k, (\mathit{state}_{k-1}^{(d_1)}, \cdots, \mathit{state}_{k-1}^{(d_n)})} \left(\mathit{mode}_k^{(d_1)}, \cdots, \mathit{mode}_k^{(d_n)}\right)\end{aligned} \tag{3}$$

where $\mathit{mode}_k^{(d)}$ is the shared notation of the supervisory mode for for device d kept by the supervisor at cycle k. Indeed, the system developer is supposed to define $\mathit{mode}_k^{(d)}$ to supervise surgical procedures. It is the core of the supervisory logic.

Notice that the supervisory mode transition performed by Eq. (3) must not violate the transition constraints given by $\bullet\!\!\xrightarrow[\mathit{Modes}]{}$:

$$(\forall d \in \mathcal{D})(\forall k), \quad \mathit{mode}_{k-1}^{(d)} \bullet\!\!\xrightarrow[\mathit{Modes}]{} \mathit{mode}_k^{(d)} \tag{4}$$

which is the result of the projection of the global transition function $\xmapsto[\mathit{Spv.}]{}$. Eq. (4) is called a *transition validity* of the system. Although the transition validity must be obviously managed in the system design based on the basic model, it is one of the major concerns of some of algorithms in the

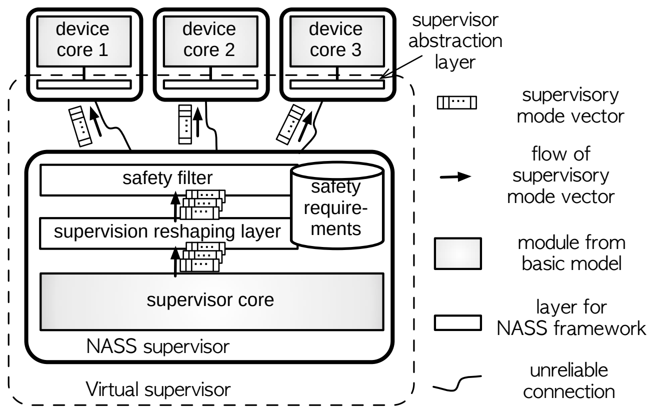

Figure 2: Architecture overview of NASS framework

NASS framework because the algorithms have to reform the supervisory commands to contend with network problems, and transition validity must be preserved during the reformation of the NASS framework.

The safety of the system at each cycle is captured by the fact that the patient is safe if and only if the containment relation holds $\left(p_k, \left(mode_k^{(d_1)}, \cdots, mode_k^{(d_n)}\right)\right) \in Safe^*$.

3. NASS FRAMEWORK

Using the basic model, the NASS framework preserves the central supervisor topology with a *NASS supervisor* and connected devices. Once the devices and supervisor are developed using the basic system model, the NASS framework wraps the components of the basic model with safety layers to deploy in unreliable networks (Fig. 2 shows an overview).

The most noticeable difference of the NASS framework from the basic model is that the supervisor forwards a supervisory mode vector instead of a single supervisory mode to a device. For each device, the vector contains a plan of supervisory modes for future cycles in the case of a network failure or packet losses. The NASS framework generates the vectors using the original supervisor core defined in the basic model. The safety of the generated vectors by the supervisor core is double checked by the supervision reshaping layer and the safety filter of the NASS supervisor as shown in Fig. 2. The resulting supervisory mode vector is provably safe when employed by each device. Furthermore, any loss of vectors in the middle of delivery cannot affect the system safety.

The device developed using the basic model is not ready to accept supervisory mode vectors, so it must have an additional layer called the *supervisor abstraction layer*. The supervisor abstraction layer plays the role of the original supervisor in the basic model: providing one supervisory mode at a cycle. The supervisor abstraction layer takes a supervisory vector and delivers a corresponding supervisory mode per cycle to the core device logic (original device logic in the basic model). Even if the delivery of a supervisory mode vector over the physical network is lost, the supervisory abstraction layer always provides a supervisory mode at each cycle based on an earlier supervisory vector. See the dashed line combining the NASS supervisor with these supervisor abstraction layers of the devices, which constitutes the *virtual supervisor* at Fig. 2. This virtual supervisor plays the same role of the supervisor in the basic model: generating a supervisory mode to every device at a cycle ensuring the safety.

The reason why we employ the supervisory vectors is to handle network problems, ultimately failures. How can the system ensure safety under network failure using such vectors? Indeed, each element of the supervisory vector represents a supervisory mode for one future cycle; when a new supervisory vector is not delivered, the supervisor abstraction layer consumes one element of the vector for the supervisory mode of the cycle. When the finite length vector runs out of commands, the last supervisory mode in every vector is reused for the rest of the future; this last element is called the network-fail-safe mode of the device because it is used when network is technically disconnected. How to deal with the network-fail-safe modes appropriately is the key of ultimate safety under network failures. We will discuss the details in this section.

3.1 Supervisor abstraction for device core

Let us first see how each supervisor abstraction layer extracts a supervisory mode of each cycle from the delivered supervisory vectors. Recall that the vectors are not guaranteed to be delivered due to the nature of real networks. The supervisory vector, $\overrightarrow{mode\text{-}vec_k}$, for each device delivered at cycle k is denoted by

$$\overrightarrow{mode\text{-}vec_k} = (\dot{m}_{k,k}, \dot{m}_{k,k+1}, \cdots, \dot{m}_{k,k+\mu})$$

where $\dot{m}_{k,j} \in Modes$, and the size of every vector is $(\mu + 1)$.

The supervisory mode at cycle k to the device core is generated based on:

$$mode_k = \begin{cases} \dot{m}_{r_k k} & \text{if } r_k + \mu > k \\ \dot{m}_{r_k, r_k+\mu} & \text{if } r_k + \mu \leq k \end{cases} \tag{5}$$

where r_k is the most recent cycle at which the device received a valid supervisory vector at the point of cycle k.[1] As long as r_k can be given, $mode_k$ is given at any cycle without discontinuity of generation.[2] Intuitively, Eq. (5) presents that the i-th element of $\overrightarrow{mode\text{-}vec_k}$ is given for the use in the case that $(i-1)$ successive supervisory-vector packet losses happen after cycle k. Recall that the last element of $\overrightarrow{mode\text{-}vec_k}$, which is $\dot{m}_{k,k+\mu}$, is the special supervisory mode designed to be used when more than μ vectors are lost after cycle k, called the *network-fail-safe supervisory mode* of the device, and is a constant denoted by $mode_{\text{safe}}^{(d)}$ for device d, such that $(\forall k)\ \dot{m}_{k,k+\mu}^{(d)} = mode_{\text{safe}}^{(d)}$. For the richness of formal expression, we generalize the notation of $\dot{m}_{k,j}$ such that $(\forall j > k+\mu), \dot{m}_{k,j} \triangleq mode_{\text{safe}}$ even though it is out of the supervisory vector content.

[1]The meaning that a supervisory vector is valid is that the vector does not violate the valid transition requirement, such that $mode_{k-1} \overset{\bullet}{\underset{Modes}{\longrightarrow}} \dot{m}_{k,k}$.

[2]If the first supervisory vector, $\overrightarrow{mode\text{-}vec_0}$ of a device has not been delivered, Eq. (5) is not sufficient for the claim. We assume that $\overrightarrow{mode\text{-}vec_0}$ is delivered at the connection establishment transaction.

3.2 NASS supervisor

Generation of supervisory vectors with ensured safety is in three steps by the supervisor core, the supervision reshaping layer, and the safety filter depicted in Fig. and 2.

- *Supervisor core*: initial vector is generated by the supervisor core, which is delivered from the basic model.
- *Supervision reshaping layer*: Recall that the supervisory vectors must have the transitions to the network-fail-safe modes at the end of the vector, which the supervisor core does not care for. The second step is to make the transition by the supervision reshaping layer.
- *Safety filter*: The safety filter at the final stage certifies ultimate safety of the vectors against network faults.

3.2.1 Supervisor core

According to the definition of the supervisor core (i.e. the supervisor of the basic model) in Eq. (1), the supervisor core has to keep track of the supervisory modes and the device states of all the devices. Even though it is capable in the basic model because of the ideal network, the NASS framework cannot always be aware of the exact status of each device due to the unreliable network assumption. For example, the supervisor expects a device employs the first element of the vector ($\dot{m}_{k,k}$) at any cycle k, but the device may not receive the vector and the second element of vector received at the previous cycle ($\dot{m}_{k-1,k}$) can be used instead. If we have $\dot{m}_{k,k} \neq \dot{m}_{k-1,k}$, the expectation of the supervisor does not match the actual mode of operation at the device. The same situation is applied for the device states. Thereby, the supervisor abstraction layer of each device forwards not only $state_k$ but also $mode_k$ as an acknowledgement of the supervisory vector at each cycle k. And the supervisor core of the NASS supervisor uses the best knowledge of device states and the supervisory modes of actual devices, such that

$$est\text{-}mode_k = \begin{cases} mode_k & \text{if acknowledgement is received} \\ \dot{m}_{k,k} & \text{otherwise} \end{cases}$$

$$est\text{-}state_k = \begin{cases} state_k & \text{if acknowledgement is received} \\ \epsilon & \text{otherwise} \end{cases}$$

where $est\text{-}mode_k$ and $est\text{-}state_k$ is the estimated supervisory mode and device state at cycle k based on the best knowledge of the supervisor, and ϵ is a special notation that means no device state update is given.[3]

The generation of the first element of the vector is the same as the one given in Eq. (2) and (3) because the element is for the current cycle; however $est\text{-}mode_k$ and $est\text{-}state_k$ is used instead of $mode_k$ and $state_k$; for Eq. (3), the first element of the vector is given by

$$\left(est\text{-}mode_{k-1}^{(d_1)}, \cdots, est\text{-}mode_{k-1}^{(d_n)}\right)$$
$$\xmapsto[Spv.]{p_k, (est\text{-}state_{k-1}^{(d_1)}, \cdots, est\text{-}state_{k-1}^{(d_n)})} \left(\overset{init}{m}{}^{(d_1)}_{k,k}, \cdots, \overset{init}{m}{}^{(d_n)}_{k,k}\right)$$

where the initial supervisory vector generated by the supervisor core is denoted by The generation of the rest of the vector is given by $\overrightarrow{init\text{-}vec_k} = (\overset{init}{m}_{k,k}, \cdots, \overset{init}{m}_{k,k+\mu})$. For the rest of the vector, we have

$$\left(\overset{init}{m}{}^{(d_1)}_{k,j-1}, \cdots, \overset{init}{m}{}^{(d_n)}_{k,j-1}\right) \xmapsto[Spv.]{\hat{p}_{k,j}, (\epsilon, \cdots, \epsilon)} \left(\overset{init}{m}{}^{(d_1)}_{k,j}, \cdots, \overset{init}{m}{}^{(d_n)}_{k,j}\right)$$

for $k < \forall j < k + \mu$ where $\hat{p}_{k,j}$ is the worst-case patient state of future cycle j estimated at cycle k. By appending $mode_{\text{safe}}$ to the initial supervisory vector at the end as the last element, the initial supervisory vector generation is completed.

The worst-case patient state estimator can be available because the patient state changes gradually. Some literature about estimation of some patient states is available in [19,22] as a different research area. We do not address how to estimate the worst-case patient state of future cycles, in this paper.

3.2.2 Reshaping initial supervisory vector

The next step is the supervision reshaping layer to provide pessimism to the initial supervision vector for the total network disconnection. The first step toward handling total connection failure is already performed by appending the network-fail-safe mode, $mode_{\text{safe}}$ at the end of $\overrightarrow{init\text{-}vec_k}$. However, it is not sufficient for the total connection failure because that appending may cause transition invalidity; notice that $\overset{init}{m}_{k,k+\mu-1} \bullet\!\!\xrightarrow[Modes]{} mode_{\text{safe}}$ must hold for the transition validity, but has not been tested in any part of the supervisor core. Therefore, some intermediate states making the transition valid must be inserted between the initial vector and the network-fail-safe mode at the end if the transition is invalid.

Moreover, the reshaping layer has to take care of the order of the devices moving to the network-fail-safe modes. Recall the airway surgery example, in which the laser must be deactivated before the oxygen flow is resumed as the network-fail-safe mode. To take care of both the transition validity and the transition order of the devices is a little complicated. In this paper, we presented how to reshape the initial supervisory vector by demonstrating it with case study of airway laser in Section 4.3.

3.2.3 Safety filter

The last step to provide the supervisory vectors to the devices is to filter out yet remained unsafe supervision. Since the initial supervisory vectors are generated based on incomplete information, they cannot guarantee the total safety, and so do reshaped ones. The safety filter measures all possible cases that can be realized at the devices, and drop the supervisory vectors that are revealed to be potentially unsafe.

As the supervisor cannot be aware of the exact supervisory mode that a device would employ, it investigates the set of all the supervisory modes that a device can employ with its best knowledge, such that

$$\textit{eff-mode}_{k,j} = \bigcup_{\beta_k \leq i < k} \{\dot{m}_{i,j}\}$$

where $\textit{eff-mode}_{k,j}$ is the set of supervisory mode that a device can employ at cycle j when it is estimated by the super-

[3]The supervisor core must be able to handle ϵ for the case that a device state is not delivered at a cycle.

visory at cycle k, and β_k is the most recent cycle at which a device received a valid supervisory vector with the best of the supervisor's knowledge. As an induction, we can assume that all vectors that have been delivered previously are safe in any occasion; *i.e.* $\mathit{eff}\text{-}\mathit{mode}_{k,j}$ of each device is proven to be safe for employment in a previous cycle already. If we send a reshaped vector to a device, the device will have more option of supervisory mode in addition to $\mathit{eff}\text{-}\mathit{mode}_{k,j}$. The safety filter evaluates whether each reshaped vector is safe combined with $\mathit{eff}\text{-}\mathit{mode}_{k,j}$'s, and other reshaped vectors for the other devices that are generated at the same cycle. Only safety-certified vectors by the safety filter are sent to the devices, and the others are trashed. A case study how safety filter operates is given in Section 4.

3.3 Network-fail-safe mode

In presenting the NASS framework, we have declared that there exist the network-fail-safe modes, $\mathit{mode}_{\text{safe}}$, but have not addressed what they are and how they are found, yet. One of the biggest challenges that the NASS framework confronts is the fact that all the devices can be completely disconnected from the supervisor. Recall that the last element of the vector is set to $\dot{m}_{k,k+\mu} = \mathit{mode}_{\text{safe}}$, essentially to prepare for the complete network failure. Suppose that all the devices are disconnected for a while. Then, all of them must be in the network-fail-safe mode. When every device is in the network-fail-safe mode, the whole system must be safe regardless of the state of the patient. Thereby, we have

$$p \times \prod_{d\in\mathcal{D}} \mathit{mode}_{\text{safe}}^{(d)} \subset \mathit{Safe}. \quad (6)$$

From Eq. (6), the network-fail-safe mode is found, and a practical algorithm is demonstrated in Section 4.3 with a case study.

Availability of network-fail-safe mode. One of the natural questions that we can have at this point is if all the surgical procedure supervisory systems have this network-fail-safe mode. The answer can be found from traditional surgical procedures. In such a procedure, the caregivers play the role of the network in an ICSS. If the system is not network-fail-safe, there is no supervisory mode satisfying Eq. (6); *i.e.* some device operation is totally dependent upon a patient state. That means, in a traditional procedure, there must be a dedicated person to keep watching the patient state and the device operation, together for the safety in real-time, which is not practically advisable. Therefore, the medical device manufacturers commonly put the related patient sensing function to the device, in order to make the device maintain the safety locally. For example, an air pressure sensor is always embedded in a ventilator to avoid the safety issue of high-pressure ventilation. Because surgical procedures and the devices used in the procedures are designed by people with common senses, pursuing a network-fail-safe mode in a surgical procedure does not seem to be nonsense.

4. CASE STUDY: AIRWAY-LASER SURGERY

We present a case study of an airway-laser surgery system to show how the NASS framework can be employed in practice. By presenting how the components of NASS supervisor, the supervisor core, the supervision reshaping layer and the safety filter, generate and manipulate the supervisory mode vectors with ensured safety.

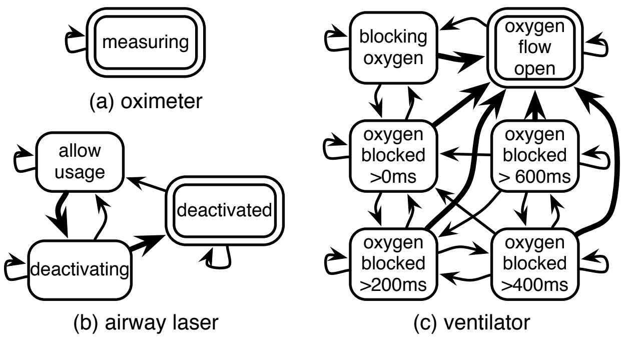

Figure 3: Supervisory state transition diagrams for devices in airway-laser surgery

Let us briefly recall the airway-laser surgery example. An airway-laser surgery has a potential danger of an accidental burn if the laser is activated while high oxygen concentration is supplied by the ventilator. Whenever the laser is being activated, the ICSS must block (or significantly reduce) the air path from the oxygen concentrate, first. However, before *SpO* level of the patient becomes below a given threshold, the laser must be deactivated to open the oxygen flow through the ventilator; otherwise, the patient can suffer a low-oxygen shock. This case study assumes that three devices: airway-laser, ventilator and oximeter are connected for the surgery ($\mathcal{D} = \{\text{laser}, \text{vent}, \text{oximeter}\}$). we assumed the operational cycle period is 200 ms.

4.1 Safety Requirements

First of all, the safety requirements for the airway surgery must be defined. In plain English, it can be described as follows:

R1 To allow the usage of the airway-laser, the flow of the oxygen concentrate connected to the ventilator must be blocked at least for a half second.

R2 To open the flow of the oxygen concentrate, the laser must be completely deactivated.

R3 When measured SpO_2 value is lower than given threshold, Th_{SpO_2}, the flow of the oxygen concentrate through the ventilator must be open.

Recall that the safety requirements must be described as functions of the patient states and the supervisory modes of the device of each cycle. Notice that the requirements have some temporal relationship between the states of the devices: *e.g.* the half-second requirement between the operations in **R1**. Thereby, the supervisory modes of the device to describe the safety must encapsulate time information. As a result, some supervisory modes of some devices, depicted in Fig. 3, have time information: *e.g.* the time passed after oxygen-flow blocking is represented in some supervisory modes of the ventilator. With the given supervisory modes in the figure, the safety requirements listed above are given as equations, respectively, by

$$\big(mode_k^{(\text{laser})} = \langle\text{allow-usage}\rangle\big) \Rightarrow \big(mode_k^{(\text{vent})} = \langle\text{oxy-blocked>600ms}\rangle\big)$$
$$\big(mode_k^{(\text{vent})} = \langle\text{oxy-open}\rangle\big) \Rightarrow \big(mode_k^{(\text{laser})} = \langle\text{deactivated}\rangle\big)$$
$$[(\text{SpO}_2 < \text{Th}_{\text{SpO}_2})] \Rightarrow \big(mode_k^{(\text{vent})} = \langle\text{oxy-open}\rangle\big). \quad (7)$$

4.2 Supervisor core logic

The supervisor core logic reflects the effectiveness of the operation when the network is normal. The supervisor takes care of the two safety interlocks in the example: the interlocks between the airway laser and the ventilator and between SpO_2 and the ventilator. Since SpO_2 is not controllable by the supervisor, the later interlock must have higher priority. In the supervisory core, logic codes for multiple interlocks reside, and higher priority logic is executed earlier; SpO_2 interlock logic, given in Algorithm 1 is executed first, and then the laser interlock logic, given in Algorithm 2 is performed. Based on **Require** field of each algorithm, no assignment must have been performed before each interlock logic is called; *i.e.* SpO_2 interlock assigns any mode to either laser or vent and laser interlock logic will be skipped.

Notice that the supervisory core logic does not concern any network problem but is based on ideal network assumption. Recall that all the network issues are addressed by the framework, not the core logic.

Algorithm 1 SpO_2 supervision logic

```
Require: vent.$mode$ and laser.$mode$ have been unassigned
  if worst case $\text{SpO}_2$ in 1 s $< \text{Th}_{\text{SpO}_2}$ then
    if laser.$mode \neq \langle\text{deactivated}\rangle$ then
      laser.$mode \leftarrow$ next deactivation mode
    else
      vent.$mode \leftarrow \langle\text{oxy-open}\rangle$
    end if
  end if
```

Algorithm 2 Airway laser supervision logic

```
Require: vent.$mode$ and laser.$mode$ have been unassigned
  if $laser.reqstate = want\text{-}activation$ then
    if vent.$mode \neq \langle\text{oxy-blocked>600ms}\rangle$ then
      vent.$mode \leftarrow$next blocking mode
    else
      laser.$mode \leftarrow \langle\text{allow-usage}\rangle$
    end if
  end if
```

4.3 Network-fail-safe mode and supervision reshaping

One of the responsibilities of the NASS is to enforce the devices to have appropriate mode transition to the network-fail-safe mode when the network is disconnected. It is taken care of by the supervision reshaping layer. After supervisor core logic generates supervisory mode vectors, the supervision reshaping layer reshapes the vectors to add transitions to the network-fail-safe mode at the end of the vectors for the safety in network failures.

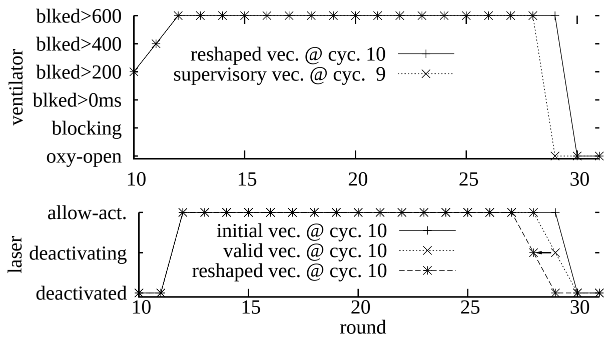

Figure 4: Supervisory vectors of airway laser and ventilator to demonstrate supervisory vector reshaping of airway laser

To reshape the vectors, the system must be aware of the network fail-safe modes of the devices, which are derived from the safety requirements. Once safety requirements are given in Boolean logic (Eq. (7)), the network-fail-safe modes are easily found from Eq. (6). In our case study, by assigning $mode_{\text{safe}}^{(\text{laser})} = \langle\text{deactivated}\rangle$ and $mode_{\text{safe}}^{(\text{vent})} = \langle\text{oxy-open}\rangle$ all safety requirements are resolved as network-fail-safe modes.

The reshaping layer must reshape vectors of the devices in a certain order, which is the reverse order of a desirable transition order of network-fail-safe mode.[4] In the example, the laser must be deactivated first, and then the ventilator opens the oxygen flow. Thereby, the reshaping is performed for the ventilator first, and then laser is taken care of later. This sequence dependency is denoted by vent $\rightsquigarrow$ laser and called *network-fail-safe mode dependency order.* This dependency order can also be found from the Boolean logic of the safety requirements as follows: Let us put each of the network-fail-safe modes into Eq. (7). $mode_{\text{safe}}^{(\text{laser})}$ makes no violation of the requirements, while $mode_{\text{safe}}^{(\text{vent})}$ can violate Eq. (7) when $mode_k^{(\text{laser})} = \langle\text{allow-usage}\rangle$. Then, we conclude to have vent $\rightsquigarrow$ laser, which means vent depends on laser in network-fail-safe mode safety.

Fig. 4 demonstrates a reshaping example of an airway-laser supervisory vector. The demonstrated situation is as follows. The current cycle is 10 and $\mu = 30$. The solid line of each graph represents the initial supervisory vector generated by the supervisory core with the last element being the network-fail-safe mode, $mode_{\text{safe}}$, of each device. According to the network-fail-safe mode dependency order, ventilator is supposed to be reshaped first. When reshaping, the layer checks (1) if the transition to from an initial vector to $mode_{\text{safe}}$ at the end of the vector is valid, and (2) if the transition does not violate the safety of the system. Since, the ventilator's supervisory vector satisfies both, it is remains unchanged at the supervision reshaping layer.

Then, the reshaping for the initial vector of the airway laser is performed. First of all the initial vector (dashed line

[4]The reason why the reshaping layer takes care of the vector in the reverse fail-safe-mode transition order is that the reshaping is performed from the tail of the vector back towards the front.

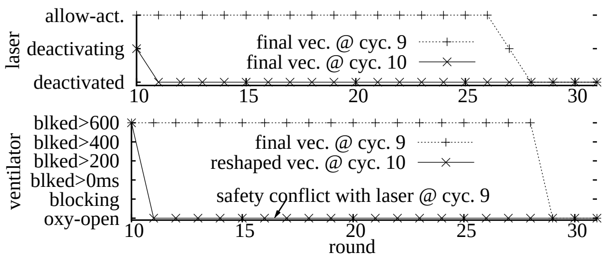

Figure 5: Supervisory vectors of airway laser and ventilator; the reshaped vector of the ventilator at cycle 10 is dropped

at the top of Fig. 4) has an invalid transition at the tail, from ⟨allow-activation⟩ directly to ⟨deactivated⟩. According to Fig. 3, there must be intermediate mode of ⟨deactivating⟩ as the dotted line of Fig. 4 (the second of the legend). However, putting intermediate mode potentially violates the safety of the system. If the network to the ventilator is disconnected at cycle 10, the ventilator is going to employ the supervisory mode plan transmitted at cycle 9 in all the future cycle, which is dotted line of ventilator graph. Then the ventilator is going to open the oxygen flow at cycle 29.

Let us combine this worst-case scenario with the new laser vector putting intermediate mode of ⟨deactivating⟩ at cycle 29. ⟨oxy-open⟩ of ventilator and ⟨deactivating⟩ of airway laser violates the safety. Therefore, when reshaping laser, the supervisory reshaping layer has to consider the worst-case scenario to reshape the tail of each supervisory vector. To eliminate all the potential safety violations, the tail transition of $\rightarrow$ ⟨deactivating⟩ $\rightarrow$ ⟨deactivated⟩ must be performed from cycle 28 as presented by the dashed line of the laser mode graph (last element of legend). Consequently, the reshaping layer takes care of the validity of transition by putting intermediate modes if necessary, and puts the transition to $mode_{\text{safe}}$ as late as possible but not to incur safety violation.

4.4 Safety filter

Even though the supervision reshaping layer takes care of the safety, it only takes care of the tail part of each vector that is reshaped for the network-fail-safe mode. The remainder of the vector generated by the supervisory core has not been fully certified to be safe, and safety filter is in charge of it. It inspects all the vectors from supervision reshaping layer, and filter out the vectors that are potentially unsafe.

Fig. 5 shows an example that the filter actually drops a vector in our airway-laser surgery example. The situation is as follows: Until cycle 9, the airway laser had been in use. Because the use has been completed, the supervisor, at cycle 10, decides to deactivate the laser and to open the oxygen flow. As a result, the supervisory plans, represented by the supervisory vectors, generated at cycle 9 and cycle 10 are entirely different as shown in Fig. 5; dotted lines and solid lines depict the vectors generated at cycle 9 and cycle 10, respectively. If devices employ the plan of cycle 9, the laser will be allowed to be activated, and the ventilator will block the oxygen flow between cycle 10 and cycle 26. If the supervisory vectors of cycle 10 are applied, the laser will be

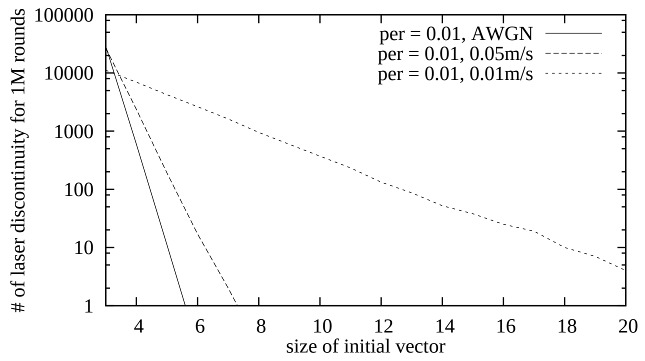

Figure 6: The number of occurrences of the network-fail-safe mode in the system for one million cycles; the size of initial vector means μ

deactivated, and the ventilator will resume the oxygen flow between cycle 11 and cycle 26.

Suppose that network of laser is only disconnected, while the connection to the ventilator is still good at cycle 10. Then, the laser will employ the plan of cycle 9, while the ventilator employs the plan of cycle 10, which is a serious safety violation of laser explosion. The reason of safety violation is that the supervisory vectors at cycle 10 are initially generated by the supervisory core, which does not consider the plan generated at the previous cycles. The safety filter examines the worst-case packet delivery combinations and detect this hazard. As a result, the reshaped vector of ventilator is filtered out and not delivered to the device. If the supervisory vector for the laser of cycle 10 is delivered successfully, the next supervisory vector for the ventilator to open the oxygen flow generated at cycle 11 will not be dropped again, managing original effectiveness of the supervisor core.

4.5 Evaluation of procedural effectiveness

Safety is not everything. We could design a system such that the airway laser is prohibited from operation. This system ensures the safety from airway fires, but is useless because the physician cannot perform the surgery. Thus, aside from safety, the supervisory system must also provide maximal effectiveness for the surgical procedure.

We show the effectiveness of the system through simulation and using a real testbed. First, we performed a couple of simulations to see how the NASS framework is affected by packet losses. The simulation is written in Python 2.6.2 implementing the NASS framework. The cycle period is 200 ms, and μ has a value from three to 20. The simulator assumes a wireless network suffering Rayleigh fading with average packet loss rate of 0.01 (Zorzi's Rayleigh fading model [25]). The speeds of the object and the environments are set to either 50 mm/s or 10 mm/s to reflect the medical environments; the slower the speed is, the more the packet losses collocate. The additive white Gaussian noise (AWGN) channel is also measured for comparison. In such a channel, all packet losses are independent of each other.

First, we measured how often the system falls into the network-fail-safe mode in terms of the wireless environments and the

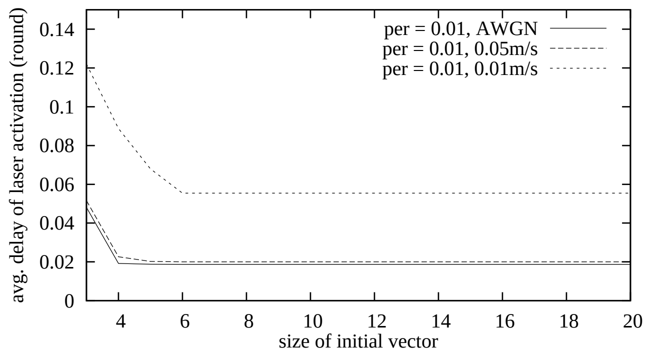

Figure 7: The average delay of airway laser activations due to packet losses; the size of initial vector means μ

value of μ. To measure it, the system is set to continuously turn on the laser and to block the oxygen flow. As a metric of effectiveness, we counted the number of discontinuities of the airway laser. The low SpO_2 case was intentionally neglected for the experiment. The result is depicted in Fig. 6 in log scale. When fading is really slow, the laser falls into the network-safe-mode, even with $\mu = 20$. Falling into a network-fail-safe mode is undesirable because it discontinues the procedure of the surgery. If a system is in wireless network, it is recommended to set the value of μ large enough because the surgical environment is very steady. Thus, the network tends to have collocated packet losses causing a sudden network-fail-safe mode.

The same tendency was measured in the prototype system. We have implemented a prototype of the NASS framework based on Java real-time system 2.2. Since an ICSS is a collection of the devices made by various manufacturers, such a property is profitable. We put the supervisor and the devices in 10 m distance, and measured the occurrences of the network-fail-safe mode for 9 hours when $\mu = 10$. We had one network-fail-safe mode caused by 11 consecutive packet losses with 0.24 % of packet drop ratio, which will rarely occur if packet losses is like a Poisson process.

Another side-effect of packet losses is the delay of operations. If packet losses occur in a steady state of a procedure, they are consumed with no effect to the procedure. However, if they occur at the transitions of supervisory modes, they cause some delays of operations. These delays are unavoidable because the packets for the fresh operations are lost on the way in such cases. We have simulated 100,000 times of airway laser activations in the same environment settings as the previous simulation. As shown in Fig. 7, the delays are not highly dependent on the value of μ but on the network environments because it is fairly inherent, so unavoidable.

5. RELATED WORK

Our work is motivated by the ongoing efforts of the Medical Device Plug-and-Play Interoperability program [8]. This program has been leading the development of the ICE standard and gap analysis work on the ability of the IEEE 11073 family of standards [21] to meet the clinical use cases described in the ICE standard.

This work is closely related to supervision of discrete event systems with partial observations [5,15]. There is an extensive survey of the area written by Lafortune [16]. Zad *et. al* presented a framework that generates a diagnosis supervisor from a given finite-state automata [24]. It is designed to generate a supervisor for each system. It is not easy to apply this framework to a dynamic environment such as an ICSS. Bhattacharyya *et. al* proposed a discrete event systems approach to detect and diagnose a network fault [4]. It mainly focuses on the diagnosis of the network itself.

Real-time network issues in distributed control system have been addressed with various approaches. Davidson et. al [6] addressed the issue of coordinating distributed system over a network in real-time by proposing Timed Atomic Commitment (TAC). This enables real-time coordination of distributed systems in either a centralized or a distributed manner. Moreover, Wang *et. al* introduced how to enhance the reliability of wireless networks using cell phone network paradigm for industrial controls [23], while Kottenstette *et. al* proposed how to develop feedback-control systems regarding the delays caused by a network [14].

Medical device integration is another emerging research area. Arney et. al demonstrated synchronization techniques of medical devices with an X-ray and a ventilator [3]. Fischmeister et. al. applied their work for the Network Code Machine [7] 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. [13] using publish-subscribe architectures for dynamic information flow. Currently, much of the work for medical device plug-and-play focuses on estabilishing dynamic connectivity of devices, device-to-device synchronization, and ensuring fair access to a communication medium, while our work focuses on providing safety constraints over unreliable networks.

On the other end of the spectrum, medical device safety has been a prevalent issue dating back to the infamous incidents in the 80's involving the Therac 25 radiation therapy machines [17]. Since then, much work has been done to apply formal methods to medical devices analysis [2,3,20]. The use of formal methods may even start to influence actual medical device review procedures [12]. 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.

6. CONCLUSION

The task of designing a safety-critical system on top of unreliable networks is highly nontrivial. Our Network-Aware Safety Supervision (NASS) framework solves this problem in two ways: *(a)* by providing designers with an abstraction of a completely reliable network and *(b)* by providing transformations of distributed device control vectors to ensure safety. Even though this framework uses a non-trivial pipeline planning method, all complications are hidden from the application logic.

Our design is guaranteed to be safe by pessimistic construction. To evaluate the practicality of our framework, a prototype and a simulator were implemented. We also showed how safety invariants are not violated in real networks and how the operational effectiveness is affected by network status in our framework.

This paper illustrates the NASS framework through an important example, *viz.* the airway-laser case study. In the future, we plan to consider other types of medical safety interlocks that will involve more devices. Also, we plan to consider plug-and-play safety where medical configurations may change during run-time.

7. ACKNOWLEDGMENTS

The authors would like to thank Dr. Julian M. Goldman, and Susan Whitehead with Center for Integration of Medicine & Innovative Technology (CIMIT), Sandy Weininger and Roul P. Jetley with U.S. Food and Drug Administration (FDA), and Abdullah Al-Nayeem with UIUC for their help and suggestions. This work is funded by National Science Foundation (NSF IIS 07-20482 and NSF CNS 08-34709).

8. REFERENCES

[1] A. Al-Nayeem, M. Sun, X. Qiu, L. Sha, S. P. Miller, and D. D. Cofer. A Formal Architecture Pattern for Real-time Distributed Systems. In *IEEE RTSS*, 2009.

[2] R. Alur, D. Arney, E. L. Gunter, I. Lee, J. Lee, W. Nam, F. Pearce, S. V. Albert, and J. 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.

[3] D. Arney, R. Jetley, P. Jones, I. Lee, and O. Sokolsky. Formal Methods Based Development of a PCA Infusion Pump Reference Model: Generic Infusion Pump (GIP) Project. In *Proc. of Joint Workshop on HCMDSS-MDPnP*, Jun. 2007.

[4] S. Bhattacharyya, Z. Huang, V. Chandra, and R. Kumar. A Discrete Event Systems Approach to Network Fault Management: Detection & Diagnosis of Faults. In *Proc. of American Control Conference*, 2004.

[5] R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Transactions on Automatic Control*, 33(3):249–260, 1988.

[6] S. B. Davidson, I. Lee, and V. Wolfe. Timed Atomic Commitment. *IEEE Transactions on Computers*, 40(5), May 1991.

[7] S. Fischmeister, O. Sokolsky, and I. Lee. A Verifiable Language for Programming Real-Time Communication Schedules. *IEEE Trans. on Comp.*, 56(11), Nov. 2007.

[8] J. 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.

[9] J. Goldmann. Medical Devices and Medical Systems — Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE) — Part 1: General requirements and conceptual model. ASTM 2761-2009, 2009.

[10] H. Grades. Fourth Annual Patient Safety in American Hospitals Study, 2007.

[11] 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.

[12] R. Jetley, S. P. Iyer, and P. L. Jones. A Formal Methods Approach to Medical Device Review. *IEEE Computer*, 39(4), Apr. 2006.

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

[14] N. Kottenstette, X. Koutsoukos, J. Hall, J. Sztipanovits, and P. Antsaklis. Passivity-based design of wireless networked control systems for robustness to time-varying delays. In *Proc. of IEEE RTSS*, Nov. 2008.

[15] R. Kumar and M. A. Shayman. Centralized and Decentralized Supervisory Control of Nondeterministic Systems Under Partial Observation. *SIAM Journal of Control and Optimization*, 35(2):363–383, Mar. 1997.

[16] S. Lafortune. On Decentralized and Distributed Control of Partially-Observed Discrete Event Systems. *LNCIS*, 353:171–184, 2007.

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

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

[19] R. S. Parker, I. Francis J. Doyle, and N. A. Peppas. A Model-Based Algorithm for Blood Glucose Control in Type I Diabetic Patients. *IEEE Transactions on Biomedical Engineering*, 46(2):148–157, Feb. 1999.

[20] A. Ray and R. Cleaveland. Unit verification: the CARA experience. *International Journal on Software Tools for Technology Transfer*, 2004.

[21] 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.

[22] S. Seyedtabaii and L. Seyedtabaii. Kalman Filter Based Adaptive Reduction of Motion Artifact from Photoplethysmographic Signal. *Proceedings of World Academy of Science, Engineering and Technology*, 27, Feb. 2008.

[23] Q. Wang, X. Liu, C. Weiqun, L. Sha, and M. Caccamo. Building Robust Wireless LAN for Industrial Control with the DSSS-CDMA Cell Phone Network Paradigm. *IEEE Transactions on Mobile Computing*, 6(6), June 2007.

[24] S. H. Zad, R. H. Kwong, and W. M. Wonham. Fault diagnosis in discrete-event systems: Framework and model reduction. *IEEE Transactions on Automatic Control*, 48(7):1199–1212, Jul. 2003.

[25] M. Zorzi, R. R. Rao, and L. B. Milstein. Error statistics in data transmission over fading channels. *IEEE Transactions on Communications*, 46(11):1468–1477, Nov. 1998.