LOW COMPLEXITY SYSTEM ARCHITECTURE DESIGN FOR MEDICAL CYBER-PHYSICAL-HUMAN SYSTEMS (CPHS)

BY

PO-LIANG WU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Doctoral Committee:

Professor Lui Sha, Chair
Professor Tarek Abdelzaher
Professor Alex Kirlik
Associate Professor Rahul Mangharam, University of Pennsylvania

Abstract

Cyber-Physical-Human Systems (CHPS) are safety-critical systems, where the interaction between cyber components and physical components can be influenced by the human operator. Guaranteeing correctness and safety in these highly interactive computations is challenging. In particular, the interaction between these three components needs to be coordinated collectively in order to conduct safe and effective operations. The interaction nevertheless increases by orders of magnitude the levels of complexity and prevents formal verification techniques, such as model checking, from thoroughly verifying the safety and correctness properties of systems. In addition, the interactions could also significantly increase human operators' cognitive load and lead to human errors.

In this thesis, we focus on medical CPHS and examine the complexity from a safety angle. Medical CPHS are both safety-critical and highly complex, because medical staff need to coordinate with distributed medical devices and supervisory controllers to monitor and control multiple aspects of the patient's physiology. Our goal is to reduce and control the complexity by introducing novel architectural patterns, coordination protocols and user-centric guidance system. This thesis makes three major contributions for improving safety of medical CPHS.

C1 **Reducing verification complexity:** Formal verification is a promising technique to guarantee correctness and safety, but the high complexity significantly increases the verification cost, which is known as state space explosion problems. We propose two architectural patterns: *Interruptible Remote Procedure Call (RPC)* and *Consistent View Generation and Coordination (CVGC)* protocol to properly handle asynchronous communication and exceptions with low complexity.

C2 **Reducing cyber-medical treatment complexity:** Cyber medical treatment complexity is defined as the number of steps and time to perform a treatment and monitor the corresponding

physiological responses. We propose treatment and workflow adaptation and validation protocols to semi-autonomously validate the preconditions and adapt the workflows to patient conditions, which reduces the complexity of performing treatments and following best practice workflows.

C3 **Reducing human cognitive load complexity:** Cognitive load (also called mental workload) complexity measures human memory and mental computation demand for performing tasks. We first model individual medical staff's responsibility and team interactions in cardiac arrest resuscitation and decomposed their overall task into a set of distinct cognitive tasks that must be specifically supported to achieve successful human-centered system design. We then prototype a medical Best Practice Guidance (BPG) system to reduce medical staff's cognitive load and foster adherence to best practice workflows. Our BPG system transforms the implementation of best practice medical workflow

- From reliance on human memory, verbal communication, notes and disparate data sources
- To a real-time integrated workflow-driven display, action coordination, order checking and tracking and patient response feedback system

Acknowledgments

I acknowledge the contributions of the team members. I am particularly grateful to my advisor Dr. Lui Sha not only for providing me the right guidance throughout this Ph.D. program, but also for his advice for my future career and life. I am also grateful to Dr. Richard Berlin. His advice and suggestions from a physician's point of view is essential for making clinical impact. Thanks also go to Carle Foundation Hospital for their contributions to this research. For graduate students, changing the field of study is always challenging. I would like to thank Dr. Woochul Kang and Dr. Abdullah Al-Nayeem for their help when I changed my field of study to healthcare systems. I thank Dr. Min Young Nam, Dr. Jeonghwan Choi, and Maryam Rahmaniheris for their helps and suggestions throughout my Ph.D. program. I would also like to thank Dr. Tarek Abdelzaher, Dr. Alex Kirlik, and Dr. Rahul Mangharam for being in my thesis committee and for their helpful suggestions and advices.

I will always cherish the memories of my stay in Champaign-Urbana. I am thankful to all my friends for being with me. Finally, I thank my parents, A-Jen Wu and Suh-Yuh Chen, for their love and support. It is their love and support that keep me cheerful and motivated. I also want to thank my girlfriend, Pei-Chen Peng, for bringing happiness to my life every day.

Table of Contents

Chapter 1

Introduction

Cyber-Physical-Human systems (CPHS) are systems that tightly combine computation resources, physical elements and human operators through communication technologies. Applications of CPHS include high confidence medical systems, avionics systems, and nuclear plants, etc [57]. In each of these contexts, all three components contribute collectively to safe and efficient operations. In this thesis, we focus on the complexity and safety issues of medical Cyber-Physical-Human Systems (CPHS). In 1999, a report from Institute of Medicine (IOM) suggested that "Health care in the United States is not as safe as it should be–and can be. At least 44,000 people, and perhaps as many as 98,000 people, die in hospitals each year as a result of medical errors that could have been prevented" [54]. Unfortunately, the challenge of preventable medical errors persists. According to the Subcommittee on Primary Health and Aging in a July 2014 hearing, "preventable medical errors persist as the No. 3 killer in the U.S. third only to heart disease and cancer claiming the lives of some 400,000 people each year" [67]. High complexity of healthcare environments is a major contributor for preventable medical errors [54, 81]. Without properly controlling and reducing complexity, systems with high complexity may even cause more safety hazards [78, 92, 61]. As National Science Foundation has remarked,

> *"These advances have not only made it possible to reach the frontier faster; they have also increased by orders of magnitude the levels of complexity open to exploration and experimentation. Understanding complexity and learning how best to harness these new capabilities are both a challenge and a responsibility."* [5]

From a computer science perspective, complexity reflects the workload of performing tasks by measuring number of steps or elapsed time. For example, in computation theory, time complexity is usually approximated by the asymptotic notation of the number of steps. In this thesis, we examine complexity from a safety angle, identify three major types of complexity, and propose complexity reduction and control mechanisms to address them. Followings are the three types of complexity.

- **Verification complexity:** Verification complexity can be measured by model checking state space. Verification complexity of a system has two parts. First is the complexity of each component itself and second is the complexity of the interaction between components. Interactive complexity is a major contributor in the accidents of complex systems [81]. This is especially true in medical systems, where each components are certified by Food and Drug Association (FDA) but the configuration of system components and their interactions is outside of the scope safety certification. The main reason is that there are too many possible combinations of medical devices, and it is impossible for FDA to certify each combination given difference patient conditions. To appreciate the interplay between safety and system interaction, consider the case of Laser Airway Surgery, which vividly illustrates that fatal accidents can be caused by incorrect interactions of certified components even in a simple setting.

  Example Under anesthesia a patient needs oxygen enriched air. Unfortunately, the human tissue in the airway becomes flammable under oxygen enriched air. Surgical fire in the airway would start when both the laser is on and the oxygen supply to enrich the air is on. However, withholding oxygen enriched air for too long would lead to brain damage. Thus, incorrect coordination between the laser operation and oxygen supply would lead to preventable medical errors. What makes this seemingly easy coordination difficult is that the computer and network to coordinate the activity may fail randomly. And there is no automatic coordination that has been approved by FDA yet. Hence, the coordination has been done by human operators. Unfortunately, to err is human. The American Society of Anesthesiologists estimated that roughly 100 such fires occur each year causing roughly 20 serious injuries [65][1].

  In order to mitigate safety hazards, all the possible interactions between the system components

[1] Readers who are interested to known when this is not simple problem and what are the solutions may read [50]

must be thoroughly verified. The high interactive complexity nevertheless causes state space explosion problems. One of the major sources of interactive complexity is message interleaving due to asynchronous communications[2]. In CPHS, the components are physically distributed and communicate asynchronously with each other. However, asynchronous communication may create combinatorial possible message interleaving, because a sender can send messages at anytime and a receiver may receive the message with any order. From a verification perspective, all the message interleaving must be thoroughly verified to guarantee that the incorrect and unsafe states are not reachable. On the other hand, concurrency control of distributed devices and controllers is another major source of interactive complexity. The combinatorial possible execution sequences of all the system components should be verified for guaranteeing safety. In medical CPHS, one of the main reasons for concurrency control of multiple devices and controllers is to handle exceptions, such as patient adverse events. Most exceptions are raised asynchronously by distributed medical devices at different time. In order to safely handle an exception, physicians or supervisory controllers require the timely and consistent view of patient condition and device status. Moreover, the devices must be properly coordinated to avoid potential adverse devices interactions.

- **Cyber medical treatment complexity:** Cyber medical treatment complexity is defined as the number of steps and time to perform a treatment and monitor the corresponding physiological responses. Several steps are generally involved to perform a treatment, including deciding the best fit treatments, checking preconditions, administering, monitoring side effects, and checking expected responses[3].

  Example If a patient suffers from cardiac arrest, physicians usually consider administering epinephrine to improve the patient's cardiac output. Before administering epinephrine, physicians need to check the preconditions of epinephrine, e.g. patient's blood pH should be larger than 7.2. After administering epinephrine, physicians need to check if the patient develops

[2]Asynchronous communications mean that a sender can initiate transmission at anytime and sporadically send messages without waiting for the responses from a receiver. The receiver may receive the messages at anytime with any order. Email and bulletin-boards system are two well-known asynchronous communication examples.

[3]For certain complicated treatments, such as surgery, additional steps may be required.

pulmonary dysfunction (side effect) and the patient's cardiac output is improved (expected response). If the patient does not response to epinephrine and/or the side effects of epinephrine adversely affect patient conditions, physicians need to consider alternative treatments.

However, in an intensive care unit (ICU), patients are seriously ill, and there is often short of time and short of information for medical staff to make decisions. Moreover, medical staff are under tremendous pressure and overloaded by the great amount of unorganized information. A 2006 study [7] suggested that medication errors are among the most common medical mistakes, harming at least 1.5 million people every year. According to the study, there are more than 400,000 preventable drug-related injuries in hospitals every year. Medical complications may arise if any treatment is not performed correctly, which significantly increase the complexity, because more corrective treatments must be performed.

- **Cognitive load complexity:** Cognitive load (also called mental workload) complexity measures human memory and mental computation demand for performing tasks. In the current medical practice, medical staff are usually overwhelmed by the great amount of unorganized physiological data. A Canadian study identified that more than 1,300 data points are generated per patient per day of ICU stay, an increase in 26% over 5 years [64]. Furthermore, related physiological information are usually spread across multiple devices and screens. Medical staff need to mentally gather and correlate all the physiological information, which considerably increases cognitive load and the likelihood of medical errors. We have identified the four major cognitive tasks: clinical information assembly, recalls, real-time tracking, and calculation. We believe that the above cognitive tasks significantly contribute to medical staff's cognitive load and are generic to many medical scenarios. The current display systems lack of context-dependent information and guidance, such as workflows, to reduce medical staff's cognitive load and help them follow best practice guidelines. Many preventable medical errors are result from unintended deviation[4] from the medical workflows [54]. Unintended deviations further increase the medical staff's cognitive load, because medical staff must perform more treatments to either recover from the deviation or

[4]Unintended deviation is the medical staff unintentionally perform treatments noncompliance with the guidelines. Deviation may adversely affect patients' outcomes or even cause patients' death.

switch to an alternative workflow.

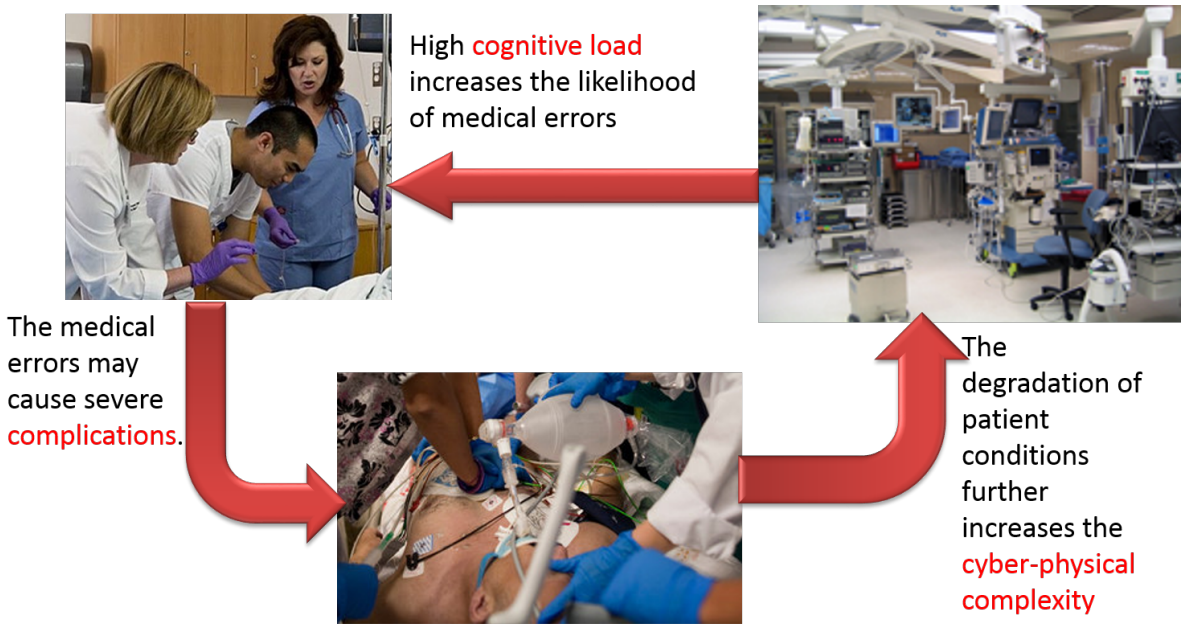


Figure 1.1: The interaction between the three types of complexity

Moreover, those three types of complexity could interact with each other and form a vicious cycle, as described in Figure 1.1. The high cognitive load imposed in such stressful and rapidly changing medical environments significantly increases the likelihood of medical errors. Those medical errors may cause deterioration of patient conditions and severe medical complications. Therefore, physicians and nurses need to perform additional treatments to treat those complications. Since more treatments must be performed and more patient conditions must be monitored, the cognitive load of medical staff is further increased. It is worth mentioning that the verification complexity is also increased because the medical complications and treatments introduce more concurrent events to the supervisory system. As described before, all the possible interleaving between concurrent events must be thoroughly verified, which makes the verification complexity grow exponentially.

In order to reduce preventable medical errors, there is an ongoing effort to enable supervisory control among distributed medical devices. For example, Medical Device Plug-and-Play (MDPnP) [37] was proposed to collect and consolidate medical information, and allows the reconfiguration of medical devices. Integrated Clinical Environment (ICE) infrastructure is designed to integrate networked medi-

cal devices using MDPnP criteria and provides logically centralized coordination and control [38]. We argue that such an integrated infrastructure opens a door for reducing preventable safety hazards, but it also significantly increases the complexity of the system due to the interactions of devices, controllers, patients and medical staff. Therefore, we need new breed of system designs, which include architectural design patterns, protocols and user-centric integrated display, to reduce and control the three types complexity of medical cyber-physical-human systems. We first addressed the verification complexity problem by proposing architectural patterns and communication protocols for achieving *consistent coordination* with low complexity [97, 93, 49] Second, we addressed the cyber medical treatment complexity by developing treatment and workflow validation protocols to help medical safely perform treatment and adapt workflows to adverse patient conditions [95, 96]. Third, we addressed the cognitive load complexity problem by developing a Best Practice Guidance (BPG) system to keep track of workflow steps, flag deviations from the workflows, and highlight in real-time the progress and status of medical treatments, as well as provide ready access to the patient's physiological responses to these treatments. As a result, the medical staff's cognitive load is reduced, which leads to better adherence to the best practice medical guidelines. As the ICU director of Carle Foundation Hospital has remarked,

> *"It is expected that the use of the system will result in rapid and consistent timing of medical interventions, stricter adherence to standardized medical treatment guidelines, more accurate record keeping, and improved team situation awareness"*

1.1 Related Work

Complexity reflects the workload of performing tasks by measuring the number of steps and/or elapsed time. In computation theory, time and space complexity are used to evaluate the time and space take by an algorithm to generate the output as a function of input length [85]. In order to evaluate system or software complexity, Card *et al.* proposed to evaluate the software design complexity by considering the module size, fan-in, and fan-out, etc [23]. Cyclomatic complexity [66] is used to quantitatively measure the complexity of a program by calculating the number of linearly independent paths. Chidamber *et al.* introduce six object-oriented complexity metrics: weighted methods per class, coupling between object classes, response for a class, number of children, depth of inheritance tree and lack of

cohesion of methods [25]. Ranganathan *et al.* defined the system complexity with five aspects: task structure complexity, unpredictability, size complexity, algorithmic complexity, and chaotic complexity [82]. From a human factor perspective, cognitive load complexity can be measured by a set of metrics, including response time, accuracy, and NASA-TLX [88, 45]. In this thesis, we address three types of complexity, verification complexity, cyber medical treatment complexity and cognitive load complexity, and present a set of evaluation metrics to quantitatively measure the complexity.

Several complexity reduction techniques are proposed for different application domains. Atomic transaction [39] is widely used in the database system to reduce the interleaving of transactions. Some techniques are proposed to reduce the verification complexity, such as abstraction [29] and assume-guarantee framework [44]. Researchers have also proposed various architectural patterns to manage the system complexity and improve the reliability. Simplex architecture [84] is based on the principle of "using simplicity to control complexity." The system dynamically switches to a simple safety controller subsystem when the stability of the controlled system is threatened. The verification of the system safety and stability is only applied to the safety controller and the decision logic. Some architectural patterns and solutions are proposed to reduce complexity by implementing logical synchrony over an asynchronous computation architecture, e.g. PALS [2], TTA [83]. The developers can design and verify the system as if the system components were driven by a global clock, which considerably reduces the verification cost. In this thesis, we develop a set of mechanisms reduce verification complexity, cyber medical treatment complexity, and cognitive load complexity. An organ-based architecture and consistent view generation and coordination protocol are developed to reduce verification by providing consistent control and limiting asynchronous messages. A treatment validation protocol is proposed to reduce cyber medical complexity by checking preconditions, monitoring side effects, and checking expected responses. A workflow-driven display system is developed to organize physiological information based on the medical context, so medical staff's cognitive load can be reduced.

Medical Device Plug-and-Play (MDPnP) provides flexibility and interoperability for medical systems [38]. However, the dynamic system configurations introduce serious challenges to medical systems. Some frameworks are proposed to guarantee safety under closed-loop and open-loop control [50, 6]. Some mechanisms are also used to improve the accuracy and the safety of the systems, such as fuzzy logic [87], and information technology [12]. Some work aims to improve the safety and

reliability of medical devices [58]. One promising technique is formal method, which is widely used to specify and verify medical devices [76, 3, 77]. Pajic *et al.* proposed an verification approach, which combines simulation-based analysis and model checking, to guarantee safety properties of closed-loop medical systems [77]. King *et al.* [52] proposed a formal specification language to express and reason safety properties of on-demand medical systems.

A comprehensive review of computer-supported cooperative work (CSCW) research conducted in healthcare is provided by Fitzpatrick *et al.* [36]. Several studies have shown that sense-making and situation awareness are critical in healthcare [10, 20, 56, 79]. Bossen *et al.* [20] conducted an ethnographic study in order to understand how physicians achieve a sufficiently informed, accountable and coherent understanding of a situation. Paul *et al.* [79] examined the cooperative sense-making of a medical team in a hospital emergency department. Hajdukiewicz *et al.* [41, 41] applied work domain analysis and structured medical knowledge in a hierarchical manner. Their work established a flexible and comprehensive framework for designing human-computer displays. Nemeth *et al.* [72] presented a set of methods to efficiently study a complex, high hazard healthcare environment. Our work has adopted several presented methods, including interviews, artifact analysis, rapid prototyping, and evaluation, to design and improve our medical Best Practice Guidance system. Kusunoki *et al.* [56] studied how vital signs monitors support teamwork during trauma resuscitation and provided some suggestions to improve the monitors. One particular suggestion of theirs that we adopted in the system design is in "providing rich contextual information." Specifically, the developed system provides guidance given contextual information on prescribed workflow and the patient's physiological responses.

Medical workflows aim to help medical staff adhere to the medical guidelines and correctly perform the medical procedures. Some work suggests that workflows can capture clinical information in more structured formats and support clinical practice if the workflows are properly designed and validated [16, 40, 21]. There are several workflow systems developed for efficient and safe execution of medical procedures [86, 90]. Song *et al.* classified healthcare workflows, including the approaches, goals, and major characteristics, and discussed application issues and software challenges [86]. Wang *et al.* [90] proposed a Workflow Intuitive and Formal Approach (WIFA) to model and analyze response time of emergency healthcare workflows. However, in spite of the potential clinical improvement based on medical workflows, some research work identifies the disparities between the formal models (work-

flows) and actual work practices [19, 43]. Bossen conducted an ethnographic case-study on the use of electronic health records and stated that the medical staff "experienced a fragmentation of patient cases and a critical lack of overview of patient treatment and care" [19]. Hartswood *et al.* also argued that there are discrepancies between the role of electronic records and the ways that medical staff actually uses and communicate information in clinical practices. On the other hand, this research work also pointed out that by establishing strong working relations between the system designers and the users [19] and adopting a user-led design methodology [43], the gap between IT technologies and clinical practices could be bridged. Consequently, our system design methodology is user-centered, and we continually involved the users in the deign process.

In medical contexts, high cognitive load due to disorganized information and a stressful environment poses significant challenges for medical staff to maintain an overall awareness and may further result in safety hazards [54]. Some research work focuses on analyzing medical staff's cognitive work [47, 71] and decision making process [56]. Jiancaro *et al.* [47] provided a comprehensive review on cognitive work analysis in healthcare domain. Burns *et al.* [22] applied CWA to cardiac nursing coordinators. Nemeth *et al.* [73, 71] described how the analysis of cognitive artifacts can reveal individual and group cognitive work. On the other hand, a wide range of research has been done for integrating physiological information in order to improve medical team's situation awareness [63, 1, 11]. Kimura system [63] was developed to provide users with an awareness on the progress on their work and display background activities on peripheral large displays. The AWARE [1] display system was developed to prioritize patient's physiological data and present to medical team in order to reduce task load and medical errors. However, displaying information to which an operator can attend and perceive is important, but is insufficient if it does not also foster operator comprehension and understanding [59]. To the best of our knowledge, little work on the design of context-dependent display systems for medical team has been reported. In this work, we developed a workflow-driven display system to integrate physiological measurements as well as context-dependent information based on both medical workflows and patients' physiological conditions.

1.2 Technical challenges

This thesis identifies the major technical challenges for designing safe medical cyber-physical-human systems with low complexity:

Challenge 1. In medical cyber-physical-human systems, asynchronous messages are common due to interrupts and exceptions, but they significantly increase verification complexity: In medical CPHS, two types of communication patterns can be considered for the control of devices: *synchronous communication* and *asynchronous communication*. In synchronous communication, supervisory controls are performed in a call-and-block manner, and, hence, the interleaving of individual messages is limited. However, synchronous communication is inappropriate for timely processing of urgent events, such as patient adverse events (PAEs). For example, patient adverse events, such as SpO_2_low, are generated asynchronously, and the supervisory controller must be blocked until all previously sent messages are handled by the devices. The blocking time may cause unacceptable delays for the handling of urgent events. In the asynchronous communication, in contrast, the system can progress without blocking, which enables arbitrary message flows. Asynchronous communication is usually used to handle urgent and unexpected messages. However, message interleaving of asynchronous interrupts or exceptions may significantly increase the state space for verification, since all the combinatorial possible interleaving must be verified. Our experiment in Chapter 2 shows that the model checking tool, must explore more than 28,000 states to verify a system with only 3 devices. Moreover, the state space grows exponentially as the number of sensors and queue size increase and quickly makes the model checking tool run out of memory. Therefore, verifying safety properties is like searching for needles in a haystack because of the state-explosion problem.

Proposed solutions: In Chapter 2, we propose a communication protocol, *Interruptible Remote Procedure Call (RPC)* [97], to reduce verification complexity due to asynchronous message interleaving. The *Interruptible RPC* exploits the tradeoff between asynchronous and synchronous RPC and limits the asynchronous messages, which prevents combinatorial possible message interleaving. Based on the proposed protocol, we further develop a composition method for designing complex hierarchical command and control systems while maintaining low complexity.

Challenge 2. Concurrency control of distributed devices and controllers result in complicated

race conditions, which in turn may cause safety hazards:

There are growing demands to utilize medical devices connectivity and interoperability in order to improve the effectiveness of medical services and patient safety [37, 38]. However, ad-hoc integration of medical devices through networking can significantly increase the verification complexity. The increasing number of devices and hierarchical control layers leads to more complicated race conditions between devices and controllers. Therefore, modern model checking tools, may quickly run out of memory and cannot thoroughly verify the safety and correctness properties. Moreover, the distributed system components must be properly coordinated, which is known as concurrency control, to prevent conflict/adverse interactions. For example, if multiple controllers command the same infusion pump without proper coordination, the infusion pump may give the drugs that can cause adverse interactions, such as sodium bicarbonate and calcium chloride[5]. Therefore, we need a low complexity architectural pattern for consistently control distributed devices and controllers and prevent potential conflicts.

Proposed solutions: In Chapter 3, we propose an architectural pattern, which consists of *Hierarchical Organ-based Clustering Architecture* and *Consistent View Generation and Coordination (CVGC) Protocol* [93]. The organ-based hierarchical architecture represents basic human physiology, human homeostasis[6]. Therefore, patient adverse events can be locally handled by different organ-system controllers, which improves the system effectiveness. Moreover, the separation of concerns for different layers of controllers enables modular verification. Based on the proposed architecture, *CVGC* protocol is developed to mitigate safety hazards by coordinating distributed medical devices and controllers. *CVGC* protocol generates consistent view and prevents unsafe race conditions among distributed devices and controllers, which in turn reduces the verification complexity.

Challenge 3. Medical complications may occur if the medical staff incorrectly perform treatments and improperly adapt workflows to patient conditions

As mentioned in the previous section, medical staff in ICU are usually under great stress and have very little time to make critical decisions. Slips and lapses of performing treatments may adversely affect

[5]Sodium bicarbonate and calcium chloride cannot be simultaneously injected through the same IV pump, because they would form calcium carbonate and make the two drugs ineffective.

[6]Human homeostasis is the body's ability to physiologically regulate its inner status to ensure the stability in response to the physiological changes.

patients' outcomes or even cause patients' death. Medical workflow codifies best practice guidelines to help physicians timely and correctly perform treatments. In medical CHPS, synchronizing supervisory medical systems, physicians' behavior and patient conditions in compliance with best practice workflow is essential for patient safety. However, patient conditions change rapidly and asynchronously. Therefore, the medical workflow must be adapted to patient conditions. For instance, if patient's heart rhythm changes from ventricular fibrillation to sinus bradycardia, a different set of treatments, including administering atropine and placing a pacemaker, should be considered. Nevertheless, adapting a workflow without validating safety requirements may cause safety hazards. For one thing, some treatments may cause severe adverse interactions if they are performed simultaneously. For another, at certain workflow steps, the workflow is not safe to be interrupted.

Proposed solutions: In Chapter 4, a *treatment validation protocol* is developed to help medical staff correctly perform treatments and monitor patient's physiological responses [95]. Specifically, the developed protocol validates treatments by checking preconditions, monitoring potential side effects, and checking patient responses, which can assist medical staff to perform treatments in accordance with medical guidelines. In addition, like model-based feedback control system, the proposed protocol validates the preconditions and corrective treatments and provides feedback to the medical staff. In Chapter 5, we proposed a *workflow adaptation and validation* protocol to dynamically adapt the workflow to the patient conditions while validating safety requirements in collaboration with physicians [96]. From a system verification perspective, we use a model checking tool, UPPAAL [7], to formally verify the safety and correctness of the proposed two protocols.

Challenge 4. Lack of context-dependent physiological display considerably increases medical staff's cognitive load complexity and, further, the chance of medical errors:
In the current medical practice, medical staff are generally overwhelmed by the great amount of unorganized data. Related physiological information are usually spread across multiple devices and screens. This situation significantly increases medical staff's cognitive load and the chance of preventable medical errors. In addition, the current display systems lack of medical context-dependent information, such as workflow, to assist medical staff follow the medical guidelines. Unintended deviation from

[7] In Appendix A, we give a brief introduction of model checking and UPPAAL. Readers, who are not familiar with AADL, are encouraged to read the chapter in the appendix.

the best practice medical guidelines may adversely affect patients' outcomes or even cause patients' death. However, in medical practice, medical staff must recall the steps in the guideline, verbally order medications, check patient's physiological measurements displayed across multiple monitors, diagnose patient, and decide treatment plans. Moreover, certain treatments are time sensitive, and medical staff are required to keep track of the time progress as well. For example, CPR should be sustained for at least two minutes, and epinephrine should be injected every three to five minutes. If the guideline is not followed correctly or the medication order is missed, patient's safety may be compromised. Therefore, a situation awareness improved display system is required to provide context-dependent physiological information and flag the deviations in order to reduce medical staff's cognitive load.

Proposed solutions: In Chapter 6, we developed a **Best Practice Guidance** (BPG) system to reduce medical staff's cognitive load and foster adherence to the best practice guidelines in real-time [94]. Our system 1) keeps track of the workflow steps, 2) flags deviation from the workflow, and 3) highlights the real-time progress of the treatments, the status of the treatments, and the patient physiological responses. Based on the context information provided by the workflow, the medical staff can gather concise and comprehensive physiological information to diagnose patients and decide treatment plans. We qualitatively and quantitatively evaluated the medical BPG system in a clinical simulation environment with a medical SimMan 3G Manikin[8], and the preliminary results demonstrate promising clinical effectiveness.

[8]SimMan 3G is an advanced patient simulator that can display neurological symptoms as well as physiological. `http://www.laerdal.com/us/SimMan3G`

Chapter 2

Reducing Verification Complexity: Interruptible Remote Procedural Call

As described in the previous chapter, message interleaving due to asynchronous communication is one of the major sources for verification complexity. In this chapter, we focus on the design methodology of providing formal description of the basic structure for composing complex systems, called building block, which is flexible, easy to extend, and only introduces low complexity and verification cost (Challenge 1). In order to achieve the design goal, we present an architectural design pattern – *interruptible RPC* [97] – for verifiable command and control systems (Challenge 1).

2.1 Synchronous and Asynchronous System Model

To quantify the system complexity in terms of the number of states, we first define a formal model of computation and communication for distributed command and control systems.

2.1.1 Synchronous system

The computation and communication model of the synchronous system is similar to the hand-shake synchronization model of UPPAAL [15].

A single FSM M_i is a tuple $\langle S_i, l_i^0, \Sigma, E_i \rangle$ where

- S_i is a set of locations of M_i,
- $l_i^0 \in S_i$ is the initial locations,
- Σ is the alphabet, whose elements are input ($a?$), output ($a!$), and/or local (a) actions, and
- $E_i \subseteq S_i \times \Sigma \times S_i$ is the set of edges.

In the above definition and the later description, a stands for an event passing between machines encapsulating the source and the destination. We write $l_i \xrightarrow{\sigma} l_i'$ for a state transition such that $\langle l_i, \sigma, l_i' \rangle \in E$, $l_i \in S_i$ and $l_i' \in S_i$ where $\sigma \in \Sigma$.

A synchronous system is a set of finite state machines $M_1, ..., M_n$. A state in the system is defined as a vector of current locations of the machines and denoted $\vec{l}$. There are two transition rules in the system: local transition rule where one FSM make its own move, and synchronized transition rule where two FSMs make simultaneous move. In the latter case, the two FSMs are synchronized using input actions, and output actions. Let l_i is ith element of vector $\vec{l}$ and $\vec{l}[l_i'/l_i]$ represents l_i is replaced by l_i'. The transition rules are as follows:

- *Local*: $\vec{l} \xrightarrow{a} \vec{l}[l_i'/l_i]$ if $l_i \xrightarrow{a} l_i'$
- *Sync*: $\vec{l} \xrightarrow{a} \vec{l}[l_i'/l_i][l_j'/l_j]$ if $l_i \xrightarrow{a!} l_i'$ and $l_j \xrightarrow{a?} l_j'$

2.1.2 Asynchronous system

In asynchronous model, a system is composed of a set of FSMs communicating through queues. we introduce queues to model the communication delay in the distributed system. Let $Q_{i,j}$ denotes a queue connecting from M_i to M_j. The queue is served as an unidirectional communication channel. $q_{i,j}$ represents an instance of $Q_{i,j}$, having a sequence of events. If a queue is a concatenation of two event sequences q, q', it is represented by $q \cdot q'$. Then, the distributed system can be defined as a parallel composition $M_1|...|M_n|Q_{i_1,j_1}|...|Q_{i_m,j_m}$ of a set of finite state machines $M_1, ..., M_n$ and queues $Q_{i_1,j_1}, ..., Q_{i_m,j_m}$. A queue represents the sequence of messages sent by the originator, M_{ik}, but has not been processed yet by the destination, M_{jk} where $1 \leq k \leq m$. The state of a distributed system is a pair $\langle \vec{l}, \vec{q} \rangle$ where $\vec{l}$ denotes a vector of current locations of machines, and $\vec{q}$ is a vector of

assignments of the queues in the network. q_{ij} stands for an element of $\vec{q}$ connecting M_i and M_j. Let l_i is *i*th element of vector $\vec{l}$ and $\vec{l}[l'_i/l_i]$ represents l_i is replaced by l'_i. Also, let q_i and $q[q'_i/q_i]$ are defined in the same way. Then, the transition rules are *local transition*, *send transition*, and *receive transition* as defined in the following:

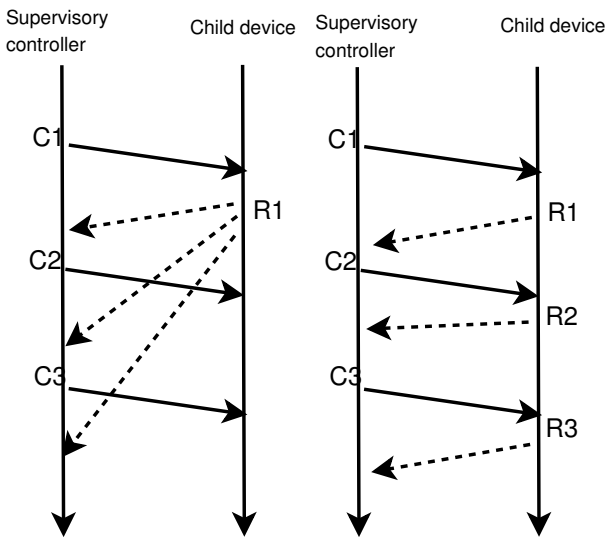
- *Local*: $\langle \vec{l}, \vec{q} \rangle \xrightarrow{a} \langle \vec{l}[l'_i/l_i], \vec{q} \rangle$ if $l_i \xrightarrow{a} l'_i$
- *Send*: $\langle \vec{l}, \vec{q} \rangle \xrightarrow{a!} \langle \vec{l}[l'_i/l_i], \vec{q}[q'_{ij}/q_{ij}] \rangle$ if $l_i \xrightarrow{a!} l'_i$, $q'_{ij} = q_{ij} \cdot a$
- *Receive*: $\langle \vec{l}, \vec{q} \rangle \xrightarrow{a?} \langle \vec{l}[l'_i/l_i], \vec{q}[q'_{ij}/q_{ij}] \rangle$ if $l_i \xrightarrow{a?} l'_i$, $q_{ij} = a \cdot q'_{ij}$

2.2 Communication Patterns and Message Interleaving

Fig. 2.1 shows the conceptual difference between asynchronous RPC, synchronous RPC. In Fig.2.1(a), response $R1$ can be interleaved with multiple commands: $C2$ and $C3$ due to asynchronous communication. The message interleaving causes exponential state space growth since all the combinatorial possible message interleaving must be verified. On the other hand, in synchronous communication, as shown in Fig. 2.1(b), responses never interleave with any commands. Therefore, there's no additional state space growth caused by message interleaving.

The complexity of message interleaving can be shown as the number of states explored by the model checking tools. We can model asynchronous communication between any two finite state machines (FSMs) using two directional queues. Let $S(Q_{k,x})$ be the total number of states of a queue, where k is the size of the queue and x is the number of distinct messages in the queue ($k \geq 1, x \geq 1$). Then, $S(Q_{k,x}) = 1 + x + x^2 + ... + x^k = \frac{x^{k+1}-1}{x-1}$ in worst case. If there are c distinct commands and r distinct responses, then the maximum number of states of the two queues can be $(\frac{c^{k+1}-1}{c-1}) \cdot (\frac{r^{k+1}-1}{r-1})$ assuming there is no order enforcement between the commands and responses. Exponential number of states of the queues represent the maximum possible number of message interleaving which is a major source of state explosion in distributed systems.

However, asynchronous communication is widely used in supervisory command and control systems for handling urgent situations. A completely synchronous communication design may not be



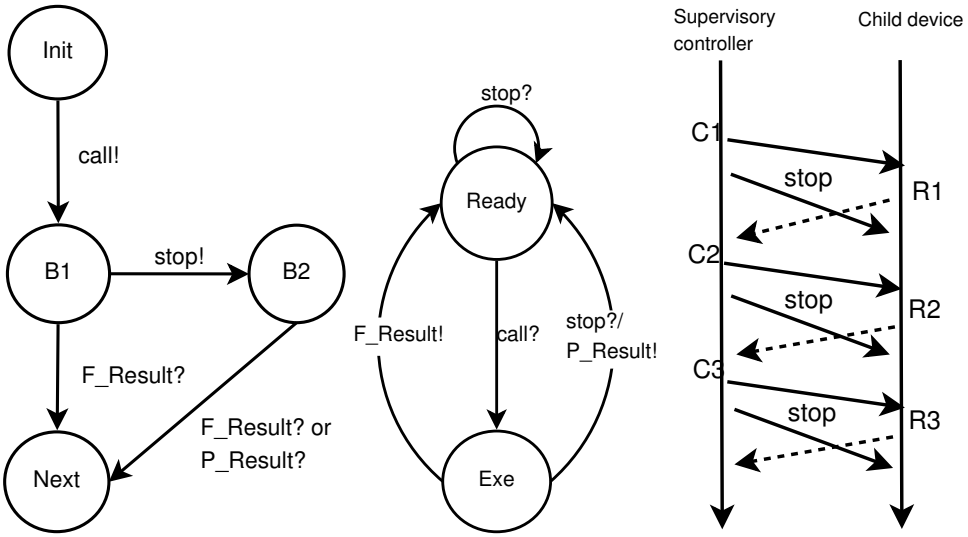
(a) Asynchronous (b) Synchronous

Figure 2.1: Message Interleaving

applicable to many systems. A new communication protocol, therefore, is required to reduce complexity with the consideration of the tradeoff between flexibility and simplicity.

2.3 Fundamental Protocol Design

The system architecture follows the supervisory command and control scenario. The system consists of a supervisory controller and a set of child devices, including sensors and actuators. Fig. 2.2 shows our proposed interruptible RPC pattern. The syntax of the model is described in Table 2.1. The basic scenario follows the synchronous RPC pattern and provides bounded asynchrony. The supervisory controller actively sends a command to the child device and waits for the result. If the child device finishes execution without receiving the *stop* command, child device will send full result and go back to the ready state. At the supervisory controller side, on receiving the full result, the supervisory controller will go to the *next* state and invokes the next command. On the other hand, the supervisory controller may need to interrupt the current execution due to environment changes or emergency alerts. Therefore, we provide bounded asynchrony, which allows the supervisory controller to send the *stop* command while in blocking state *B1*. After sending the *stop* command, the supervisory controller waits



(a) Supervisory Controller FSM (b) Child Device FSM (c) Message Interleaving

Figure 2.2: Finite state machines (FSM) and message interleaving of interruptible RPC pattern

$msg!$	Send msg (Insert msg into communication queue)
$msg?$	Receive msg (Extract msg from communication queue)
$A?/B!$	If receive A then send B
F_Result	Full result, generated after finishing the execution
P_Result	Partial result, generated after receiving $stop$

Table 2.1: FSM operation syntax

for the partial result in state *B2*. The reason for making supervisory controller enter a blocking state is to limit the asynchrony and prevent combinatorial possible message interleaving. Since the supervisory controller must wait until the child device to finish handling the *stop* message, the message interleaving is bounded. When the supervisory controller receives the *stop* command, it will stop the execution and send the partial result back. Due to the transmission delay, the *stop* command may arrive at child device side after it has already finished the execution (the message interleaving between *stop* and full result). In that case, the supervisory controller will accept the full result after sending the *stop* command.

The proposed model contains three verified properties.

- The only possible message interleaving is between *stop* command and full result, as shown in

Fig. 2.2(c). The supervisory controller will receive full result instead of partial result if the child device has already finished the execution.

- If the child device finishes the execution or the supervisory controller sends *stop* command, then the supervisory controller goes to the *next* state and the child device goes to ready state without leaving pending messages in the queue. Consequently, there is no deadlock in this building block.
- The size of the queue is bounded by two since the child device can only receive two consecutive commands: *call* and *stop*.

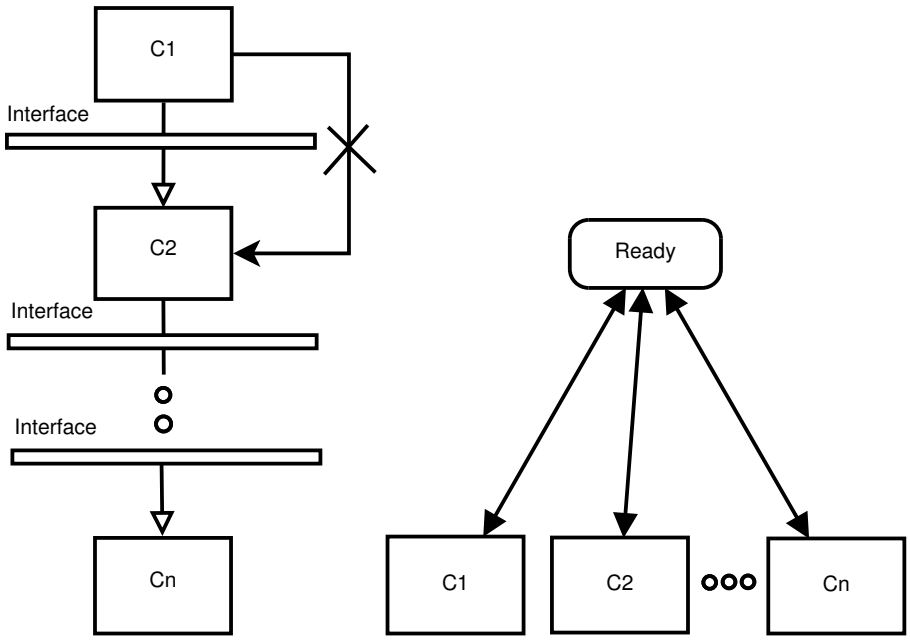
Interruptible RPC reduces the verification complexity but also introduces limitations to the system designs. The concurrency of the child devices is strictly limited. The supervisory controller cannot send a new command until all child devices send full or partial results. In addition, the supervisory controller can only issue an asynchronous message and must wait until it is handled by all the child devices to continue execution. However, we argue that the limitation of currency is a worthwhile comprise, or the combinatorial possible interleaving may cause state space explosion and result in a un-verifiable system design.

2.4 Composition of Interruptible RPC

In this section, we describe how our interruptible RPC pattern can be used in designing more complicated systems with multiple commands. Also we describe the complexity of composed system.

2.4.1 Composition Methodology

Fig. 2.3 shows the multi-command model on the caller side can be modeled as a sequential composition of state machines of commands that are built based on the interruptible RPC pattern. On the other hand, the composition on the callee side is in a parallel style and can also be easily composed with the interruptible RPC pattern. However, in reality the commands are executed in a sequential manner and there is no interleaving between the execution steps of different commands. The order of execution of commands on the callee side is enforced by the caller. Moreover, the only dependency between the commands of the proposed pattern is in the *interface* between the commands. This interface is

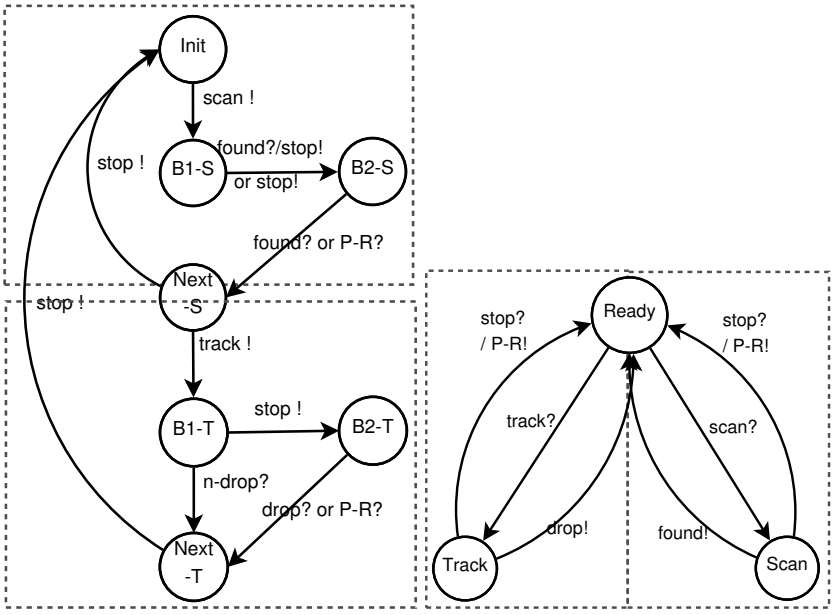


(a) Caller (b) Callee

Figure 2.3: Composition of multiple commands (C1,...,Cn).

the output of one command used as the input of the next command. The order of commands is the run-time order in which they are executed. Each command can start only when the previous command has completed its execution.

Fig. 2.4 shows a search and rescue system with two commands, *scan* and *track*. The system is composed using the interruptible RPC pattern and the composition methodology illustrated in Fig. 2.3. In the first phase, the mission manager sends out the *scan* command to both of the sensors. When a target is detected by any of the sensors, the mission manager moves to the tracking phase. In this phase the sensors will follow the target and depending on the application can report back the location, direction and speed of the target. In this model each sensor remains in the tracking mode until it receives a *stop* message from the mission manager or it loses the target. The sensor model is shown on the right side of Fig. 2.4. Similar to the original interruptible RPC model, on receiving a *stop* command, the sensors report back their partial result or full result if they have already finished the execution.



(a) Manager FSM (b) Sensor FSM

Figure 2.4: Interruptible RPC pattern based search and rescue system model

2.4.2 System Complexity

The verification cost of the system consists of two phases: verification of the commands, $C_{command}$, and verification of the composition, $C_{composition}$. These costs are *added* together instead of being *multiplied* due to the bounded asynchrony property provided by our proposed design pattern.

For the first part, $C_{command}$, when composing the system using the building blocks, the proposed composition methodology prevents *inter-command interleaving*. Therefore, there is no need to verify the different possible interleaving of execution steps of the commands in our system (Fig. 2.3). The verification cost is the *sum* of the verification cost of the commands.

In contrast, the asynchronous model may result in significant *inter-command interleaving*. Therefore, the complexity of verifying all possible interleaving is the *product* of verification cost of the commands that is much larger than the proposed pattern and can easily cause the state space of the system to explode.

The second part, $C_{composition}$, is the verification cost that is paid by the designer to find the correct wiring of the building blocks. This cost depends on the approach that the designer takes to obtain the

correct composition for the system. It should be mentioned that the focus of our proposed methodology is $C_{command}$. $C_{composition}$ is beyond the scope of this thesis, but we intend to address it in future work.

Based on the proposed design pattern, after verifying each command separately ($C_{command}$), the designers only need to verify the correct wiring of the building blocks ($C_{composition}$), that is independent of the individual commands. In this manner, the correctness of the composed system is guaranteed and there is no need to verify the whole system. The proposed design pattern not only provides the designers with a library that consists of building blocks and multiple options for each of the blocks but also greatly reduces the design and verification costs. The next section contains more details on the system design and verification cost.

2.5 Verification and Complexity Evaluation

(a) State space comparison (**number of states**)

	synchronous	interruptible	async(2)	async(3)	async(4)	async(5)
1 sensor	5	29	70	121	195	275
2 sensors	11	133	1410	4424	11816	23732
3 sensors	22	701	28691	184418	782417	*NF*

(b) Execution time comparison (**msec**)

	synchronous	interruptible	async(2)	async(3)	async(4)	async(5)
1 sensor	0	1	4	7	12	16
2 sensors	1	13	160	549	1571	3276
3 sensors	1	120	6586	49307	1096480	*NF*

Table 2.2: State space and execution time comparison of search and rescue system. async(n) denote asynchronous system of queue size n. *NF* means it didn't finish within 12 hours.

Table 2.2 shows state space and verification time of modeling a supervisory command and control system with different patterns: synchronous, asynchronous RPC, and the proposed interruptible RPC. For each pattern we varied the number of child devices from 1 to 3 in the model. For asynchronous RPC pattern, we varied the queue size from 2 to 5. Note that interruptible RPC uses queues of size two

since the pattern do not need more than two messages in the queue. We used Maude [30] to model the systems and compared the state space.

We see that the growth rate of the state space and verification time of the three patterns as the number of child devices increases. It is evident that asynchronous RPC patterns grow much faster than the other two. The growth rate of 'interruptible' pattern is about an order of magnitude slower, for each child device addition, than 'async(4)' – asynchronous with queue size of four. Also, the asynchronous RPC pattern suffers severely from state space explosion as the queue size grows. Therefore, model checking of even reasonably sized asynchronous RPC systems will be infeasible. On the other hand, the interruptible RPC pattern shows reasonable state space growth rate. Moreover, it is not affected by the queue size since the maximum interleaving is bounded by the design.

2.6 Summary and Future Work

In this chapter, we focus on the command and control systems for search and rescue and propose a design pattern, *Interruptible RPC*, as a building block of system designs. We show the proposed pattern has low complexity and verification costs in terms of number of states explored by the program verification tool. In addition, we propose a composition methodology to compose complex command and control systems without introducing state space explosion.

In future work, we will exploit the other safety-critical applications in order to assess the applicability of the proposed design pattern. Moreover, we shall further explore the hierarchical structure of command and control systems and provide the formal description and proof of the complexity.

Chapter 3

Reducing Verification Complexity: Coordination Architecture for Networked Supervisory Medical Systems

In the previous chapter, we identify that asynchronous communication significantly increases the verification complexity and propose a protocol, *interruptible RPC*, to reduce it. However, *interruptible RPC* does not handle exceptions, which are used to notify the high level controllers, e.g supervisory controller, of abnormalities in a hierarchical control system. In medical systems, one of the major types of exceptions is patient adverse events (PAEs) due to unexpected worsening of patient's conditions. PAEs introduce many challenges to design and verify medical systems. First, most PAEs are raised asynchronously by distributed medical devices at different times. In order to safely handle a PAE, physicians or supervisory controllers require the timely and consistent view of patient condition and device status. In addition, the devices must be coordinated to avoid potential adverse device interactions. Second, asynchronous PAEs could generate combinatorial message interleavings. In order to guarantee safety, all possible message interleavings must be thoroughly checked, which can increase the verification cost significantly [89]. In addition, in order to deal with more sophisticated medical scenarios and medical devices, additional layers are required, which make the verification even more challenging. In this chapter, we propose a reduced complexity architectural pattern [93], which consists

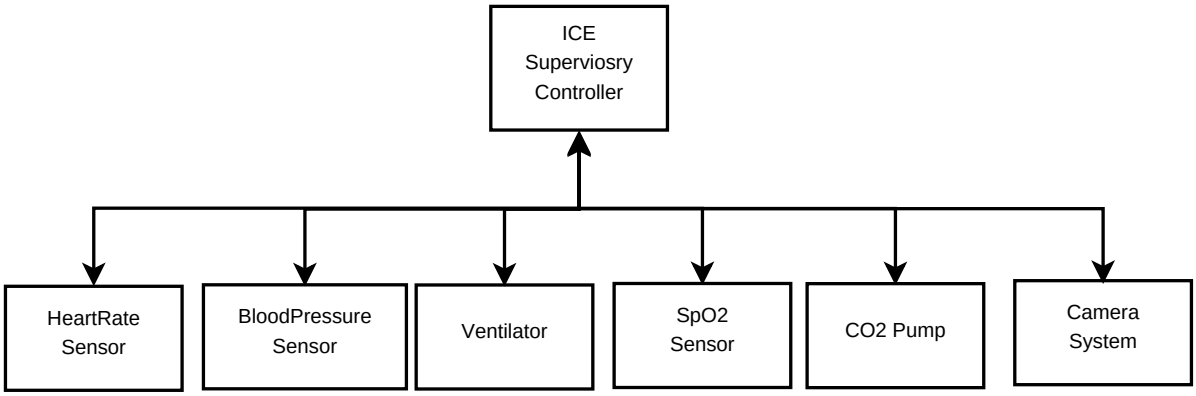


Figure 3.1: ICE structure for laparoscopic surgery

of *Hierarchical Organ-based Clustering Architecture* and *Consistent View Generation and Coordination (CVGC) Protocol* for coordinating medical networked supervisory systems and handling PAEs (Challenge 2).

3.1 System Model and Clinical Motivation

In this section, we present a simplified laparoscopic surgery scenario [46] under the Integrated Clinical Environment (ICE) [1] infrastructure as a motivating example to show potential safety hazards that might be caused by uncoordinated interaction between devices[2]. We then discuss the system models of the targeted networked supervisory medical systems.

3.1.1 Clinical Motivation

At the beginning of laparoscopic surgery, the patient is placed under anesthesia and requires a ventilator to provide oxygen. An SpO_2 sensor monitors patient's oxygen saturation (SpO_2) and raises a patient adverse event (PAE) if the oxygen saturation value is too low. The abdomen is inflated with carbon dioxide gas (CO_2) to elevate the abdominal wall to provide a working and viewing space for the surgeon within the abdominal cavity. Nevertheless, when the CO_2 pressure is increased, the diaphragm is

[1] Introduction of ICE infrastructure can be found in Appendix B

[2] Similar situation can be seen in many other medical scenarios, such as airway laser surgery [50] and orthopedic knee surgery [69].

pushed upward toward the lung. This upward movement of the diaphragm can make ventilation of the lungs and oxygenation difficult at times during these laparoscopic procedures. The laparoscopic video camera is inserted into the intra-abdominal space to transmit images from the abdominal cavity to high-resolution video monitors for surgeon operative view. As shown in Fig. 3.1, the ICE supervisor controls multiple medical devices directly. The medical devices raise a PAE to the ICE supervisor if an abnormality is detected, and the ICE supervisor sends commands to request the services from the devices to handle the PAE. For example, the ICE supervisor may receive an SpO_2_low PAE and can handle it by either improving the lung ventilation or decreasing the intra-abdominal pressure. But a conflict arises and a PAE may also be registered if the physicians detect that decreasing the CO_2 filled intra-abdominal space results in an inadequate surgical camera view. The ICE supervisor is faced with this conflict of a low SpO_2 PAE that requires the CO_2 pressure be lowered while the surgeons register a PAE that lowering the CO_2 restricts the view of the operative field within the abdomen. The SpO_2_low PAE and the inadequate space PAE are generated asynchronously, which may cause message interleaving and lead to safety hazards.

In supervisory medical systems, such as mentioned above, two types of communication patterns can be considered for the control of devices: *synchronous communication* and *asynchronous communication*. In synchronous communication, supervisory controls are performed in a call-and-block manner, and, hence, the interleaving of individual messages is limited. However, synchronous communication is inappropriate for timely processing of urgent events, such as PAEs. For example, PAEs, such as SpO_2_low, are generated asynchronously, and the ICE supervisor must wait until all previously requested services are completed by the devices. Otherwise, it might break the consistency between the devices, potentially incurring safety hazards. The blocking time may cause unacceptable delays for the handling of urgent PAEs. In the asynchronous communication, in contrast, the system can progress without blocking, which enables arbitrary message flows. Asynchronous communication is usually used to handle urgent and unexpected messages, such as PAEs [97]. However, asynchronous handling of PAEs introduces two significant design challenges.

PAE handling without timely and sufficient information may cause safety hazards. If the ICE supervisor handles a PAE without collecting the status of all the devices, the system may go to an unsafe state as shown in Fig 3.2. Suppose the patient's oxygen saturation is too low, and the SpO_2 sensor

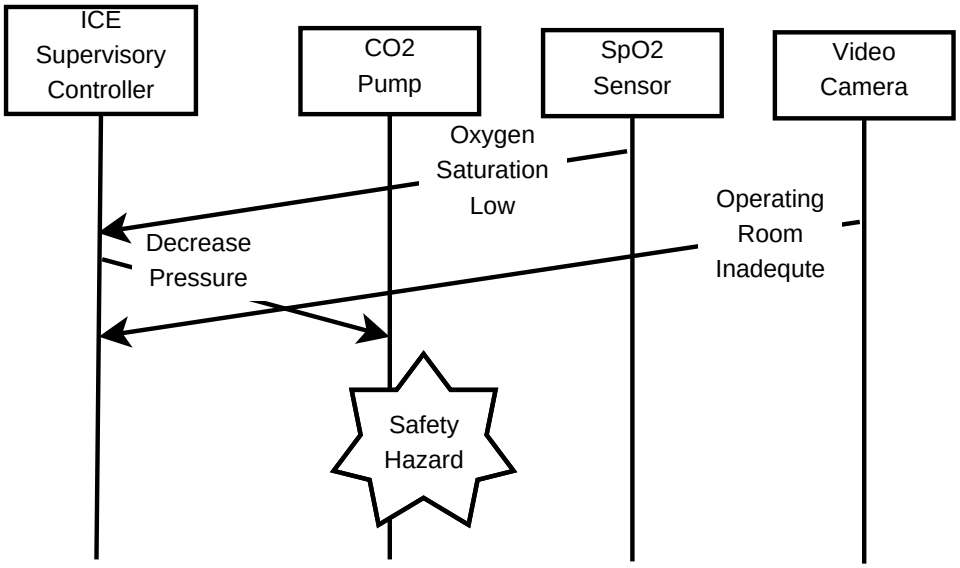


Figure 3.2: Safety hazards due to uncoordinated PAE handling

raises a PAE. The ICE supervisory controller requests the laparoscopic CO_2 pump to decrease intra-abdominal pressure without coordinating the camera system. At the same time, the surgeon detects the surgical space is inadequate and issues a PAE through the camera system. The previous command to decrease the pressure will cause the loss of the intra-abdominal field of vision. This situation is extremely dangerous if the surgeon is operating at a critical point of the surgical procedure.

Message interleaving of asynchronous PAEs may significantly increase the state space for verification. Asynchronous events may introduce a lot of message interleaving to systems. All the possible interleaving must be checked to make sure the PAEs are properly handled without causing any safety hazard. However, thoroughly checking all the interleaving of a reasonable size system is usually unfeasible even to modern model checking tools. Therefore, without properly controlling and reducing the complexity, the safety of the supervisory medical system cannot be guaranteed.

Note that similar situation can be seen in many other medical scenario. For instance, in the airway surgery model (laser surgery for a tracheal or airway surgical procedure), if a laser scalpel surgical device is used while the concentration of oxygen in the airway is high, a potential fire hazard may cause fatal damage to the patients [51]. The ICE supervisor must block the oxygen supply to the surgical field before allowing the laser scalpel surgical device to be activated. However, the exception

may be raised by the ventilator because the oxygen saturation is too low. Instead of unblocking the oxygen supply immediately, the ICE supervisor must turn off the laser scalpel surgical device first. However, for the rest of this chapter, we will use laparoscopic surgery as our primary case study.

3.1.2 System Model

In this section, we define system models that are used throughout this chapter.

Device controllers model: A device controller state consists of two parts: physiological measurements and configuration settings. The physiological measurement keeps changing to reflect patients' condition. However, a device reading is considered clinically valid if the reading is within a given time bound. To this end, a device reading O_i is associated with *physiological validity interval (t_{pvi})*, and it is considered as valid if $|timestamp(O_i) - current\_time| < t_{pvi}(O_i)$. The configuration settings of a device can be changed by device controller itself or by the ICE controller. For instance, the CO_2 pump's up-to-date state consists of valid CO_2 pressure level and the current target pressure level requested by the ICE controller.

ICE Controllers model: An ICE controller controls a set of device controllers according to configuration settings, which include boundary conditions for PAE generation, limits of control variables, and PAE handling routines.

Hereafter, we use the term device as the device controller and the term controllers to refer both device controllers and ICE controllers if not explicitly specified. The term *command* refers to the adjustment request to change the controllers configuration settings.

Communication and interaction model: The devices periodically send the measurements to the controllers and raise PAEs if abnormalities are detected. The controller sends command to the devices to change devices settings. The above message passings are through asynchronous communication channels. We model each communication channel with two FIFO queues. One is for the messages from the ICE controller to the device, and the other is for the messages from the device to the ICE controller. All messages are put in the corresponding queues and wait to be processed.

Finally, according to the above definitions of controller models and physiological validity, we define the *consistent view* as follows: :

Definition 1 *A **consistent view** of a controller C_α, denoted as CV_α, is the union of its own settings and the underlying controllers' settings and devices states, in which for n controllers $\forall_{i \leq n} |timestamp(O_i) - current\_time| < t_{pvi}(O_i)$ and the configuration settings are up-to-date.*

In this work we take advantage of the fact that human physiology changes are usually slower than the computer actions. Therefore, we assume that the consistent view can be generated within the valid time interval of the corresponding measurements. However, the real-time mechanisms to guarantee such timing requirements are for future work.

Since PAEs are the major reason to change devices settings, in the following sections we use PAEs handling scenarios to explain the proposed architecture and protocol. However, the proposed architecture can also guarantee consistency when the surgeon commands the configuration changes.

3.2 Hierarchical Organ-based Clustering Architecture

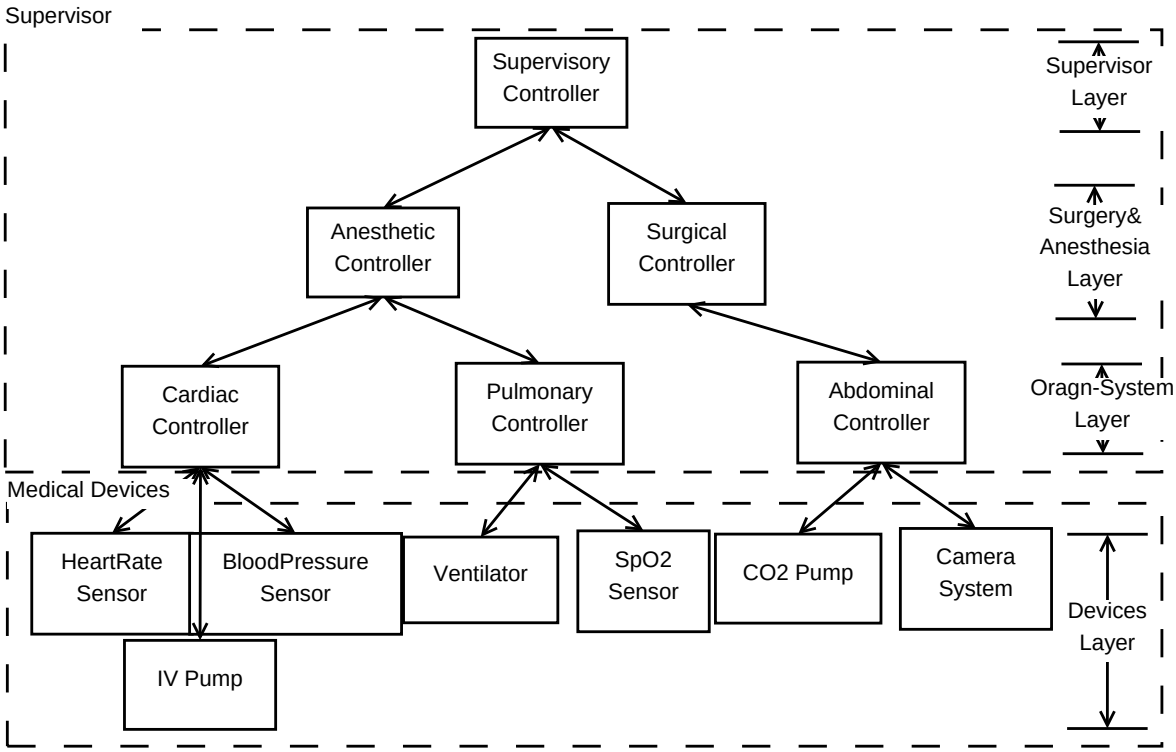


Figure 3.3: Hierarchical organ-based structure of the laparoscopic surgery

We propose an organ-based hierarchical control structure to enable a low-complexity coordination pattern among medical devices and controllers. The proposed architecture organizes the controllers in a tree structure, as shown in Fig. 3.3[3].

At **organ-system layer**, devices are grouped into organ-based clusters according to their physiological coupling. The rationale behind organ-based clustering is that human physiological variables within an organ system have strong correlation. For example, since the oxygen saturation level (SpO_2) and the configuration settings of the ventilator are closely related to the human breathing function, both the SpO_2 sensor and the ventilator belong to the same pulmonary cluster. Each organ-based cluster is the minimum unit to handle patient adverse events (PAEs). For instance, when the pulmonary controller receives an SpO_2_low PAE, it adjusts the ventilator settings, such as the tidal volume, fractional inspired oxygen (FiO_2), and breathing rate. The typical allowed adjustments are the tidal volume between 400 to 550 cc and FiO_2 between 40% to 60%[4]. Similarly, the cardiac controller is responsible for stabilizing the cardiac measurements, such as heart rate, blood pressure, and it adjusts the IV pump settings, which include medication and fluid volume. The abdominal controller is responsible for adjusting the CO_2 pump and the camera system to maintain adequate surgical operating space. If an organ-system controller receives a PAE, e.g., SpO_2_low, it first attempts to locally compensate if the device states are within the boundary conditions. However, if the device settings have reached the control limits or inter-organ-system coordination is required, the PAE is propagated to the surgery and anesthesia layer.

At **surgery and anesthesia layer**, clusters are grouped to provide inter-cluster coordination. In our current design, this layer consists of two controllers, an *anesthetic controller* and a *surgical controller*. They are responsible for handling the PAEs, which cannot be locally compensated by organ-system controllers. The surgical controller handles surgery related events, such as, increasing or decreasing the CO_2 pressure levels. Since many PAEs are handled by the anesthetic controller or its child controllers, the interference to the surgery can be minimized.

The **supervisory layer** provides the highest level of control, and it coordinates surgical and anes-

[3]The controller boxes shown in the figure represent the controller processes or threads, which can be physically placed in the same machine or distributed in different machines. In this work, the controllers are placed in an ICE supervisor computer.

[4]For more details, see `http://www.merckmanuals.com/professional/critical_care_medicine/respiratory_failure_and_mechanical_ventilation/overview_of_mechanical_ventilation.html`

thetic controllers. It also serves as a surgeon interface to receive commands from human operators or surgeons.

The proposed organ-based tree structure has the following benefits. First, the organ-based tree structure captures human homeostasis and enables local compensation. Therefore, PAEs can be locally handled by different organ-system controllers, which improves the system effectiveness. For example, within the control limits, the pulmonary controller and the cardiac controller can handle the PAEs concurrently. Moreover, the separation of concerns for different layers of controllers enables modular verification and reduces the complexity. Organ-system controllers focus on stabilizing individual organ-system, and the higher level controllers focus on inter-organ-system coordination. Second, since there is no direct communication between two controllers of the same layer, messages must go through higher level controllers. This tree communication structure avoids potential conflicts of commands and controls, since the coordination commands always go through the parent controller. In the medical environment, conflicting commands may cause safety hazards. For example, if multiple controllers command the same infusion pumps, they may give drugs that cause adverse interactions. In addition, organ-based tree structure provides fault-isolation and fault-tolerance. For instance, even if the surgery controller fails, the other controllers under the anesthesia controller can still function. For the details of the fault-tolerance issue in the organ-based hierarchical structure, readers are referred to our work [48].

3.3 Consistent View Generation and Coordination (CVGC) Protocol

Consistent views are essential for controllers to safely handle PAEs. However, since PAEs can be handled by the distributed controllers, the distributed PAE handling may cause race conditions and violates consistency. For example, the anesthetic controller receives CV_{pulm}, but in the mean time, the pulmonary controller issues a command to change the ventilator setting. CV_{pulm} received by the anesthetic controller is not consistent with the up-to-date device settings and cause ambiguity. In order to avoid the above situation, we introduce a locking mechanism to lock the settings of a subtree. *The locking means the controllers are not allowed to change their settings but still function according to the current settings.*

Definition 2 *A **consistent view** of a controller C_α, denoted as CV_α, is the union of its own settings and*

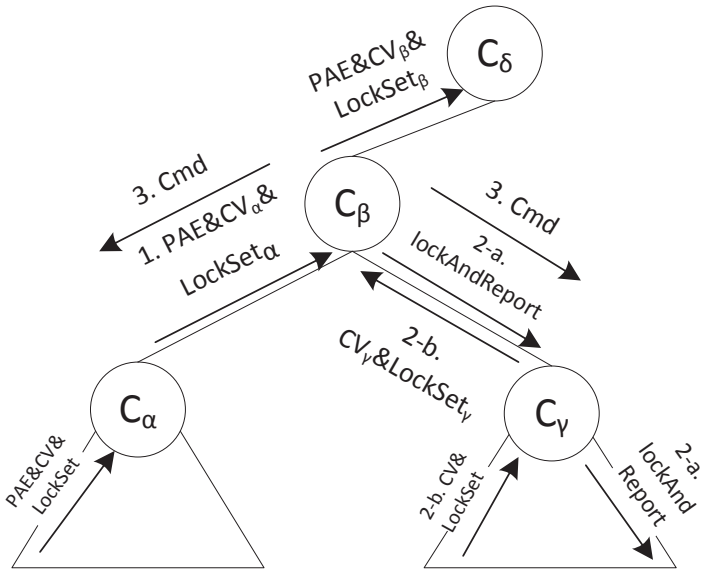


Figure 3.4: The *CVGC* protocol descriptions

the underlying controllers' settings and devices states, in which for n *controllers* $\forall_{i \leq n} |timestamp(O_i) - current\_time| < t_{pvi}(O_i)$ *and the configuration settings are up-to-date.*

Definition 3 *Each controller has a corresponding* ***lock*** *to protect its settings. A controller* C_α *can send command to change the settings of another controller* C_β *only if* C_α *holds* C_β*'s lock, denoted as* $Lock_\beta$

The *CVGC* protocol contains the following phases, as shown in Fig. 3.4.

1. **PAE propagation phase**: The controller C_α propagates the PAE to the controller C_β if C_α cannot handle the PAE. The settings of C_α's subtree must be locked, and CV_α and the set of locks that C_α currently holds, denoted as $LockSet_\alpha$, are also sent to C_β.

2. **Locking and consistent view generating phase**:

   2-a In order to establish a consistent view and lock the settings of the other subtree, the controller C_β sends the *lockAndReport* message to C_γ. The *lockAndReport* message is propagated downward until it reaches the device layer. The devices lock their settings and send their states and locks to their parent controller. The parent controllers of the devices combine the responses and propagate the consistent views and the locks upward.

2-b Following this layer-by-layer manner, C_β can obtain CV_γ and $LockSet_\gamma$. C_β then decides if it can handle the PAE. If yes, C_β goes to the *command execution and unlocking phase*. Otherwise, C_β further propagates the PAE and the locks to its parent controller, C_δ.

The subtree of C_γ may have been locked or in the middle of processing the *lockAndReport*, because C_γ receives a PAE$'$ from its child controller. There are two possible situations. In one situation, C_γ cannot handle the PAE$'$ and needs to propagate it to C_β or C_γ is waiting for the responses from its child controllers. C_γ waits until receiving the consistent views and the locks from its child controllers. C_γ then sends the PAE$'$, CV_γ and $LockSet_\gamma$ to C_β.

In another situation, C_γ has already sent the commands to locally compensate the PAE$'$. Since the subtree of C_γ is currently changing the settings, the *lockAndReport* message from C_β stays in the queue until the commands are executed and the subtree is unlocked. The command execution and tree unlocking will be detailed in the next phase. C_γ can then lock the subtree and generate the consistent view to C_β.

3. **Command execution and unlocking phase**: Controller C_β sends a command to handle the PAE and the command is propagated to all the C_β's descendant controllers. The controllers receiving the command change their settings and waits for the *unlock* message. In order to prevent the potential side effects of multiple simultaneous commands, physicians generally separate the two treatments with a time interval, unless they belong to different organ systems, which is described as human homeostasis. Therefore, C_β will wait for a pre-specified interval, and then sends the *unlock* message to its subtree and unlock itself. The set of locks that C_β holds, except $Lock_\beta$, are also released. The controllers receiving the command unlock itself, restores its own lock, and propagates the *unlock* message to its child controllers.

Moreover, in order to avoid unbounded waiting, each controller sets a timer when it sends the *lockAndReport* message. If any child controller does not respond within the specified timeout interval, it is regarded as failed and an alarm is generated to ask human intervention.

3.4 Correctness of the CVGC Protocol

In this section, we prove the correctness of the proposed protocol.

Theorem 1 *The CVGC protocol is deadlock free.*

Proof.

First, by assumption, a controller cannot deadlock with itself. Thus, a deadlock can only be formed by a set of controllers creating a circular wait. Assume there are n controllers $\{C_1, ..., C_n\}$ involving in the circular wait. Since the controllers are structured as a tree, the circular wait may happen only between the parent and child controllers. Suppose C_i is the parent controller and C_j is C_i's child controller and they form a circular wait. According to the pseudocode, C_i sends the *lockAndReport* message and wait for responses, while C_j sends CV_j and $LockSet_j$ and waits for the commands. However, in this case, the parent controller can successfully receive all the child controllers responses without forming a circular wait. Since we arrive at contradiction, the proposed protocol is deadlock free. □

Since PAEs are raised by distributed devices and handled by distributed controllers, the potential conflicts among commands may cause safety hazards. In order to guarantee the proper concurrency control, let us define the following terminologies and prove that the *CVGC* protocol guarantees proper concurrency control.

Definition 4 *An **action** is defined as a pair, action = (C, S), where C is a controller that receives the new settings, and S is the new settings*[5]*. A **command** is defined as a set of actions.*

In our protocol, a command sent from a controller, C_β, consists of the actions of all the C_β's descendants. For example, an action to the ventilator controller looks like $action_{vent} = (Ventilator, FiO_2 = 50\%)$, and a command sent from the pulmonary controller looks like $Cmd_{pulmonary}$ = $\{action_{vent}, action_{SpO_2}\}$.

Definition 5 ***Sequential Execution***

Cmd_1 and Cmd_2 are sequentially executed and Cmd_1 is executed prior to Cmd_2, denoted as $Cmd_1 \prec$

[5]If the settings of a controller are not changed, the S is set to null.

Cmd_2 iff $\forall_{i,j} action_1^i \hookleftarrow action_2^j$, where $\hookleftarrow$ represents the precedence relation of two actions. In other words, all the actions belonging to Cmd_1 must complete before the actions belonging to Cmd_2 can start.

Recall that our architecture captures human homeostasis and enables the local compensation within an organ system. Therefore, the commands from the pulmonary controllers are compatible with the commands from the abdominal controllers, because they control two disjoint subtrees. However, the commands from the pulmonary controllers are incompatible with the commands from the anesthetic controller, because they both contain the actions to the ventilator and SpO_2 sensor.

Definition 6 ***Compatibility and incompatibility of commands***

Two commands Cmd_1 and Cmd_2 are compatible if

$\forall_{i,j} action_1^i.C \neq action_2^j.C$. Otherwise, the two commands are incompatible.

Lemma 2 *Under the proposed architecture and protocol, two commands are incompatible iff the corresponding sending controllers are ancestor and descendant.*

Proof.

Suppose two controllers C_α and C_β send the commands Cmd_α and Cmd_β to their descendants. Proof by contradiction. To prove the "if" direction, we assume two commands are compatible. Without loss of generality, we assume C_α is C_β's descendant. Since the two commands are compatible, there are no actions of the two commands controlling the same controllers. However, since C_α is C_β's descendant, according to the protocol, Cmd_β must contain the actions of the subtree of C_α. We arrive at contradiction.

To prove the "only if" direction, we assume C_α and C_β are not ancestor and descendant and the two commands are incompatible. Since C_α and C_β are not ancestor and descendant, the two commands contain the actions of two exclusive subtrees. We arrive at contradiction, since the two commands are compatible. □

In medical environment, PAEs are generally handled sequentially unless they can be locally compensated by different organ systems. Therefore, the following theorem proves the correctness of the concurrency control.

Theorem 3 *Under the CVGC protocol, compatible commands are concurrently executed, and incompatible commands are sequentially executed.*

Proof.

Suppose two controllers C_α and C_β try to send the commands Cmd_α and Cmd_β to their descendants. The concurrency of the compatible commands is clearly true. Since Cmd_α and Cmd_β are compatible, C_α and C_β can both successfully receive the consistent view of their subtrees and send the commands.

We prove the sequential execution by contradiction. Suppose that Cmd_α and Cmd_β are incompatible and they are not sequentially executed. According to Lemma 2, C_α and C_β must be ancestor and descendant. Without loss of generality, we assume C_α is C_β's descendant. Since Cmd_α and Cmd_β are not sequentially executed, $\exists_{i,j,m,n} action_\beta^j \hookleftarrow action_\alpha^i$ and $action_\alpha^m \hookleftarrow action_\beta^n$. In addition, it implies that both C_α and C_β receive the consistent view according to the protocol and send commands. Recall that only the controller that holds all its descendant controllers' locks can issue a command. Since C_α and C_β are ancestor and descendant, it is impossible that they both holds all the locks and issue the commands. We arrive at contradiction, so incompatible commands are sequentially executed. □

The above theorem proves that the *CVGC* protocol strictly regulates the interactions between commands. In addition, since the interleaving of the incompatible commands is eliminated, the verification complexity is significantly reduced. Moreover, sequential execution and independence of concurrent execution enables modular verification [8].

In addition, the queue size (which is defined as the number of messages that reside in the queue) of four is sufficient for the *CVGC* protocol[6]. The *lockAndReport* message enforces the nodes to process all the messages and lock the subtree. In addition, after a controller sends a command, it must wait for a pre-defined time interval and then sends the *unlock* message. We assume the time interval is sufficient for the controllers to process the command. Therefore, only the *lockAndReport* and the *unlock* messages can stay in the queue that is for the messages from the parent to the child controllers. For another, as stated in the previous section, the devices periodically send the measurements. We assume

[6]We use a model checking tool to verify this property.

that the controllers can process the measurements faster than the devices can generate. Therefore only the controllers' states, consistent view, locks, and PAEs can stay in the queue that is for the messages from the child to the parent controllers.

As the readers may have already noticed, generation of consistent view requires extra messages exchanges, which may introduce delays to PAE handling. The consistent view generation time (*CVGC-Time*) can be calculated by the following formula

$$\begin{aligned} Controller.CVGCTime = 2\mu + Controller.StatesWCET \\ + TreatmentInterval + \max_{i \in Children} Child_i.CVGCTime \end{aligned} \tag{3.1}$$

Here μ is the maximum one-way message transmission delay and *StatesWCET* is the worst case execution time for generating the responses after receiving the *lockAndReport* message, and TreatmentInterval is the waiting time for the subtree to be unlocked if it is currently execution a command. The above formula recursively computes the *CVGCTime* of each node according to the *CVGCTime* of the child nodes. We believe that the delay introduced is a worthwhile compromise. Directly changing the device settings without knowing the up-to-date system state can potentially cause safety hazards. Furthermore, in a supervisory medical system, the number of hierarchical layers and medical devices are limited, so the introduced delay is fairly tolerable.

3.5 Verification and Complexity Evaluation

In order to evaluate the verification complexity of the proposed architecture and verify the correctness and safety properties, we use abdominal laparoscopic surgery as the case study and model the systems in UPPAAL [13] to show the number of states and verification time. We compare the proposed design with totally asynchronous communication model, in which a node can arbitrarily send messages until the receiver's queue is full. We use queue size as a parameter to evaluate how the message interleaving affects the verification complexity. We also show the number of hierarchical layers as a parameter to evaluate the scalability of the design.

Abdominal laparoscopic surgery case study [46]: At the beginning of laparoscopic surgery, the patient is placed under anesthesia and requires a ventilator to provide oxygen. An SpO_2 sensor monitors patient's oxygen saturation and raises a PAE if SpO_2 value is too low. The abdomen is inflated with

carbon dioxide gas (CO_2) to elevate the abdominal wall to provide a working and viewing space for the surgeon within the abdominal cavity. Nevertheless, when the CO_2 pressure is increased, the diaphragm is pushed upward toward the lung. This upward movement of the diaphragm can make ventilation of the lungs and oxygenation difficult at times during these laparoscopic procedures. The laparoscopic video camera is inserted into the intra-abdominal space to transmit images from the abdominal cavity to high-resolution video monitors for surgeon operative view.

	Verified Properties
Two Layers	P1: The pulmonary controller will increase the control variables of the ventilator upon receiving a SpO_2_low PAE if they are within the normal ranges.
	P2: The abdominal controller will increase the laparoscopic CO_2 pump upon receiving an inadequate space PAE.
Three Layers	P3: If the cardiac controller issues a PAE indicating the heart rate and the blood pressure are above the threshold and the pulmonary controller issues a SpO_2_low PAE, the anesthetic controller decreases the Intravenous (IV) fluid volume.
Four Layers	P4: If the surgery is not at the critical point and the ventilator raises a SpO_2_low PAE, which is propagated to the supervisory controller, the supervisory controller must command the laparoscopic CO_2 pump to reduce intra-abdominal pressure.
	P5: If the surgery is at a critical point, the supervisory controller should not allow a reduction of the laparoscopic CO_2 pump pressure. Instead, an alarm must be raised to the surgeon or human operator because the control logic has no predefined rules to handle this situation.
CVGC Properties	P6: The system is deadlock free.
	P7: A controller can issue a command if and only if it holds all its descendant controllers' locks.
	P8: There are no incompatible commands executed concurrently
	P9: The queue size of the CVGC protocol is bounded by four.

Table 3.1: Verified properties of the laparoscopic surgery

The verified properties are shown in Table 3.1. The two-layer system consists of three organ-system controllers and medical devices. The three-layer system includes the surgical and anesthetic controllers, which are responsible for coordinating multiple organ systems. For the four layer system,

Hierarchical Layers	Metrics	CVGC	Asynchronous				
			queue 2	queue 4	queue 6	queue 8	queue 10
Two Layers	Number of states	1887	12084	239410	3346875	40639431	OM
	Checking time (s)	<1	3	57	797	8677	OM
Three Layers	Number of states	10651	603056	25040070	OM	OM	OM
	Checking time (s)	2	143	5173	OM	OM	OM
Four Layers	Number of states	21888	OM	OM	OM	OM	OM
	Checking Time (s)	8	OM	OM	OM	OM	OM

Table 3.2: Complexity comparisons with different hierarchical layers (OM indicates UPPAAL runs out of memory.)

a supervisory controller is introduced to coordinate surgical and anesthetic controller[7]. The verified properties of the layers are inclusive. In other words, to verify the four-layer system, the properties of three-layer and two-layer systems must be verified as well. Note that the properties are used to demonstrate the efficiency of complexity reduction. Other safety or liveness properties can also be checked.

Table 3.2 shows the evaluation results in terms of number of states and verification time. The asynchronous model provides flexibility but also introduces a lot of message interleaving. For example, for the two-layer system, the synchronous model has only 546 states while the asynchronous model with queue size 2 involves more than 12,000 states. Furthermore, the complexity of asynchronous model grows exponentially as the queue size increases. For instance, while the queue size increases from 4 to 8, the state space increases by several orders of magnitude. In addition, the hierarchical structure results in more complicated message interleaving and introduces extremely large search space in the asynchronous model. In the four-layer system, UPPAAL runs out of memory for all asynchronous designs. This situation gets worse when more devices and more hierarchical layers are involved in the systems. Consequently, the model checking tools cannot thoroughly verify the asynchronous model of a reasonable size. Without thorough verifications, the safety of the system cannot be completely

[7]A supervisory medical system must include all four layers to guarantee safety. However, we conceptually separately them to demonstrate the impact of the hierarchical layers to the complexity.

guaranteed.

The *CVGC* model deploys a locking mechanism for coordination and reduces the unnecessary message interleaving. Moreover, the complexity of the proposed model is not affected by the queue size, because the message interleaving is reduced. Even for the four-layer system, the number of states is still less than 22,000, and the verification time is less than 10 seconds. Therefore, we believe that our proposed architecture design can achieve good scalability.

3.6 Pattern Deployment and Tool Support

In this section, we describe how to specify the proposed architecture design to an AADL pattern. The system developers can follow the same principle to instantiate the proposed architecture in different systems. In addition, we developed a prototype tool in OSATE (the Eclipse-based AADL development environment [33]) to validate the specifications of the system designs. The tool reads the system model instances and checks if the AADL specifications follow the behavior of the proposed pattern described in the previous sections. Once the AADL specifications are validated, the system developers can safely use the pattern with specified properties. Therefore, the design flow can be summarized as follows. The system developers first design the system architecture in AADL and deploy the proposed pattern as a communication layer. Second, our developed tool can perform sanity checks on the AADL architecture design and raise warnings if any architectural property is violated. Once the design passes the sanity check, the proposed protocol is correctly deployed with specified semantics. In order to thoroughly verify the application specific properties, the system developers need to translate the AADL design to the models that are used by the model checking tools. In this stage, we do not have the tools for automatic translation[8]. However, the developed tool can help system designers perform an early check to make sure the proposed pattern is correctly deployed. In the following sections, we first show a sample AADL specification and then describe what are the sanity checks performed.

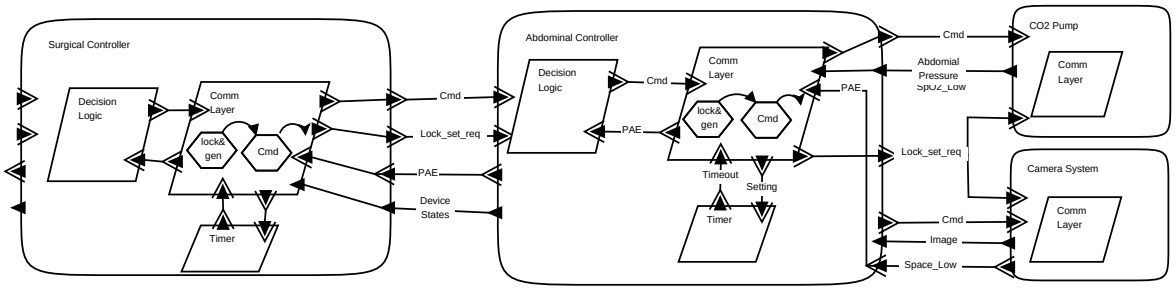


Figure 3.5: AADL structure

3.6.1 Pattern Specifications in AADL

In order to instantiate the proposed architectural pattern, first of all, the system components should have communication layers implementing the CVGC protocol. In addition, event data ports for the *lock&report* message and each PAE are required for each system component. The connections must also be correctly declared to specify the message passing between the components. Secondly, the communication layers should have two execution modes: one corresponds to the lock and consistent view generation phase (*lock_and_gen*), and the other corresponds to the command sending phase (*cmd*). A node enters a *lock_and_gen* mode upon receiving a PAE or a *lock&report* message through the pre-defined event data ports. The consistent view is sent in a bottom-up manner when the nodes are in the locked mode. A node transits from the *lock_and_gen* mode to the *cmd* mode when sending or receiving a command. Thirdly, a node specifies the worst case execution time for generating the *status* message upon receiving *lock&report* message in *CVGC::StatusWCET* property. The maximum tolerable time of waiting for consistent view generation is specified in *CVGC::TolerableTime* property. We will show how to check if the above specifications follow the proposed pattern in Section 3.6.2.

Fig. 3.5 shows the part of the AADL structure of a simplified system for the laparoscopic surgery (some ports and connections are removed for readability). The messages go through the communication layer in each component, which has the corresponding ports for commands, PAEs and device data. For example, there is an event data port corresponding to the Space_low PAE from the camera system to

[8]We are currently developing a tool for automatic translation, which can close the gap between architecture design and formal verification.

the communication layer of the abdominal controller. We show part of the AADL specifications of the abdominal controller for demonstration.

Algorithm 1 The AADL specifications of the abdominal controller

```
System implementation AbdominalCtrl.impl
subcomponents
  decision_logic: system DecisionLogic.impl;
  comm_layer: system CommLayer.impl;
end AbdominalCtrl.impl;

System implementation CommLayer.impl
Properties
  CVGC::PAEPorts=>("Space_low");
  CVGC::StatusWCET=>10ms;
  CVGC::TolerableTime=>100ms;
  ...
Modes
  normal: initial mode;
  cmd: mode;
  lock_and_gen: mode
end CommLayer.impl;
```

3.6.2 CVGC Protocol Property Check

We perform a set of sanity checks to guarantee the CVGC protocol is correctly implemented in each node. The checks of critical components guarantee that the communication components having two execution modes. The checks of ports connections guarantee that the message passings between components are correctly specified. The checks of consistent view generation time (*CVGCTime*), which is calculated using Equation 3.1, guarantees that each node's *CVGCTime* is less than or equal to its

maximum tolerable time. If any property is violated in the system design, the tool generates warning messages to indicate which part of the system does not satisfy the properties. Following are the checked properties for each node:

- Critical Components
    1. Each communication component must have two modes: a *lock_and_gen* mode and a *cmd* mode.
    2. A node transits to a *lock_and_gen* mode upon receiving a PAE or a *lock&report* message.
    3. A node transits an *cmd* mode upon sending or receiving a command.
- Port Connections
    1. There is no port connection between the nodes of the same hierarchical layer (the system components must be in a tree structure).
    2. The messages must go through communication layers.
    3. The ports of a node are correctly connected to the corresponding ports of the upper layer nodes.
- Consistent View Generation Time
    1. $Controller.CVGCTime \leq Controller.TolerableTime$

In summary, in order to close the gap between architectural patterns and system implementations, we proposed a methodology to model the proposed architectural pattern in AADL. In addition, we developed a sanity-check tool to validate if the system implementations follow the architectural pattern.

3.7 Summary and Future Work

In this chapter, we propose an architecture pattern to provide consistent coordination in the context of ICE-based supervisory medical systems. The hierarchical organ-system architecture allows local compensation for PAEs within different layers of controllers. The proposed CVGC protocol establishes the consistent view and coordinates the commands in a layer-by-layer manner. Therefore, the safety

hazards can be mitigated and the message interleaving is bounded. We use the laparoscopic surgery as a case study and show that the verification space and time is orders of magnitude smaller than those of the corresponding asynchronous communication models. The effectiveness of the proposed design is more evident as the number of hierarchical layers increases.

In our future work, we plan to introduce fault-tolerance mechanisms to the proposed pattern, such as self-stabilizing mechanisms [31] and NASS framework [50]. In addition, we further plan to develop an architectural tool in SAE Architecture Analysis $ Design Languages (AADL) [34] to instantiate the proposed architecture pattern.

Chapter 4

Reducing Cyber Medical Treatment Complexity: A Treatment Validation Protocol

In this chapter, we will address cyber medical treatment complexity. Incorrectly performing treatments may increase the chance of medical complications and compromise patient's safety. In order to assist medical staff to correctly perform treatments, we propose a treatment validation protocol [95] address the following three essential aspects (Challenge 3).

Precondition: A treatment can be performed only if the preconditions are satisfied. Unlike traditional cyber system preconditions, the medical system cannot lock or rollback the states of physical components, such as patient conditions. The system should request corrective treatments from the medical staff if certain preconditions are not satisfied. Nevertheless, the corrective treatments may have preconditions as well, which result in cascading of preconditions and treatments. Therefore, the system must organize preconditions and treatment to help medical staff keep track of preconditions and performed treatments.

Potential side effect: The side effects of a treatment may adversely affect other treatments or invalidate previously satisfied preconditions. The system should continuously monitor the potential side effects and alert medical staff to adjust the treatments. Moreover, the system must dynamically change the

structure of preconditions and treatments to reflect the adjustments from medical staff.

Expected response: The patient response must be checked after a treatment is performed. If patient response is not as expected, the system must alert the medical staff to issue an alternative treatment. Therefore, the medical errors due to mistakenly performed treatments can be reduced.

4.1 Motivation

In this section, we use cardiac arrest resuscitation to illustrate the concepts of treatment validation. **Example 1:** In a cardiac arrest resuscitation, medical staff intend to activate a defibrillator to deliver a therapeutic level of electrical shock that can correct certain types of deadly irregular heart-beats such as ventricular fibrillation. The medical staff need to check two preconditions: 1)patient's airway and breathing are under control and 2)the EKG monitor shows a shockable rhythm[1]. Suppose the patient's airway is open and breathing is under control. However, the EKG monitor shows a non-shockable rhythm[2]. In order to induce a shockable rhythm, a drug, called epinephrine, is commonly given to increase cardiac output. Giving epinephrine, nevertheless, also has two preconditions: patient's blood pH value should be larger than 7.4.[3], and urine flow rate should be greater than 12 mL/s[4]. In order to correct these two preconditions, sodium bicarbonate should be given to raise blood pH value, and intravenous fluid should be increased to improve urine flow rate.

The cascading relations between preconditions and corrective treatments can be captured by a tree structure, as shown in Figure 4.1. It seems that the satisfaction of preconditions can be achieved by the well-known post-order tree traversal. However, in medical environment, a treatment may not be effective, and the side effects of a treatment may invalidate the previously satisfied preconditions of any tree nodes at any time.

Example 2: One potential side effect of sodium bicarbonate is suppressed respiratory drive[5], which

[1]The shockable rhythms are ventricular fibrillation and ventricular tachycardia [35].

[2]Non-shockable rhythms are asystole and pulseless electrical activity

[3]Severe acidosis, which is an increased acidity in the blood and other body tissue, will significantly reduce the effectiveness of epinephrine [35]

[4]If a patient suffers from kidney insufficiency, giving epinephrine may worsen the kidney function and cause acute renal failure [35].

[5]Respiratory drive is the control of respiration, which involves the exchange of oxygen and carbon dioxide

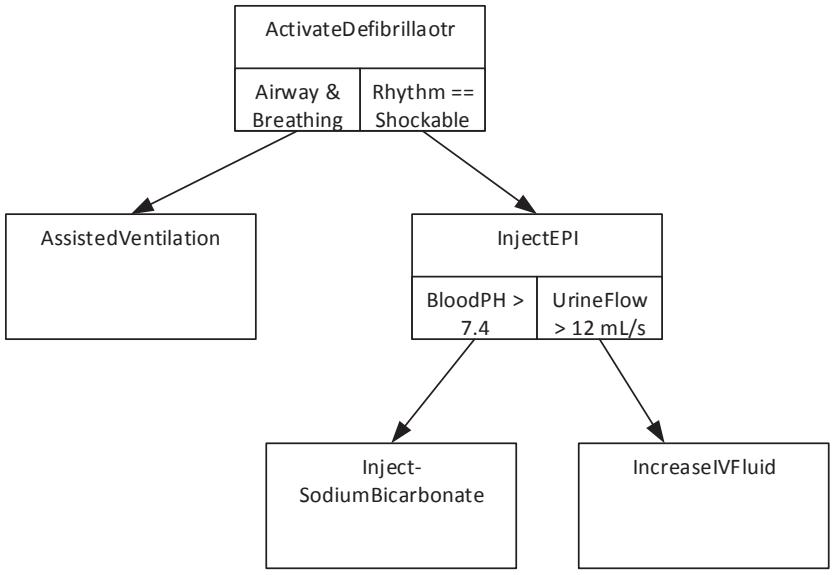


Figure 4.1: Treatments and preconditions tree

adversely affect patient breathing. Since the precondition is invalidated, the tree should be expanded to include the corrective treatment, such as provide assisted ventilation. In addition, increasing IV fluid volume may not successfully improve patient's urine flow rate. In this case, diuretics, such as Lasix, should be given, which leads to a different tree structure.

As illustrated in Example 2, the dynamics of patient conditions and the non-deterministic behavior of treatments pose significant challenges. The post order tree traversal alone may not be able to address these challenges. Similar situation can be found in many other medical scenarios, for instance, laparoscopic abdominal surgery [80] and airway laser surgery [48].

4.2 Treatment Validation Protocol Design

First, we propose a *Treatment Precondition and Correction (TPC)* tree for structuring the preconditions and treatments. Based on the TPC tree, the medical staff can keep track of the preconditions and treatments with concise and comprehensive physiological information. A tree node represents a treatment, and the number of children equals the number of the preconditions of the treatment. An edge represents the relation of a precondition and the corresponding corrective treatment. The tree is built in a top-down manner, and the root node is the treatment that the medical staff intend to perform in the

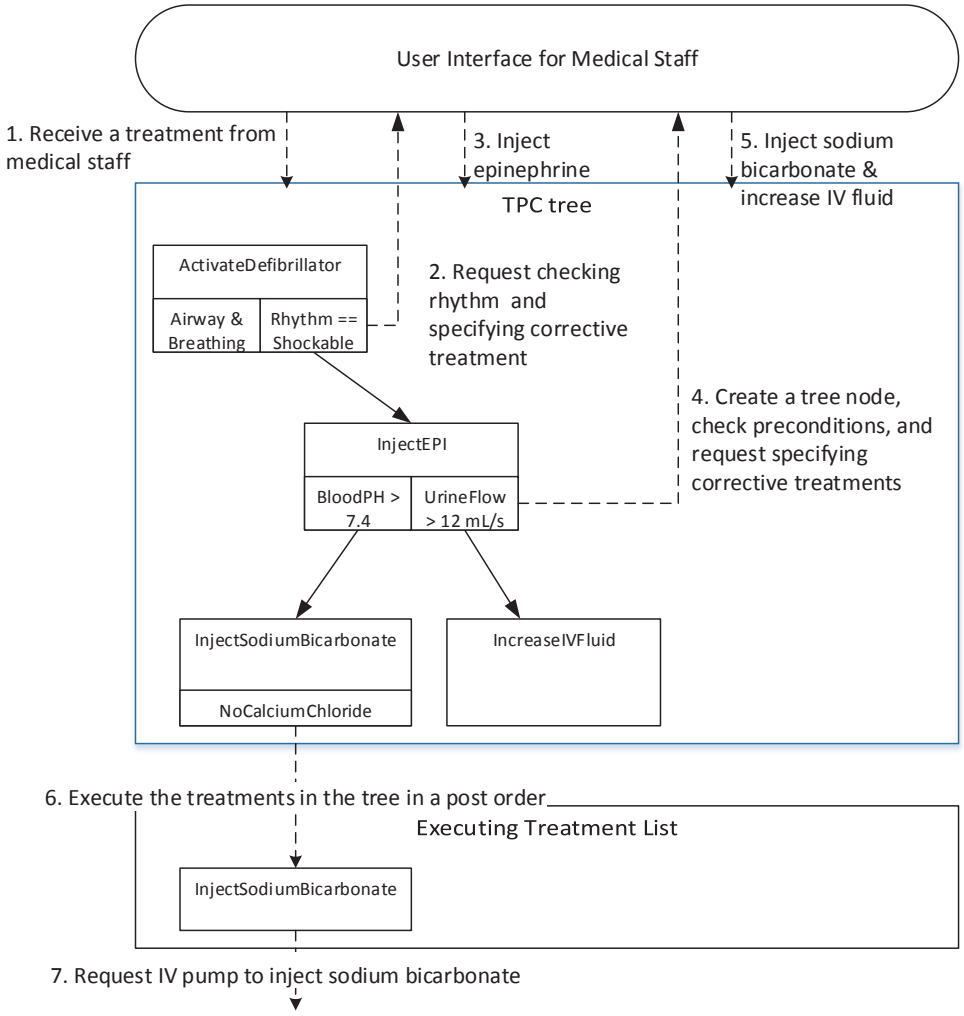


Figure 4.2: TPC tree construction and execution

beginning. If any precondition of the treatment is not satisfied, a child node for the corrective treatment is added. Since the corrective treatment may have its own preconditions and further corrective treatments, the height of the tree increases accordingly. The leaf nodes are the treatments that either have no preconditions or the preconditions are satisfied. In addition, due to the dynamics of patient conditions and potential side effects, the TPC tree is not static and requires to be dynamically updated. Like fault-tree [60], the proposed TPC tree aims to capture the cause and effect relations and analyze whether the root node can be reached or not. Unlike fault-tree, TPC tree intends to reach the root node by adding corrective treatments and must adapt to the dynamics of the medical environment. Moreover, TPC tree also serves as an interface between the medical staff and the system. Since the TPC tree main-

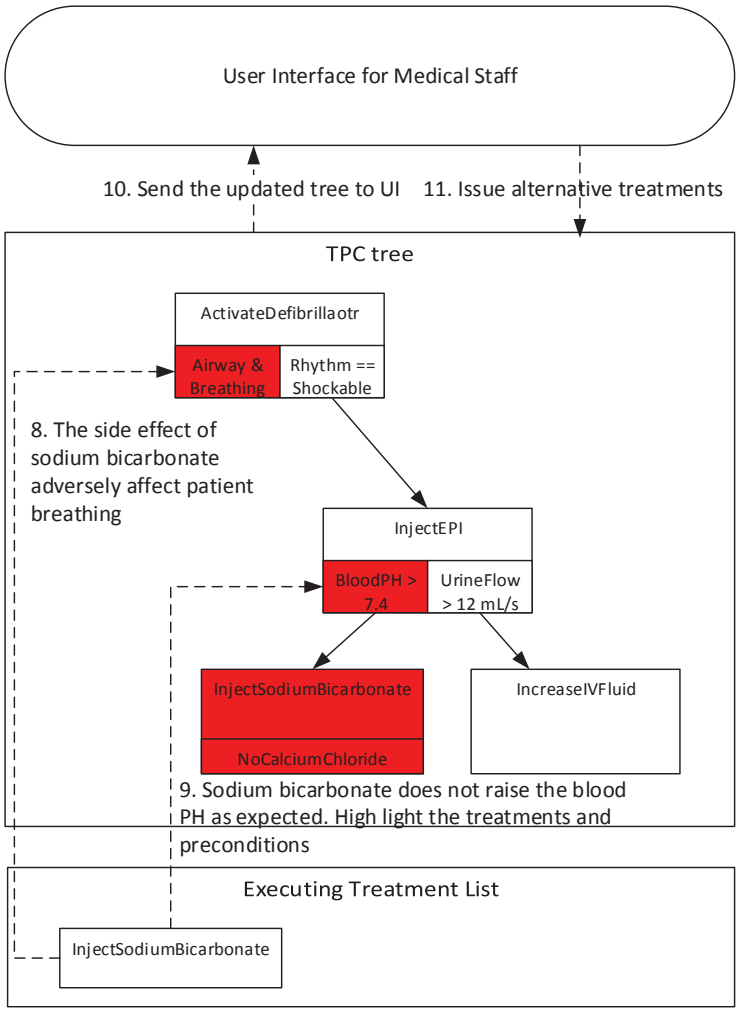


Figure 4.3: Side effect monitoring and expected responses checking

tains a logical relations between preconditions and corresponding corrective treatments, medical staff can keep track of the progress of the medical procedures. In addition, the system provides feedback through the tree to the medical staff if the side effects of a treatment start to interfere other treatments or invalidate the previously satisfied preconditions. On the other hand, the medical staff specify the corrective treatments for the unsatisfied preconditions based on the tree structure.

Second, let us use the following scenario to illustrate how validation of preconditions and corrective treatments can be achieved with the proposed TPC tree. Suppose the medical staff send a request to activate a defibrillator. The system builds the TPC tree rooted at *ActivateDefibrillator*. Since activating defibrillator has two precondition: airway and breathing are under control, and EKG shows a shockable

rhythm. Assume that the first precondition is satisfied, but our system cannot automatically analyze the EKG rhythm and requires medical staff's diagnosis. The system requests the medical staff to check EKG rhythm and specifies a corrective treatment if EKG shows a non-shockable rhythm. Suppose the medical staff determine that EKG rhythm is non-shockable and specify giving epinephrine, as illustrated in step 1–3 of Fig. 4.2. Giving epinephrine has two preconditions: patient's blood pH larger than 7.4 and urine flow rate larger than 12 mL/s. In this scenario, these two preconditions can be checked by the system and neither of them are satisfied. The system, then, requests the medical staff to specify the corresponding corrective treatments. One corrective treatment is injecting sodium bicarbonate for correcting blood pH value, and the other is increasing IV fluid volume for improving urine flow rate, as illustrated in step 4 and 5 of Fig. 4.2. Since injecting sodium bicarbonate requires that calcium chloride is not currently being injected[6], the system must check the IV pump status. Suppose IV pump is not currently injecting calcium chloride and there is no precondition for increasing IV fluid volume. The construction of TPC tree is complete, because there is no further corrective treatment to be added.

The system sends the TPC tree to the medical staff for approval. If the medical staff disapproves the treatments, the system discards the TPC tree and waits for the medical staff to issue a new treatment. Otherwise, the system executes the treatments in a post order, since the treatment of the leaf node must be performed first to correct the precondition. In this case, the system requests the IV pump to inject sodium bicarbonate, as illustrated in step 6 and 7 of Fig. 4.2.

The system monitors the potential side effects and checks the expected responses of sodium bicarbonate. The following cases might occur:

Case 1: Injecting sodium bicarbonate does not invalidate any precondition and successfully raises patient's blood pH value higher than 7.4. The system removes *InjectSodiumBicarbonate* node from the TPC tree and executes *IncreaseIVFluid*.

Case 2: Sodium bicarbonate adversely affects the patient breathing. The system "highlights" the preconditions in the TPC tree and requests the medical staff to specify a corrective treatment, as illustrated in step 8 of Fig. 4.2. Suppose the medical staff specifies providing assisted ventilation. The system adds a treatment node to the TPC tree*AssistedVentilation*.

[6]Sodium bicarbonate and calcium chloride cannot be simultaneously injected through the same IV pump, because they would form calcium carbonate and make the two drugs ineffective.

Case 3: Sodium bicarbonate fails to raise patient's blood pH value. The system "highlights" the treatment node and the corresponding preconditions, as illustrated in step 9 of Fig. 4.2. The system sends the TPC tree to the medical staff and requests the medical staff to specify an alternative corrective treatment. Note that once the medical staff specify a new corrective treatment, the system must update the TPC tree and check the preconditions of the new treatment, as described before.

4.2.1 Protocol Algorithm

In this section, we describe the validation protocol in detail and show part of the pseudocode in Algorithm 2 and Algorithm 3. Algorithm 2 shows the construction of the TPC tree. Algorithm 3 shows the post-order execution and dynamic side effect monitoring. Our protocol consists of the following phases.

1. TPC tree construction phase: The system receives a treatment from the medical staff and starts to build a TPC tree in a breath-first manner. The system checks the preconditions of the received treatment. If any precondition is not satisfied or must be checked by the medical staff, the system sends the tree to the medical staff and requests them to check the preconditions and specify the corrective treatments, as shown in the line 7–14 of Algorithm 2. After getting the input from the medical staff, the system checks if each unsatisfied precondition has a corresponding corrective treatment. If the corrective treatments are incomplete, an exception is sent to the medical staff, as shown in the line 19–23 of Algorithm 2. The system then adds the corrective treatments as child nodes to the TPC tree. Since the corrective treatments may introduce a new set of preconditions, the system checks the preconditions and expands the tree. If there are no further preconditions to check or all the preconditions are satisfied, TPC tree is sent to the medical staff for approval. If the medical staff approves the TPC tree, the system enters the execution and monitoring phase.

2. Execution and monitoring phase: The system executes the treatments in the TPC tree in a post order. In order to keep track of all the ongoing treatments, the system maintains an executing treatment list. Since patient conditions dynamically change, the system checks the preconditions of the treatment again before performing it[7]. If the preconditions are satisfied, the system inserts the treatment into the

[7]If the precondition that the treatment intends to correct is satisfied before the treatment is performed, the treatment is

Algorithm 2 Pseudo code for constructing TPC tree

```
1  checkPrecondAndBuildTree(TPCTreeNode root){
2    PrecondSet precondSet←root.treatment.PS;
3    TreatmentSet correctiveTreatmentSet;
4    Precondition precond;
5
6    while(precondSet ≠ ∅){
7      for each precond ∈ precondSet{
8        if(precond.Checker is medical devices){
9          if(precond is satisfied)
10           precond.status←SATISFIED;
11         else
12           precond.status←UNSATISFIED;
13       } else if(precond.Checker is medical staff)
14         precond.status←UNKNOWN;
15     }
16     ...
17     correctiveTreatmentSet←receiveFromUI();
18     // check the completeness of corrective treatments
19     for each precond ∈ precondSet{
20       if(precond.status == UNKNONW || (precond.status == UNSATISFIED && no
             corrective treatments))
21         sendExceptionToUI(INCOMPLETE);
22         ...
23     }
24     for each treatment ∈ correctiveTreatmentSet{
25       // update the precondition set
26       precondSet←∪treatment.PS
27     }
28
29   }
30 }
```

removed from the tree.

Algorithm 3 Pseudo code for post order execution and side effect monitoring

```
1  postOrderExecution(TPCTreeNode node) {
2    ...
3    TPCTreeNode childNode;
4    for each childNode ∈ node.childNodeList{
5      postOrderExecution(childNode);
6    }
7
8    if(all preconditions are satisfied){
9      performingTreatment(node);
10     setUpTimerForExpectedResponse(node);
11     ...
12   } else{
13     sendExceptionToUI(IneffectiveCorrectiveTreatment);
14     ...
15   }
16 }
17 // Monitors the side effects of the treatments in the executing list
18 runTimeMonitoring(TPCTreeNode root) {
19   TPCTreeNodeSet affectedNodeSet←∅;
20   // Monitoring side effect
21   for each treatment in the executingList{
22     if(sideEffects affect other treatments || sideEffects invalidate the
         preconditions){
23       affectedNodeSet←affectedNodeSet∪getAffectedNodeSet(treatment);
24       setTreeNodeStatus(affectedNodeSet);
25   }
26 ...
27 }
```

executing list and requests the medical devices to perform the treatment. The system needs to check the expected response after a time interval, specified by the medical staff, as shown in the line 8–15 of Algorithm 3. The details of checking expected responses will be explained in the next phase. In addition, the system periodically monitors or requests the medical to check the potential side effects of

the treatments in the executing list. The side effects may lead to the following situations:

2-a The side effects of a treatment interfere the other ongoing treatments. Specifically, the side effects cause the patient's physiological measurements changing in an opposite direction to the expected responses of other treatments.

2-b The side effects invalidate the previously satisfied preconditions in the TPC tree.

In both cases, the system will highlight the interfered treatments and the corresponding preconditions in the TPC tree and send an exception to the medical staff, as shown in the line 22–25 of Algorithm 3. The medical staff can adjust the existing treatments, such as increasing or decreasing the drug dosage, or specifying alternative treatments. The system then updates the tree, as described in the previous phase.

After the system informs the side effects to the medical staff and updates the TPC tree with their approval, the system restarts the post order execution.

3. Checking expected responses phase: As explained in the previous phases, the system must check patient's conditions against the expected responses of the treatment when the timer fires. If the patient conditions are as expected, the system removes the corresponding treatment node from the TPC tree and executes the next treatments based on the post order of the TPC tree. If the patient conditions do not improve as expected, the system highlights the unsuccessful preconditions and the corresponding corrective treatments on the TPC tree for the medical staff. The medical staff can specify an alternative corrective treatment, and the system updates the TPC tree accordingly and restarts the post order execution.

By following the above procedures, the system preforms the treatments and corrects the preconditions in a bottom-up manner. Even if the side effects adversely affect other treatments or invalidate the preconditions, the system is capable of updating the TPC tree and let medical staff change the treatments.

4.3 Correctness of the Treatment Validation Protocol

In this section, we prove the correctness of the developed treatment validation protocol.

Theorem 4 *Under the proposed protocol, a treatment is performed only if all preconditions of the*

treatment are satisfied.

Proof.

Proof by contradiction. We assume that a TPC tree node n_α is executed, but one of the preconditions of n_α is not satisfied. Since the protocol adopts post-order execution, child nodes of n_α must be executed before n_α can be executed. A child node is removed from the tree if and only if its expected responses are satisfied. Consequently, the preconditions of n_α must all be satisfied before the child nodes are removed from the tree. In addition, according to the line 8–9 of Algorithm 3, the precondition is checked again before the treatment is performed. We arrive at contradiction. □

We then prove that our protocol can correct the unsatisfied preconditions and reach the root node, which is the treatment that the medical staff intend to perform in the beginning.

Definition 7 *An TPC tree is **well-formed** if each unsatisfied preconditions have a tree node for the corrective treatment.*

Theorem 5 *The root node of a well-formed TPC tree is reachable if the corrective treatments are effective and the preconditions are not invalidated by the side effects.*

Proof.

Proof by induction. Let N be the number of preconditions in a TPC tree.

Base case: When N = 0, the statement is trivially true.

Induction step: Assume the statement is true for $1 \leq N \leq k$, Consider N = k + 1. There are two possible cases.

Case 1: The (k+1)th precondition is satisfied, the protocol starts post order execution as if there are only k preconditions in the tree.

Case 2: The (k+1)th precondition is not satisfied. Since the tree is well-formed, by definition, the (k+1)th precondition has a corresponding corrective treatment node. Moreover, the corrective treatment can successfully correct the (k+1)th precondition because the corrective treatments are effective and the preconditions are not invalidated by the side effects. After the (k+1)th precondition is satisfied, the

protocol traverses the tree as if there are only k preconditions.

In either case, the induction case holds. Therefore, by induction, the statement is true. □

Lemma 6 *The checkPrecondAndBuildTree function, as shown in Algorithm 2, either builds a well-formed TPC tree or raises an exception.*

Proof.

Proof by contradiction. Assume Algorithm 2 build a non-well-formed tree, and no exception is raised. Suppose a precondition p_α, is not satisfied and there is no corresponding corrective treatment node in the TPC tree. The status of the precondition p_α will be set to *UNSATISFIED*, as shown in line 8–12. After the medical staff specify the corrective treatments, the protocol checks if all the unsatisfied preconditions have corresponding corrective treatments. If not, an exception is sent to the medical staff, as shown in line 21–25. We reach a contradiction. □

Theorem 7 *Suppose side effects of a treatment invalidate any precondition and make the TPC tree become non-well-formed. The protocol updates the tree to be well-formed if the medical staff correctly specifies the corrective treatments.*

Proof.

The protocol periodically monitors the potential side effects of the treatments in the executing list and checks if any precondition is invalidated. As shown in the line 22–27 of Algorithm 3, if any previously satisfied precondition is invalidated due to the side effects, the protocol ”highlights” the affected tree nodes and preconditions. The protocol, then, calls the *checkPrecondAndBuildTree()* to update the tree. According to Lemma 6, since the medical staff correctly specifies the corrective treatments, the updated tree is well-formed. □

The above theorems prove that our protocol validates treatments in respect of validating preconditions, monitoring side effects and checking expected responses and adapts the TPC tree to the dynamics of patient conditions and non-determinism of the treatments.

4.4 Cardiac Arrest Resuscitation Case Study and Verification

We use cardiac arrest resuscitation as a case study and model the proposed protocol in UPPAAL. The system consists of the following models: user interface, validation protocol, side effect monitor, EKG monitor, defibrillator, IV Pump, blood pH monitor, and urine flow rate sensor. The models communicate using UPPAAL synchronization channels and shared variables.The user interface model follows the three-step resuscitation procedure and contains a list of pre-defined preconditions and treatments, such as activating defibrillator and injecting epinephrine, as described in the previous section. The medical devices send the patient's physiological measurements, which are modeled as non-deterministic transitions, to the validation protocol. In addition, the medical devices also receive the treatment requests from the protocol and change the states accordingly.

We show part of the verified safety and correctness properties in Table 4.1. We verified two sets of properties in UPPAAL: medical safety properties and protocol correctness properties. The medical safety properties capture the safety requirements of the resuscitation scenario, which are given by the medical staff. For example, safety property *P2* can be checked by the UPPAAL temporal logic formula: *A[] ValidationProtocol.IVPumpEPI imply*
LabBloodPH.Value $>$ *7.4 && UrineSensor.Vaule* $>$ *12*. The above formula verifies that for all reachable states, the *IVPumpEPI* state implies that the *Value* of *LabBloodPH* is larger than 7.4 and the *Value* of *UrineSensor* is larger than 12. On the other hand, the protocol properties guarantee the correctness of the proposed protocol. For instance, property *P8* can be checked by the formula:
SideEffectMonitor.sideEffectOccur==true $-->$ *SideEffecMonitor.UpdateTree &&*
isWellFormed(RootNode), where *isWellFormed* is a boolean function to check if the TPC is well-formed. The above formula verifies that if variable *sideEffectOccur* is true, the *UpdateTree* state is eventually reached and *isWellFormed* returns true. Other safety and correctness properties can be verified with similar formula. In summary, we demonstrate that the proposed treatment validation protocol is correctly designed to guarantee safety and correctness properties.

Table 4.1: Verified properties of the resuscitation scenario

	Verified Properties
Medical safety properties	P1: Defibrillator is activated only if the EKG rhythm is a shockable one and airway and breathing is normal.
	P2: Epinephrine is injected only if the blood pH value is larger than 7.4 and urine flow rate is higher than 12 mL/s.
	P3: If the side effect of sodium bicarbonate adversely affects the breathing, the tree is updated with a new treatment node for assisted ventilation.
	P4: If epinephrine does not make patient's EKG rhythm become shockable, the tree is updated with an alternative treatment node for drug *vasopressin*.
Protocol properties	P5: There is no deadlock in the system.
	P6: A treatment is performed only if all its preconditions are satisfied.
	P7: If side effect does not occur, the root node of the TPC tree is added to the executing list
	P8: If side effects invalidate a precondition, the TPC tree is updated and well-formed.

4.5 Summary and Future Work

In this chapter, we propose a treatment validation protocol to enforce the correct execution sequence regarding precondition validation, side effects monitoring, and expected responses checking. The proposed TPC tree structures the preconditions and corrective treatments, which provides a logical path for medical staff to keep track of the medical procedure. In collaboration with medical staff, the proposed protocol adapts the TPC tree to the dynamics of patient conditions and non-deterministic behavior of treatments. Therefore, the preventable medical errors due to invalid treatments can be reduced.

As future work, we would like to collect medical error case studies from Food and Drug Administra-

tion (FDA) and evaluate the reduction of the medical errors with the proposed protocol. Consequently, we can provide quantitative results of the efficiency of the proposed protocol.

Chapter 5

Reducing Cyber Medical Treatment Complexity: A Safe Workflow Adaptation and Validation Protocol for Medical Cyber-Physical Systems

In this chapter, we will address another dimension of cyber medical treatment complexity: workflow adaptation. Medical workflow codifies best practice guidelines to help physicians timely and correctly perform treatments. In medical cyber-physical-human systems, synchronizing supervisory medical systems, physicians' behavior and patient conditions in compliance with best practice workflow is essential for patient safety. However, patient conditions change rapidly and asynchronously. According to a study, physicians are interrupted frequently due to patient adverse events. Those adverse events may cause distraction and further result in medical errors [26, 54]. Therefore, medical workflows must be interrupted and adapted accordingly. Nevertheless, adapting a workflow without validating safety requirements may cause safety hazards. For one thing, some treatments may cause severe adverse interactions if they are performed simultaneously. For another, at certain workflow states, the workflow is not safe to be interrupted. We propose a workflow adaptation and validation protocol [96] to safely adapt workflows with the consideration of physician-in-the-loop. Note that instead of automatically

performing treatments, our system utilizes pathophysiological models and workflows to validate safety requirements and, thus, mitigate safety hazards.

5.1 Physical Models and Definitions

Definition 8 *Workflow is modeled as a timed automaton, W = $< Q, \Sigma, C, E, q_0 >$, where*

Q is a set of the states of W. Σ is a set of actions of W.

C is a set of the clocks.

$E \subseteq Q \times \Sigma \times B(C) \times P(C) \times Q$ is a set of edges, called transitions of W, where

B(C) is the set of boolean clock constraints involving clocks from C, and P(C) is the powerset of C.

$q_0 \in Q$ is the initial state.

More details of timed automata can be found in [4, 13].

It is worthwhile mentioning that a workflow serves as guidance to help physicians follow the best practice. The developed system does not automate the execution of workflows. In order to develop an adaptation and validation protocol, we further extend the model to include patient adverse events information and preconditions.

Definition 9 *A **workflow state** is defined as a tuple $<T, \sigma, \delta, \mu >$, where*

T is the type of the state, and $T \in \{Operation, Treatment\}$

σ is an operation or a treatment that should be performed when reaching this state,

δ is a set of physiological conditions, which specifies abnormal patient conditions that can be handled in the state. An empty set indicates that the state cannot be interrupted.

μ, is the effective time of the state. If the state type is Operation, μ specifies the expected execution time. If the state type is Treatment, μ specifies the time interval between the treatment is being performed and the treatment has no further effect on the patient[1].

For example, the state of *Secure airway* is defined as $<$ Operation, (Secure airway), $\emptyset$, 5 minutes$\}$ $>$. Note that securing airway is usually a top priority operation for many medical scenario, so δ is an

[1]Different treatments have different effective time. For example, the effective time of a drug is decided based on the drug metabolism.

empty set. The state of *Epinephrine* is defined as $<$ Treatment, (Administer Epinephrine), $\{$(Heart rate $\geq$ 180)$\}$, 4 minutes $>$.

Definition 10 ***AdaptationPrecondition (AP)*** *is defined as a tuple <DS, PCS, ITS>, where*
DS (Device Set) is a set of medical devices that are required to perform the treatments or operations after the adaptation,
PCS (Physiological Condition Set) is a set of physiological condition that must be satisfied before starting the adaptation,
ITS (Incompatible Treatments Set) is a set of treatments that may cause adverse interactions with the treatment that will be performed after the adaptation.

For instance, the AP of switching to an asystole workflow is $<\{$Infusion pump, EKG monitor, oximeter$\}$, $\{$(Heart rate $\leq$ 30), (Blood pressure $\leq$ 20)$\}$, $\{$Sotalol[2]$\}$ $>$.

5.2 Workflow Adaptation and Validation Protocol

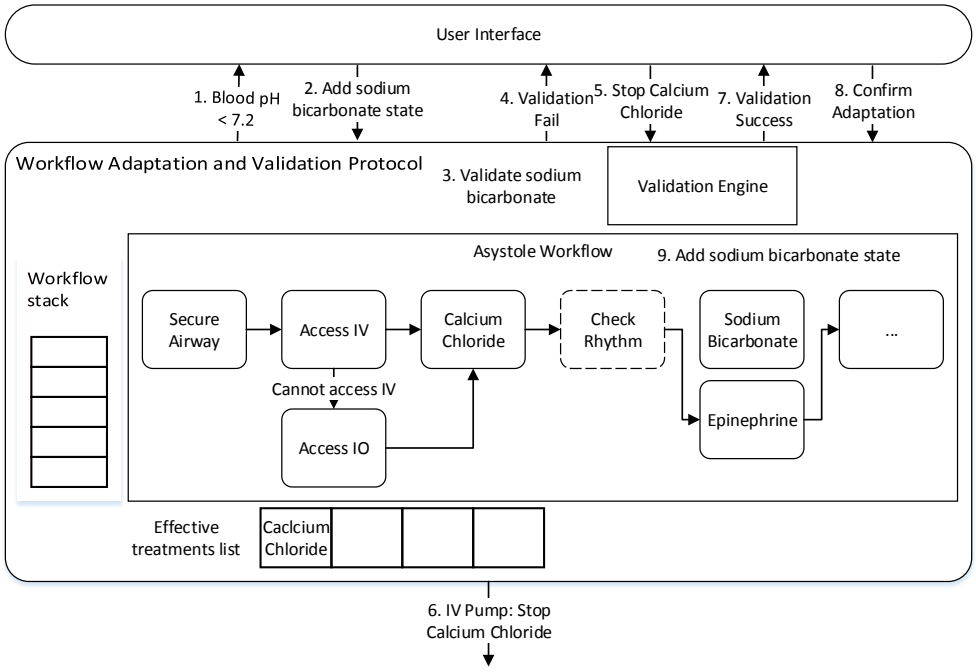
5.2.1 Illustrative Examples

Let us use the following scenario to illustrate how adaptation and validation are performed with the pathophysiological and workflow models introduced in the previous section. Suppose a patient with asystole[3]. The system validates the preconditions of executing an asystole workflow. First, the system checks if the required devices, including Electrocardiography (EKG) monitor, infusion pump and pulse oximeter, are connected to the system. Second, the system checks if the EKG monitor shows a flatline and heart rate from the oximeter is consistent with the asystole diagnosis. Third, the system checks the patient is not currently using any drug that may cause adverse interaction with epinephrine, which is a drug that will be used in the asystole workflow. If all the preconditions are satisfied, the system switches to the asystole workflow.

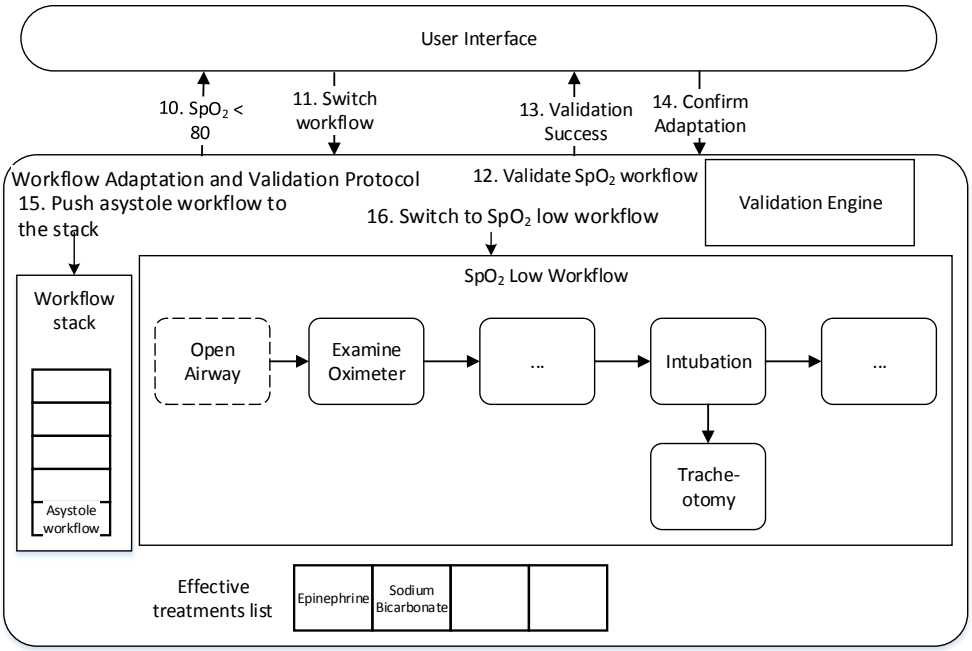
While the physicians check the EKG rhythm, the system obtains the patient's lab test results, which indicating acidosis (patient's blood pH lower that 7.2). Suppose the workflow state of *Check Rhythm* is

[2]Sotalol is usually used to treat fast heartbeat and is not suitable for a patient with asystole.

[3]Asystole, colloquially known as flatline, is one type of arrhythmia, in which patient's heart stop beating and hence no cardiac output or blood flow.



(a) Adding a state to the workflow



(b) Switching to another workflow

Figure 5.1: The adaptation and validation protocol. The workflow state with dashed rectangle indicates the physicians are currently executing that state.

interruptible to handle acidosis. The system raises a patient adverse event to the physicians, and physicians specify adapting the workflow by adding a state, administering sodium bicarbonate, as illustrated in step 1–2 of Figure 5.1(a). The system validates the preconditions by checking that the patient's airway is secured and calcium chloride is not currently administered. However, calcium chloride is continuously administered to treat high potassium level in the patient's blood. The system alerts the physicians of potential adverse drug interaction between sodium bicarbonate and calcium chloride. In this situation, physicians requests to stop administering calcium chloride. The system, then, sends a command to the IV pump to stop calcium chloride, as illustrated in step 3–6 of Figure 5.1(a). Once the preconditions are satisfied, the system adds the *administering sodium bicarbonate* state to the asystole workflow.

After a few minutes, the patient's oxygen saturation level (SpO_2) suddenly drops below 80%, which is lower than the threshold.The system raises a patient adverse event, and physicians decide to interrupt the asystole workflow and treat low SpO_2 first. The system validates the preconditions of switching to SpO_2 low workflow, as illustrated in step 10–14 of Figure 5.1(b). If all the preconditions are satisfied, the system pushes the asystole workflow into a stack and switches to the SpO_2 low workflow, as illustrate in step 15–16 of Figure 5.1(b). Note that workflow stack is for handling "nested" patient adverse events, i.e. a patient adverse event is raised in the middle of handling another patient adverse event. When the patient's SpO_2 level is back to normal, which is usually above 90%, and the SpO_2 low workflow is completed, the system notifies the physicians about the performed treatments and the changes of patient's physiological conditions. The physicians, afterward, decide to resume the previous asystole workflow. The system pops the asystole workflow from the stack and must validate the preconditions again since the patient conditions keep changing. However, it is possible that the previous workflow may not be fit the current patient conditions. The physicians may request to discard the asystole workflow and switch to another one.

5.2.2 Protocol Design

The developed workflow adaptation and validation protocol has the following four phases.

1. Raising patient adverse events and re-synchronizing phase

The pseudo code is show in Algorithm 4. As explained in the previous section, the protocol dynamically monitors the physiological measurements and raises patient adverse events if any measurement becomes abnormal. When patient adverse events are raised, there are two possible cases:

Case 1-a: The patient adverse event can be handled at the current workflow state.

The system will inform the physicians of the patient adverse event with all physiological measurements and device settings. Moreover, the system also lists all the ongoing and unfinished workflow states, as shown in line 2–8 of Algorithm 4. In this way, the system re-synchronizes the workflow states to the patient physiological conditions and provides comprehensive information to the physicians, because medical decisions based on partial information may cause safety hazards [93]. According to the provided information, physicians can decide the process of adaptation, which is either adding/removing/updating the states of the current workflow or switching to another workflow. Then, the protocol enters the *validating preconditions* phase.

Case 1-b: The patient adverse event cannot be handled at the current workflow state.

The system will add the raised adverse event to a pending list and alert the physicians. When the workflow state is changed, the system re-examines the pending patient adverse events and notifies the physicians if any patient adverse event can be handled. Note that the system also allows physicians to postpone handling patient adverse events within a time frame. However, if any patient adverse event is not handled within a time interval, the patient adverse event will be raised again, as shown in line 10–12 of Algorithm 4.

Note that it is possible that multiple patient adverse events are raised simultaneously. Our protocol does not decide which workflow should be executed, and it is the physicians' responsibility to provide the appropriate workflow. If there is a workflow to handle multiple patient adverse events, the physicians can request our system to execute the workflow. On the other hand, the physicians can prioritize the patient adverse events and provide the workflow for handling the most urgent patient adverse events.

2. Validating preconditions phase

At this phase, the system validates the preconditions according to the adaptation process specified by the physicians, as shown in line 4–22 of Algorithm 5.

The validation is performed to mitigate three types of safety hazards described in the previous section. First, the required devices are connected to the system and configured according to the new workflows.

Second, the patient conditions are compatible with the new workflow. Third, preformed or performing treatments will not cause adverse interactions with the treatments specified in the new workflow. If all the preconditions are satisfied, the system notifies the physicians that the adaptation is safe to be processed. If physicians approve the adaptation, the system goes to the *adapting* phase. Otherwise, if any precondition is not satisfied, the system rejects the adaptation and alerts the physicians along with the evidence of the unsatisfied preconditions. However, the developed system allows the physicians to override the validation results. The reason is that medical decisions require sophisticated medical knowledge and much medical information cannot be monitored and processed by the developed system. Consequently, our system helps physicians to validate preconditions instead of replacing physicians' judgments.

3. Adapting phase

Case 3-a: Adding/Removing/Updating the states:

The workflow state is adapted accordingly. In addition, the protocol also commands the corresponding medical devices to change the settings. For instance, if the adaptation process requires to stop administering or changing the dosage of a drug. The protocol sends a command to the infusion pump to change the device settings if the physicians approve it.

Case 3-b: Switching to another workflow:

The current workflow is pushed into a stack for future reference. The system starts execute the new workflow specified by the physicians

4. Completing workflow phase

The pseudo code is shown in Algorithm 6 When the workflow is completed, the system sends a summary of the workflow, including performed treatments, changes of physiological conditions, and total execution time, to the physicians. There are two possible cases.

Case 4-a: Resuming the previous workflow

The physicians decide to resume the previous workflow. The system pops the workflow from the stack, validates the preconditions, and resumes it.

Case 4-b: Switching to another workflow

Since patient conditions keep changing, the previous workflow may not suit the current patient conditions anymore, and physicians need to switch to a new workflow. The workflow on top of the stack is

discarded. Then, the system validates preconditions of the new workflow and executes it.

In addition to the above adaptation and validation procedures, the system also dynamically monitors the patient conditions and keeps track of the effective time of the treatments. First, the system periodically checks the physiological measurements from the medical devices. If measurements change significantly or become abnormal, a patient adverse event is sent to the adaptation and validation protocol. Second, the system maintains a list of effective treatments. When a treatment is performed, the treatment is inserted into the list, and a timer is initiated according to the effective time, μ. The treatment is removed from the list when the timer fires, which suggests that the treatment has no or little effect on patient. In this way, the system can keep track of the effective treatments and avoid potential adverse treatment interactions.

Algorithm 4 Raising patient adverse events and re-synchronizing

```
1  void recvAdverseEvent(PatientAdverseEvent event)
2  {
3    if(event ∈ currentState.δ){
4      for(each workflow state){
5        if(state.status == INCOMPLETE ||state.statu == ONGOING )
6          state_list.add(state);
7      }
8      notifyUI(incomplete_list∪measurements_set);
9    } else{
10     addToPendingList(event);
11     notifyUI("Pending adverse event");
12     setupTimer(event, ReRaiseInterval);
13   }
14 }
```

5.2.3 Design Rationale and Limitations

In this section, we discuss the rationale behind the developed protocol as well as the limitations and potential solutions.

First, the proposed protocol requires physicians to specify the workflow adaptation processes. It

Algorithm 5 Validating preconditions

```
1  // return true if all the preconditions are satisfied; otherwise, return false
2  boolean validatePreconditions(AdaptationProcess adaptation)
3  {
4      // check minimums required devices
5      if(!(deviceSet ⊆ adaptation.DS)){
6          notifyUI("Missing required devices");
7          return false;
8      }
9      // check patient's physiological conditions
10     for(each PC ∈ adaptation.PCS){
11         if(PC is not satisfied){
12             notifyUI("Inconsistent patient conditions");
13             return false;
14         }
15     }
16     // check incompatible treatments
17     for(each performed or performing treatment t){
18         if (t ∈ adaptation.ITS){
19             notifyUI("Incompatible treatment");
20             return false;
21         }
22     }
23     return true;
24 }
```

requires extra time and effort for physicians to use the developed system. However, we argue that the time and effort are worthwhile compromises for performing computer aided validation in order to mitigate safety hazards and further enhance patient safety. Moreover, the interaction with medical staff is necessary, because much critical medical information, such as, patient conditions and diagnosis are described in natural language, which cannot be processed by the system. To improve usability, when a patient adverse event is raised, a list of well-established adaptation processes for the corresponding physiological conditions is displayed. We are closely working with medical staff to collect routine and

Algorithm 6 Completing workflow

```
1  void completeWorkflow()
2  {
3    ...
4    notifyUI(preformed treatment lists);
5    ...
6    if(resume precious workflow){
7      Workflow workflow = popWorkflow();
8      if(validatePreconditions(...)){
9        currentWorkflow = workflow;
10     } else{
11       ...
12     }
13   } else if(switch to a new workflow){
14     popWorkflow();
15     validatePreconditions(...);
16     ...
17   }
18 }
```

well-established medical practices that can benefit from such improved adaptation choice lists.

Second, our protocol does not consider the potential failure of medical devices, supervisory controllers, and communication channels. We assume that each medical device has been certified and approved by Food and Drug Administration (FDA). If any medical device malfunctions, our system relies on medical staff to replace the malfunctioned device. On the other hand, the failure of communication channels is another major issue. When the communication channels fail, the developed system cannot dynamically monitor the patient conditions and validating the preconditions of adaptation. This fault-tolerance issue is addressed in [50], and the proposed protocol can cooperate with those mechanisms. The fundamental concept is that each workflow has a pre-specified fail-safe state, which consists of the safe settings for each medical device. For instance, a fail-safe state of the asystole workflow is deactivating the defibrillator and maintaining injection of epinephrine. At run time, the system generates contingency plans that will change each medical device setting to a safe one when communication

fails.

Third, in this work, the proposed protocol executes the workflow one at a time and does not support concurrent workflow execution. The reason for limiting the concurrent workflow execution is that medical staff generally execute one workflow at a time if applicable. For example, **A**irway, **B**reathing, and **C**irculation is a commonly accepted sequence for assessment and treatment of patients[4]. Concurrent workflow execution significantly increases physicians mental workload and may even cause safety hazards due to adverse interactions between the workflows. However, in certain situation, for instance, in trauma surgery, a patient suffers from multiple organ injuries, such as brain damage, lung collapse, and kidney failure. In this case, physicians may require to execute multiple workflows concurrently in order to treat the organ injuries. Therefore, a more complicated protocol for synchronizing and validating multiple workflows, physicians' actions and patient conditions is required. This is a serious challenge to be addressed in our future work.

5.3 Design Pattern and Protocol Instantiation

In this section, we structure the workflow adaptation and validation protocol as a design pattern. Figure 5.3 shows the class diagram of the overall software architecture. (We only show the core components and methods in the figure for readability.) The developed design pattern consists of reusable software components, i.e. *ValidationEngine* and*RuntimeMonitor*, and abstract templates, i.e. *AbstractWorkflow* and *AbstractAdaptationProcess*.

- *AbstractAdaptationProcess* provides a template for system developers to design an adaptation process. The most implement part is an abstract method, *adaptProcess*. Depends on the medical scenario and exceptions, the system developers need to implement this method, which is either adding/removing/updating workflow states or switching to another workflow. In addition, *AbstractAdaptationProcess* encapsulates the preconditions of the adaptation, which are required devices, physiological conditions, and potential conflict treatments.

- *AbstractWorkflow* declares a set of generic variables and methods to model medical workflows,

[4]Other variations, such as CAB and ABCDE, are proposed for different medical scenario [55]

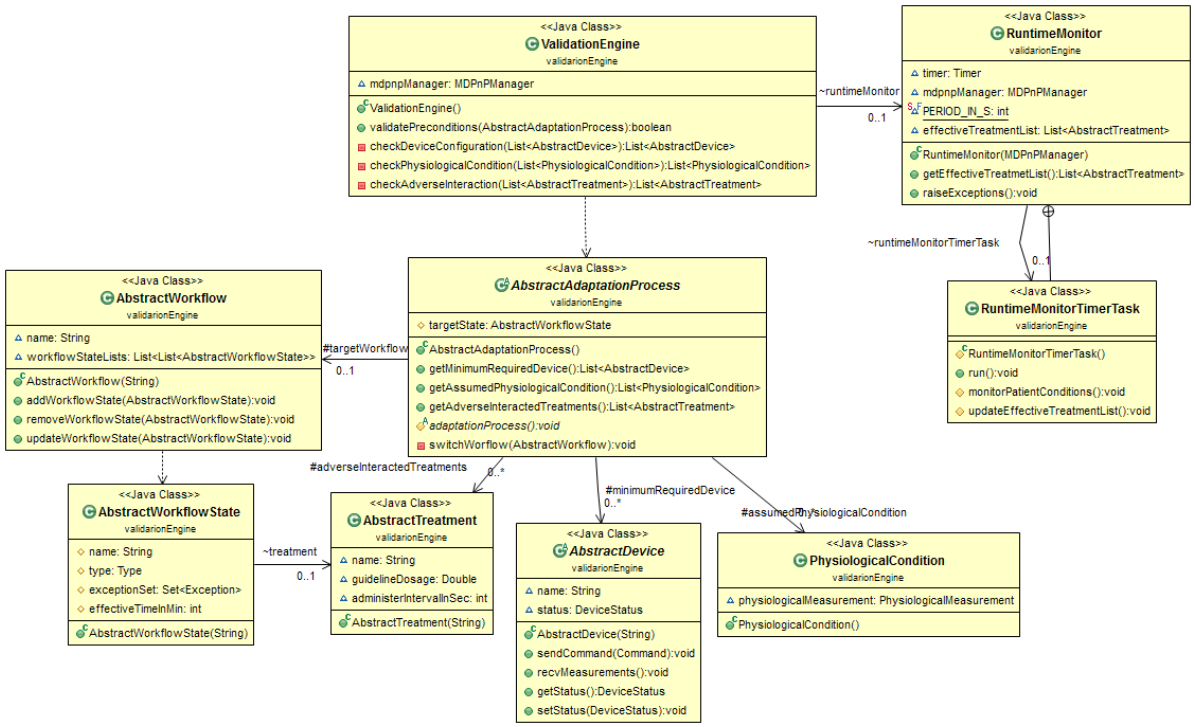


Figure 5.2: Abstract Class Diagram

and provide interfaces to allow *AbstractAdapataionProcess* to add/remove/update a workflow state.

- *ValidationEngine* implements the adaptation and validation protocol based on the preconditions specified in *AbstractAdaptationProcess*, as described in Section 5.2.

- *RuntimeMonitor* is a periodic process, which monitors the physiological measurements from *AbstractDevice* and raises exceptions if measurements change significantly or become abnormal.

In order to instantiate the developed pattern to a medical scenario, system developers must implement scenario-dependent workflows and adaptation processes in collaboration with physicians. We show the class diagram of a concrete implementation of cardiac arrest resuscitation in Figure 5.3. Following are the detail implementation strategies:

1. A set of workflows, such as asystole and ventricular fibrillation workflows, must be codified in accordance with the best practice guidelines. System developers should cooperate with physicians

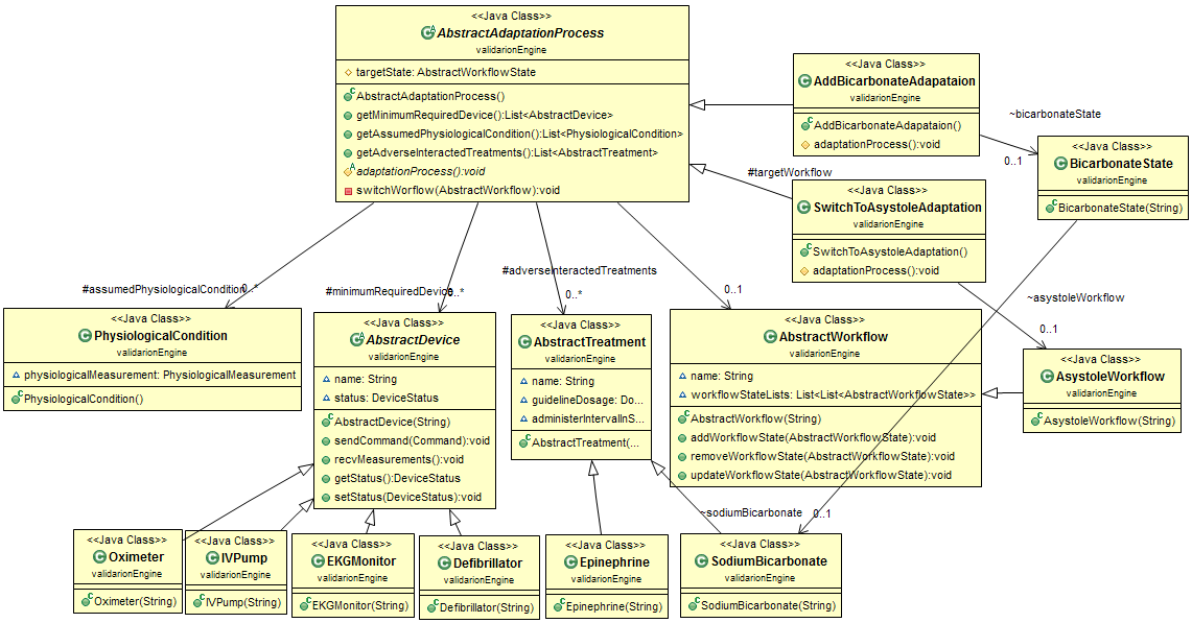


Figure 5.3: Instantiation of Cardiac Arrest Resuscitation

to verify the correctness of the codified workflow. Physiological correctness of the workflow is out of the scope of the pattern.

2. The concrete medical devices, e.g. defibrillator and EKG monitor, must be and extended from *AbstractDevice*. *AbstractDevice* class serves as a proxy of the physical medical devices and provides interface for *AbstractAdaptationProcess* to receive physiological measurements and send commands. The communication between *AbstractDevice* and physical medical device can be achieved through Medical Device Plug-and-Play (MDPnP) framework [5].

3. The treatments that may be used in the medical scenario, e.g. epinephrine and sodium bicarbonate, must be extended from *AbstractTreatment*. Moreover, *AbstractTreatment* contains best practice guideline dosage and administering interval to assist physicians to perform the treatments in accordance with the guidelines.

4. For each exception that may occur during the medical scenario, an adaptation process should

[5]More details of the implementation of MDPnP programs can be found at `http://sourceforge.net/projects/mdpnp/`

be specified to adapt the workflow as well as the corresponding preconditions. For instance, in Figure 5.3, we show two adaptation processes *addBicarbonateAdaptation* and *switchToAsystoleAdaptation* to handle acidosis and EKG rhythm changing to asystole, respectively.

By following the above strategies, the proposed design pattern can be instantiated to a medical scenario to support workflow adaptation and validation.

5.4 Cardiac Arrest Resuscitation Case Study and Verification

In this section, we first describe a cardiac arrest resuscitation scenario as our case study. We, then, model the proposed protocol in UPPAAL [13], which is a model checking tool, to verify the safety and correctness properties.

5.4.1 Cardiac Arrest Resuscitation Case Study

Cardiac arrest is the abrupt loss of heart function and can lead to death within minutes. Cardiopulmonary resuscitation (CPR) and medications, such as epinephrine and calcium chloride, are used to provide circulatory support. Defibrillator is used to deliver a therapeutic dose of electrical energy to the heart in order to reestablish normal heart rhythms. American Heart Association (AHA) provided resuscitation guidelines for the urgent treatment of cardiac arrest [35]. The guidelines contain the workflows for treating six types of life-threatening arrhythmia: ventricular fibrillation (VF), ventricular tachycardia (VT), asystole, pulseless electronic activity (PEA), sinus bradycardia, and sinus tachycardia. Each workflow consists a set of medical procedures, including secure airway, defibrillation and rhythm check, and a set of drugs, including epinephrine, sodium bicarbonate, and calcium chloride, with dosage and administering interval. In addition, patient's physiological measurements may become abnormal, for instance, sudden drop in SpO_2 level. Therefore, physicians need to adapt the workflow to react to the rhythm changes and abnormal physiological measurements. However,the adaptation of the workflows may cause safety hazards. We gather a list of safety hazards, shown in Table 5.1, by interviewing our physician collaborator.

Table 5.1: Safety hazards of workflow adaptation during cardiac arrest resuscitation

Workflow adaptation safety hazards	
Device configuration hazards	H1: The system switches to a ventricular fibrillation or ventricular tachycardia workflow, but a defibrillator is not yet connected to the system.[6]
	H2: The system adds a workflow state of administering sodium bicarbonate, but an infusion pump is not yet connected to the system.
Patient physiological condition hazards	H3: Sodium bicarbonate is given before the patient's airway is secured.
	H4: The system switches to a sinus tachycardia workflow, but patient's heart rate is slower that 150 bpm[7].
	H5: The system switches to a sinus bradycardia workflow, but patient's heart rate is faster that 50 bpm[8].
Adverse treatment interaction hazards	H6: Sodium bicarbonate and calcium chloride is administered simultaneously with the same infusion pump.
	H7: Sotalol is administered while the system switches to a sinus bradycardia workflow.[9]
	H8: Atropine is administered while the system switches to a sinus tachycardia workflow.[10]

5.4.2 UPPAAL Models and Verification

We model the proposed protocol in UPPAAL [13]. The system consists of the following models: user interface, adaptation and validation protocol, EKG monitor, defibrillator, IV Pump, and blood pH monitor. The communications between the models are through UPPAAL synchronization channels and shared variables. The user interface contains a list of pre-defined workflows, and the workflow is

[6] When patients' heart rhythm changes ventricular fibrillation or ventricular tachycardia, early defibrillation is critical for improving patients' survival rate.

[7] According to AHA guidelines [35], tachycardia is the heart rate abnormally exceeding 150 bpm and a set of drugs may be administered to slowdown the patient's heart rate.

[8] According to AHA guidelines [35], bradycardia is the heart rate abnormally slower than 50 bpm.

[9] Sotoal is usually used to treat fast heart rate and not suitable for treating sinus bradycardia.

[10] Atropine is usually used to treat slow heart rate and not suitable for treating sinus tachycardia.

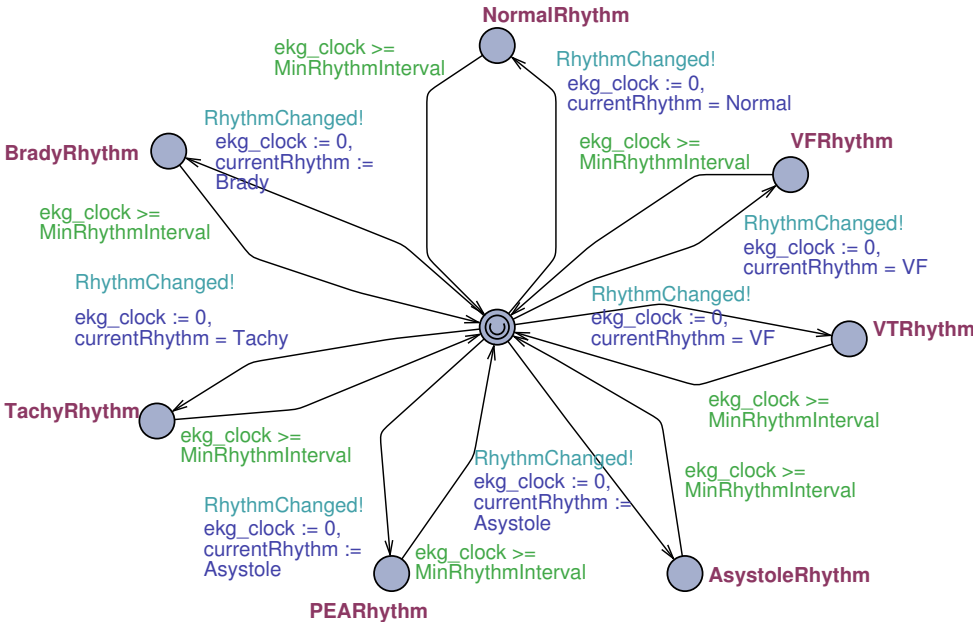


Figure 5.4: The EKG monitor in UPPAAL

Table 5.2: Protocol correctness properties

Protocol correctness properties	P1: There is no deadlock in the system.
	P2: A patient adverse event is raised to the user interface only if the current workflow state is interruptible to handle the adverse event.
	P3: An adaptation is performed only if all the preconditions are satisfied.

switched according the states (rhythms) of EKG monitor. The user interface also has a list of patient adverse events and the corresponding adaptation process. The medical devices send the physiological measurements, which are modeled as non-deterministic transitions with timing constraint, to the adaptation and validation protocol. In addition, the medical devices receive the treatment requests from the protocol and change the states accordingly.

We show two UPPAAL models of the developed system in Figure 5.4 and Figure 5.5. The other UPPAAL models share the similar structures. Figure 5.4 shows the EKG monitor model. The EKG model changes the rhythm non-deterministically, and the the minimum time interval between rhythm changes is defined by a variable *MinRhythmInterval*. When the rhythm changes, the global variable *currentRhythm* is updated and a patient adverse event, *RhythmChanged*, is raised to the adaptation and

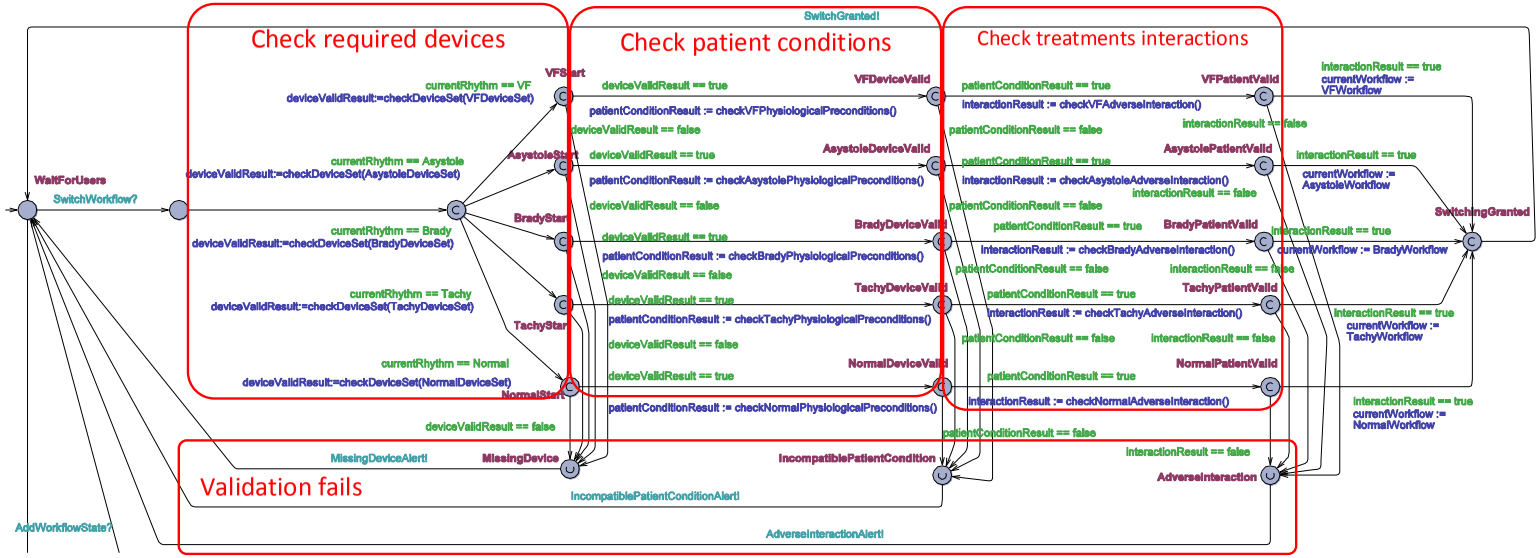


Figure 5.5: Adaptation and validation protocol in UPPAAL

validation protocol. Other medical devices, such as pulse oximeter, have a similar structure. The design of the adaptation and validation protocol, as shown in Figure 5.5, reflects the pseudocode, described in Section 5.2.2. We only show the core part of the protocol, validation procedure, for readability. When the protocol receives a *SwitchWorkflow* request from the user interface, the protocol first validates if all the required medical devices has been connected to the system. If any device is missing, a *MissingDeviceAlert* is sent to the user interface. Second, the protocol validates the physiological measurements from the medical devices is compatible with the preconditions specified in the workflow. If there is any incompatible physiological condition, an *IncompatiblePatientConditionAlert* is sent to the user interface. Third, the protocol validates the potential treatment interactions. If any treatments in the effective treatment list may have adverse interactions to the treatments specified in the workflow, an *AdverseInteractionAlert* is sent to user interface. If all the preconditions are satisfied, a *SwitchGranted* message is sent to the user interface.

We use UPAAL to formally check that none of the safety hazards, described in the pervious section, occur in our system. The main temporal operators for verifying safety properties supported by UPPAAL are *A []* ϕ, which means ϕ is satisfied in every state along every execution path, and *E []* ϕ, which means ϕ is satisfied for all states along at least one execution path. For instance, *H5* can be checked by the formula: *A[] not (nextWorkflow == Bradycardia) or HeartRate* $<$ *50)*. The above formula verifies that for all reachable states, the system switch to a bradycardia workflow only if patient's heart rate is lower than 50. Moreover, we also verify a set of protocol correctness properties, which are shown in Table 5.2. For instance, *P3* can be checked by the formula:
A[] not(av_protocol.SwitchingGranted) or (av_protocol.deviceValidResult == true and av_protocol.patientConditionResult == true and av_protocol.interactionResult == true), where *SwitchingGranted* is a state indicating the workflow approved to be switched by the protocol and *deviceValidResult*, *patientConditionResult*, and *interactionResult* are three boolean variables indicating the validation results of the three types of preconditions, respectively. Other safety and correctness properties can be verified with similar formula. According to the model checking results, we demonstrate that the proposed workflow adaptation and validation protocol is correctly designed to mitigate safety hazards.

5.5 Summary and Future Work

In this chapter, we develop a workflow adaptation and validation protocol to adapt workflows to the patient adverse events. In order to mitigate safety hazards, the proposed protocol utilizes the introduced pathophysiological and workflow models and validates the preconditions of the adaptation. In collaboration with physicians, the validation is performed on device configurations, patient physiological conditions, and potential adverse treatment interactions. We use cardiac arrest resuscitation as a case study to verify the safety and correctness properties of the developed protocol.

As future work, the developed protocol should support concurrent workflow execution by providing synchronization and adaptation mechanisms to assist physicians to treat multiple diseases and organ systems. For evaluating usefulness of the developed system, we would like to collect medical error case studies from Food and Drug Administration (FDA) and evaluate the reduction of the medical errors with the proposed protocol.

Chapter 6

Reducing Cognitive Load Complexity: Supporting Emergency Medical Care Teams with a Best Practice Guidance System

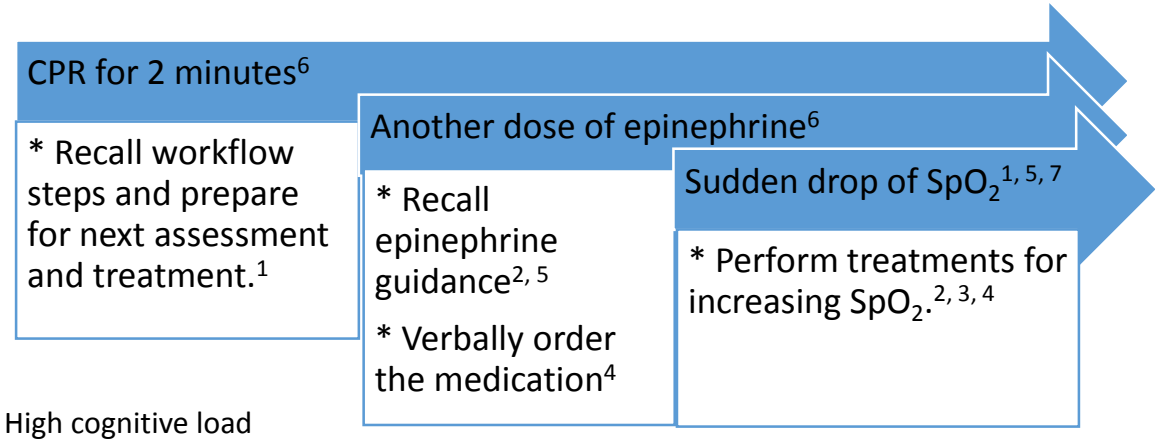
In the previous chapters, we have proposed several protocols to reduce both verification and cyber medical treatment complexity. In this chapter, we turn our focus to human cognitive load complexity, since human operators, i.e physicians and nurses, significantly contribute to safe and effective operations of medical systems. The work of a hospital's medical staff is safety critical and often occurs under severe time constraints. Unfortunately, despite recent advances [18], technological support for cognition and teamwork in healthcare remains considerably behind other industries such as aviation [59]. In many cases, technology design in healthcare does not take advantage of a user-centered design process to ensure that it provides effective cognitive support to medical professionals [70]. Effective design requires that the entire system comprised of humans and both hardware and software components be designed in a coherent way and one informed by known best medical practices. The purpose of the this chapter is to illustrate our approach by describing a prototype design informed heavily by end user input together with a proof-of-concept demonstration and evaluation in the medical domain.

In collaboration with physicians and nurses from Carle Foundation Hospital, we identified and categorized the major sources of cognitive load in cardiac arrest resuscitation. We then designed a medical Best Practice Guidance (BPG) system to reduce medical staff's cognitive load and foster adherence to best practice workflows in real-time (Challenge 4). The resulting support system is analogous to an automobile GPS navigation system, but for effectively "navigating" the pathways leading to safe and effective medical care and treatment. We have seized upon "guidance" terminology to highlight an analogy between our design concept and a GPS-enabled navigation system in an automobile or aircraft. Less skilled or experienced practitioners have best practices readily presented to them to be followed, akin to the way a driver or pilot can follow a computer-recommended route of travel. More skilled or experienced practitioners, in contrast, will still benefit by information on best practices in times of high workload or distraction, and will, like a driver or pilot, be able to override the guidance a system provides when deemed necessary. Ultimate decision making authority remains in the hands of the medical professionals using the BPG system.

6.1 Human Cognitive Load

Let us use a simplified cardiac arrest resuscitation scenario to illustrate the high cognitive load due to concurrent and interactive medical activities. As shown in Fig. 6.1, a physician is performing CPR (Cardiopulmonary resuscitation), which should be continued for at least two minutes according to AHA guidelines, so s/he needs to keep track of the time while maintaining high-quality CPR. In the middle of CPR, in order to improve patient's cardiac function, a drug, called epinephrine, should be administered every three to five minutes; therefore, the medical staff also needs to keep track whether another dose of epinephrine should be administered or not. In addition, when another dose of epinephrine should be administered, the medical staff needs to recall the guideline of epinephrine. According to the guideline, the dosage of epinephrine is 1 mg and may be ineffective when the patient's blood pH value is lower than 7.2. Consequently, the medical staff needs to check if the patient condition, i.e. blood pH, is appropriate for another dose of epinephrine.

Moreover, the patient conditions may suddenly worsen, for instance, the patient's oxygen saturation level (SpO_2) drops below 90. In this situation, the medical staff needs to provide proper treatments,



High cognitive load tasks:

1. Recall workflow steps
2. Recall treatment guidelines
3. Recall diagnosis and performed treatments
4. Recall pending medication orders
5. Assemble clinical information from scattered medical devices and monitors
6. Real-time tracking of temporal progress
7. Real-time tracking of patient's condition changes

Figure 6.1: Assessments and treatments during cardiac arrest resuscitation. The arrows indicate the concurrent medical processes. The text boxes below the arrows are the corresponding assessments and treatments. The numbers shown as superscripts are the sources of cognitive load.

such as assisted ventilation. In addition, during the whole process, the medical staff also has to keep recalling best practice workflows in order to decide the next assessment and treatment. The above high-cognitive-load situation could be worsened as more patient conditions become abnormal and more treatments need to be performed.

In collaboration with physicians and nurses from the Carle Foundation Hospital, we conducted a series of interviews, compiled a list of cognitive tasks that significantly contribute to cognitive load, and categorized these tasks into three categories: information assembling, recall, and real-time tracking.

M1. Assemble clinical information: Medical staff needs to mentally gather patient's physiological measurements from scattered medical devices and monitors, such as EKG monitor, oximeter, and blood

pressure cuff, and diagnose the patient.

M2. Recalls:

- Recall workflow steps: Medical staff can perform treatments more effectively if they can anticipate the next move and follow best practice workflows.
- Recall treatment guidelines: Medical staff needs to validate the preconditions, administer the drug based on the guideline dosage, and set the time for another dose. In an interview, one physicians mentioned

  > *We know that epinephrine could be less effective to a patient with acidosis (blood pH value lower than 7.2); in this situation we may consider using sodium bicarbonate to treat acidosis.*

  Another physician added that

  > *Moreover, before using sodium bicarbonate, there are many other factors(preconditions) we need to consider. For instance, we need to check patient is provided with adequate ventilation.*[1] *Unfortunately, sometimes we just forget to check it because we are so busy doing other things.*

- Recall pending medication order: In clinical practices, physician in charge gives a verbal medication order, and other physicians and nurses need to prepare the drug and administer it to the patient. However, because of the high-stress and chaotic environment, the verbal medication order may be neglected. The medical staff needs to recall if any medication is ordered but has not yet been administered.
- Recall the previous diagnosis and performed treatments: Medical staff needs to recall this information in real-time to decide upon future treatment plans.

M3. Real-time tracking:

- Track real-time changes of the patient conditions: Medical staff needs to keep track of the changes of the patient conditions and perform treatments accordingly.

[1] Giving sodium bicarbonate without providing adequate ventilation to the patient may further worsen acidosis.

- Track temporal progress of the treatment: Certain treatments are time sensitive, and medical staff is required to closely monitor and assure compliance with temporal task constraints. As the Carle Foundation Hospital Director of ICU put it:

  > *Timely and correctly performing treatments is crucial for us to treat patients. In cardiac arrest resuscitation, CPR should continue for at least two minutes. Unfortunately, the environment we deal with is very chaotic and requires multitasking, so correctly keeping track of the time sometimes is hard for us.*

In addition to the above three categories, we also discovered that *calculation* is another major source of cognitive load. Medical staff sometimes needs to calculate the drug dosage based on patient's sex, weight and age. However, in cardiac arrest resuscitation, most of the drug dosage does not depend on this information, so our system does not provide dosage calculation. It is worth mentioning that recalls are difficult for the medical staff who does not practice resuscitation routinely but may be easier for the experienced medical staff. On the other hand, assembling clinical information and real-time tracking significantly increases the cognitive load for both inexperienced and experienced medical staff.

6.2 Design Methodology

In this section, we describe the design methodology we performed over a 2-yr period working in close collaboration with physicians and nurses at Carle Foundation Hospital to create our medical Best Practice Guidance system.

6.2.1 Contextual Design and Design Process Overview

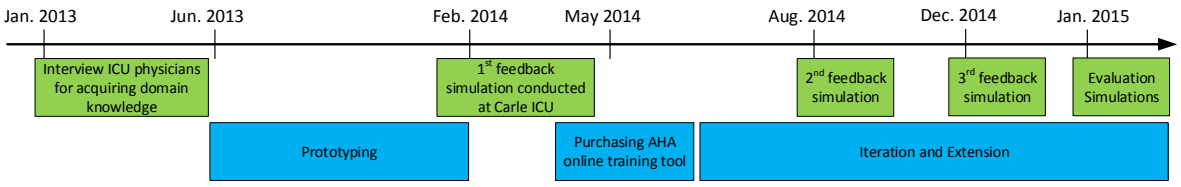


Figure 6.2: Development timeline of the Best Practice Guidance System

Table 6.1: The settings of the interviews and simulations

	Frequency	Participants	Duration
Contextual Inquiry Interviews	$\backsim$ 6	ICU director, consultant physician, 2 additional physicians (A, B), and 2 head nurses (A, B)	30 min $\backsim$ 1 hour/interview
1st Feedback Simulation	1	ICU director, consultant physician, resident A, and head nurse A	1 hour
Interview after 1st Simulation	1	The same participants as the 1st Simulation	30 min
2nd Feedback Simulation	1	ICU director, physician A, resident B, and head nurse B	1 hour
Interview after 2nd Simulation	1	The same participants as the 2nd Simulation	30 min
3rd Feedback Simulation	1	ICU director, physician B, resident C, and head nurse A	1 hour
Interview after 3rd Simulation	1	The same participants as the 3rd Simulation	30 min
Evaluation simulations	3	4 medical staff for each simulation	1 hour
During the whole design process, we had weekly 2-hour meetings with our consultant physician.			

Contextual design [17] is a user-centered design approach for developing systems on the basis of data gathered from the intended users. The design process starts by collecting data about the users in the field (contextual inquiry). The data collected from the contextual inquiry is interpreted in a structured way by using five work models: the flow model, the sequence model, the artifact model, the cultural model, and the physical model. Based on the work models, the system design team then prototypes the system with the users in the design process. The prototype is then iteratively tested and refined with the users.

The design process for our BPG system accords with contextual design. The system design and development timeline is illustrated in Fig. 6.2. The settings for the interviews and simulations engaging users are presented in Table 6.1. During the design process, our system design team, consisting mainly of two postdoctoral researchers and one Ph.D student, had weekly meetings with our consultant physician from Carle Foundation Hospital. We began the contextual inquiry first to understand the intended users' needs and to identify key design requirements. We then analyzed and structured the collected data by using a set of work models according to the contextual design methodology. All the five work models have impacted on the design of the BPG system. Based on the work models, we started to prototype the system. We afterward conducted three clinical feedback simulations with physicians and nurses from Carle Foundation Hospital to gather users' feedback of the prototype, and iteratively improved it. After improving the prototype based on their comments, we conducted three evaluation simulations to quantitatively evaluate the effectiveness. In the following sections, we describe the design process in details.

6.2.2 Interviews and Contextual Inquiry

Starting in January 2013, we conducted a series of interviews with ICU physicians and nurses at Carle Foundation Hospital in order to gather knowledge about their work environment and design requirements. Every month we had a group meeting with 3 to 5 physicians and nurses from ICU of Carle Foundation Hospital. These meetings lasted about 30 minutes to one hour each. The director of Carle ICU and our consultant physician attended every meeting, and they also invited some other physicians and nurses to join the meetings and provide feedback. In five months, we met totally 6 medical staff,

including the director of Carle ICU, our consultant physician, two additional physicians, and two head nurses. Through the interviews, we came to understand how the physicians and nurses treat cardiac arrest patients, the devices they used, and the possible mistakes they may make, etc. The system design team iteratively built a set of work models drawn from the contextual design methodology with our consultant physician's feedback. Each work model informed a different aspect of system design as described in the following sections.

Flow Model

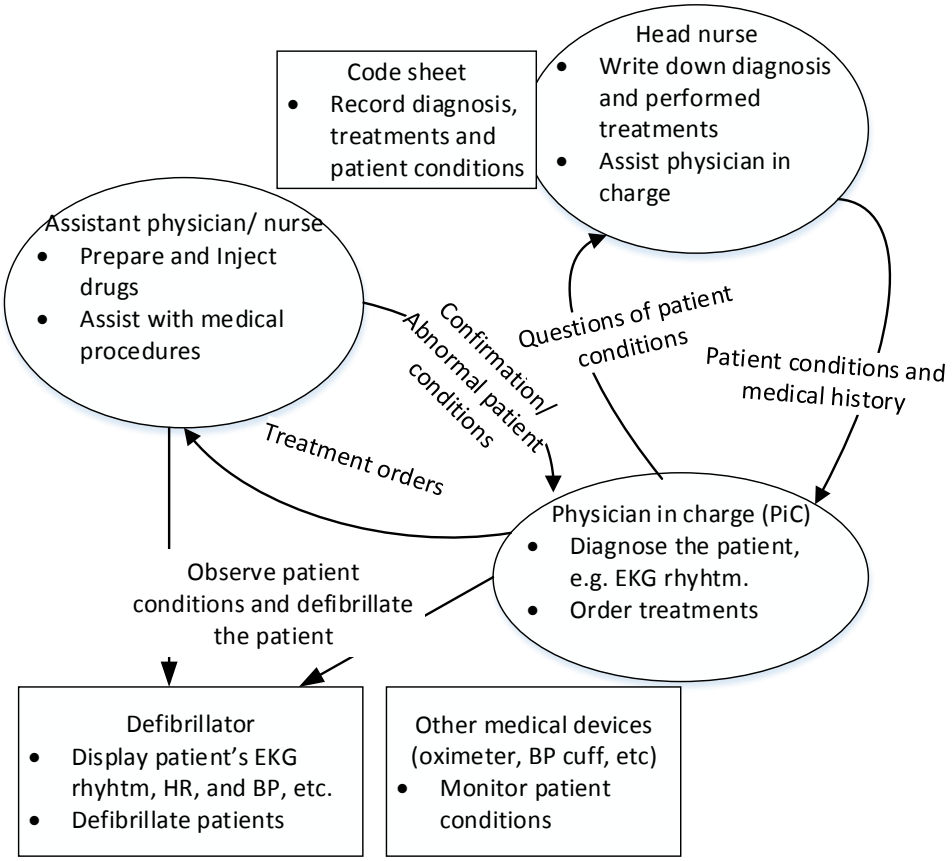


Figure 6.3: Flow model. The ellipses are the medical staff, and the rectangles are the artifacts. The items in them are responsibilities or purposes. The arrows show the communicated information.

The flow model, which is shown in Fig. 6.3, is used to depict the communication and coordination between personnel as well as their responsibilities. A cardiac arrest resuscitation team usually consists

of a physician in charge, a head nurse, and/or one to two assistant physicians/nurses depends on the availability. The **physician in charge (PiC)** is responsible for coordinating the team, diagnosing the patient, and ordering treatments. The **head nurse** is responsible for writing the diagnosis, performed treatments, and patient conditions on a code sheet, so physician in charge can review it and decide further treatment plans. The PiC may raise questions regarding patient's conditions and performed treatments, and the head nurse is the primary person to provide the answers. The team members continually look at the patient's physiological measurements displayed on a defibrillator and then use it to defibrillate the patient. If a team member notices any abnormal patient conditions, s/he needs to reports the abnormality to PiC. In addition, all the medical staff should follow best practice workflows developed by AHA [35] to the greatest extent yet actively decide upon the treatment plans that best meet the current patient conditions. Based on the above findings, the system was designed to externalize the communicated clinical information in order to reduce the possibility of misunderstandings and the medical staffs' mental workload.

Sequence Model

Fig. 6.4 shows the sequence model, which describes how a sequence of tasks unfolds over time. In order to develop the sequence model, during the interviews, we asked the physicians and nurses to describe the treatments and assessments they need to perform when treating a cardiac arrest patient. They used the AHA (American Heart Association) guidelines as the references, because they usually follow the guidelines to the greatest extent and pointed out certain treatments and assessments that they usually performed but are not included in the AHA guidelines. For instance, in addition to check the patient's heart rhythm, they will also check other abnormal patient conditions, such as acidosis (patient's blood pH value lower than 7.2), and provide treatments accordingly.

The steps of treating cardiac arrest patients can be summarized as the following. When a patient does not breathe and has no pulse, physicians or nurses need to call a code 99[2] and request help. They, then, start doing CPR, attaching medical devices, including a defibrillator and oximeter, and perform initial assessments. They usually follow the sequence of **A**irway, **B**reathing, and **C**irculation

[2]Code 99 is an emergent situation where the patient is not breathing and/or has no pulse.

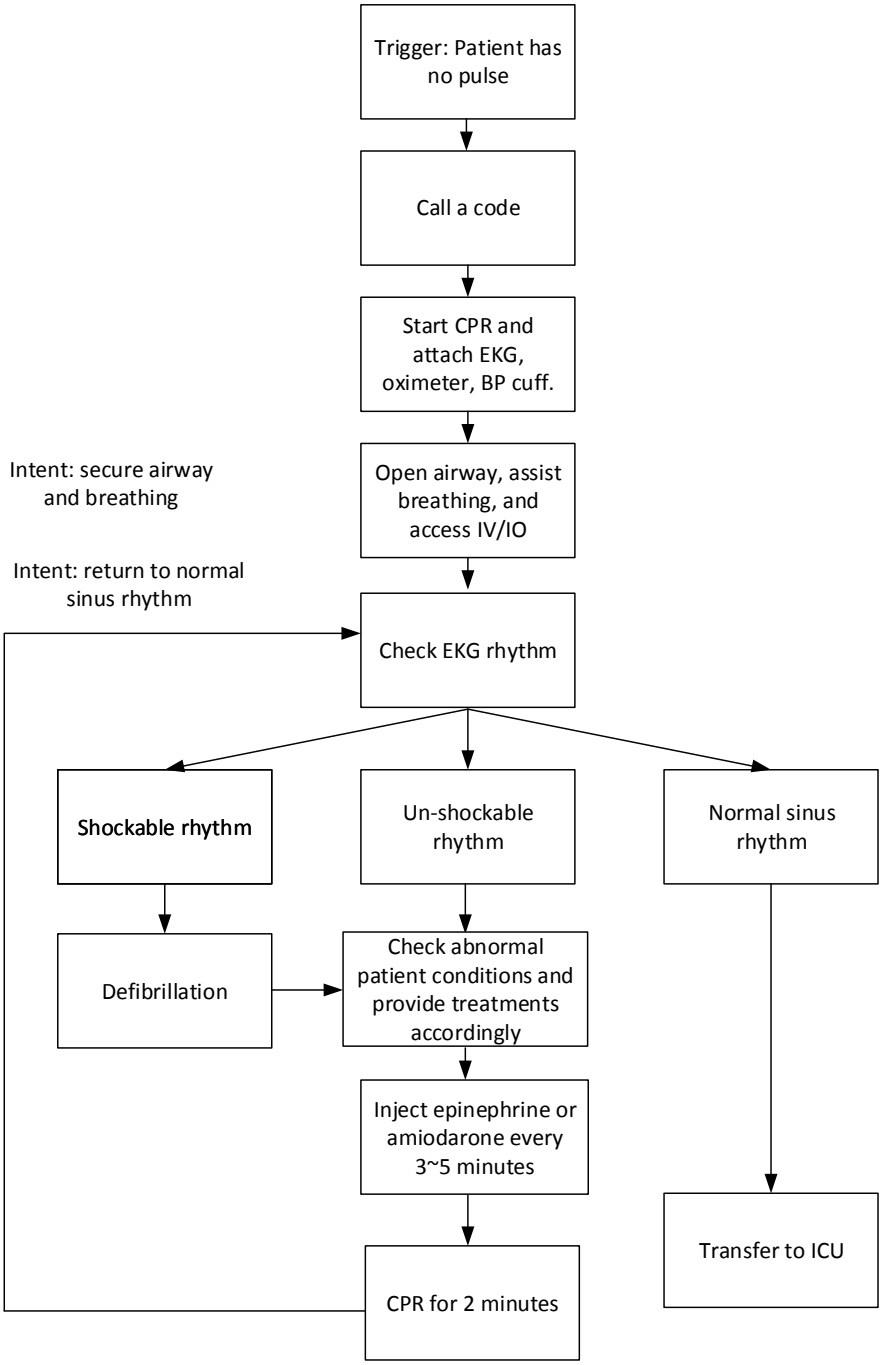


Figure 6.4: Sequence model.

(ABC)[3]. In addition, they perform intravenous (IV) or intraosseous (IO) infusion[4]. When the devices

[3]Some physicians may prefer (CAB), instead.

[4]IV or IO infusion is to infuse fluid or medication directly into the patient's vein or marrow.

are ready, they check the patient's heart rhythm displayed on the defibrillator. The rhythm can be a shockable rhythm, such as ventricular fibrillation, or a non-shockable, such as asystole. If the rhythm is shockable, the physicians and nurses activate a defibrillator to deliver a therapeutic dose of electrical energy to the heart. If the rhythm is non-shockable or the defibrillation has been performed, they usually treat abnormal patient conditions, such as low blood pH, and administer drugs, such as sodium bicarbonate. They, afterward, perform CPR for at least 2 minutes and check rhythm again until the patient's heart rhythm returns to a normal sinus rhythm.

Based on the sequence model, the system was designed to provide guidance on the assessments and treatments in complaint with the AHA guidelines.

Cultural Model

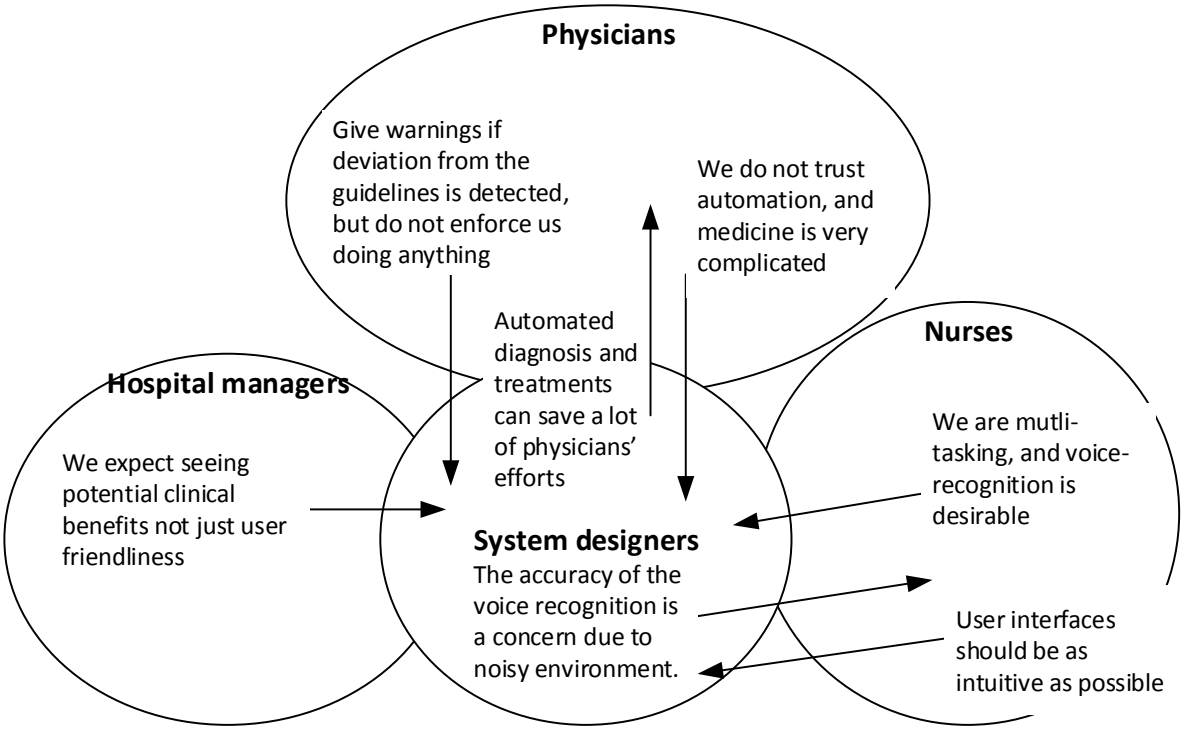


Figure 6.5: Cultural model

In cultural model, as shown in Fig. 6.5, we present how the physicians, nurses, and hospital managers influenced the system design. The physicians have a great impact on the design of our medical

BPG system. At the early stage of development, we initially intended to design a system supporting automated diagnosis and treatment in order to reduce physician effort. In contrast, the physicians worried that automated diagnosis and treatments may potentially compromise patient safety if not designed correctly. In addition, the safety and efficacy of patient care should ultimately depend on physicians' judgments and expertise. The physicians preferred that the system alert their attention to deviations from medical best practices but not to enforce their behavior. Additionally, because nurses are responsible for communicating much medical information, such as diagnoses, treatment orders, and lab test results, their opinions have also had a large influence on our system design as well. At the start of our work, the nurses expected that the system would have voice recognition capabilities. However, the noisy environment and simultaneous speaking between physicians and nurses pose significant challenges to voice recognition. After a series of discussions, the system designers and nurses agreed to have a tablet for the nurse to input medical information. The tablet is not as desirable as voice recognition, but accuracy is much higher. In addition, the user interface of the tablet is designed to be intuitive for the nurses from a look and feel perspective, and with respect to the terminology they already use. Moreover, hospital management expects to see potential improvements in clinical outcomes assuming our system is deployed. In summary, based on the cultural model, the system was designed to meet medical staffs' expectations.

Artifact Model

The artifact model was used to analyze the artifacts collected from the physicians and nurses. In particular, we focused on the medical devices and code sheet that are used during cardiac arrest resuscitation. Fig. 6.6 shows the defibrillator used to monitor patient conditions and defibrillate patients. Critical physiological measurements, including heart rate, body temperature and EKG rhythm, are displayed on an 8.4-inch diagonal LCD. Physicians and nurses can use the green knob to control the defibrillation energy. The code sheet, which is shown in Fig. 6.7, is used for documenting diagnosis and performed treatments. Physicians and nurses start from assessing **A**irway, **B**reathing, and **C**irculation. During the resuscitation, the head nurse keeps a written record of the patient's physiological measurements, and the performed treatments, including defibrillation and drugs. In addition, physician in charge may order blood test; when the test results are back, head nurse should document the results, including blood pH

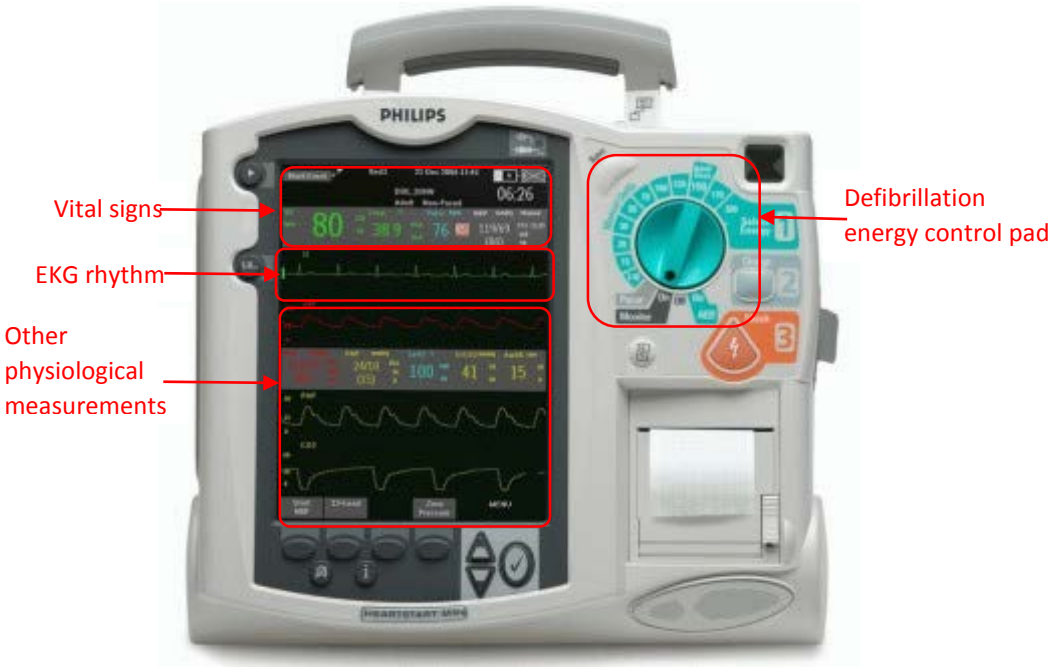


Figure 6.6: Defibrillator

value and potassium level. By studying these artifacts, the system designers were able to understand the most critical information to be displayed and recorded electronically.

Physical Model

In Fig. 6.8, we present the physical model, which illustrates the physical environment. When it is noticed that a patient has no heart rate and is not breathing, physicians and nurses rush to the patient's room and carry a cardiac arrest cart, which is equipped with a defibrillator, an oximeter, a blood pressure cuff, and the drugs commonly used for cardiac arrest resuscitation. In our original design, we used a projector to present the integrated display on the wall of the room, because we believed that a projector could be readily accommodated on the cardiac arrest cart, and the size of the display could be adjusted based on the physical environment and users' needs. In one simulated scenario however, we noticed that physicians and nurses needed to walk around the patient frequently and occasionally blocked the projected display. In addition, we found that most patient rooms are equipped with a 50-inch high resolution televisions. Consequently, we decided to use the television as the primary display

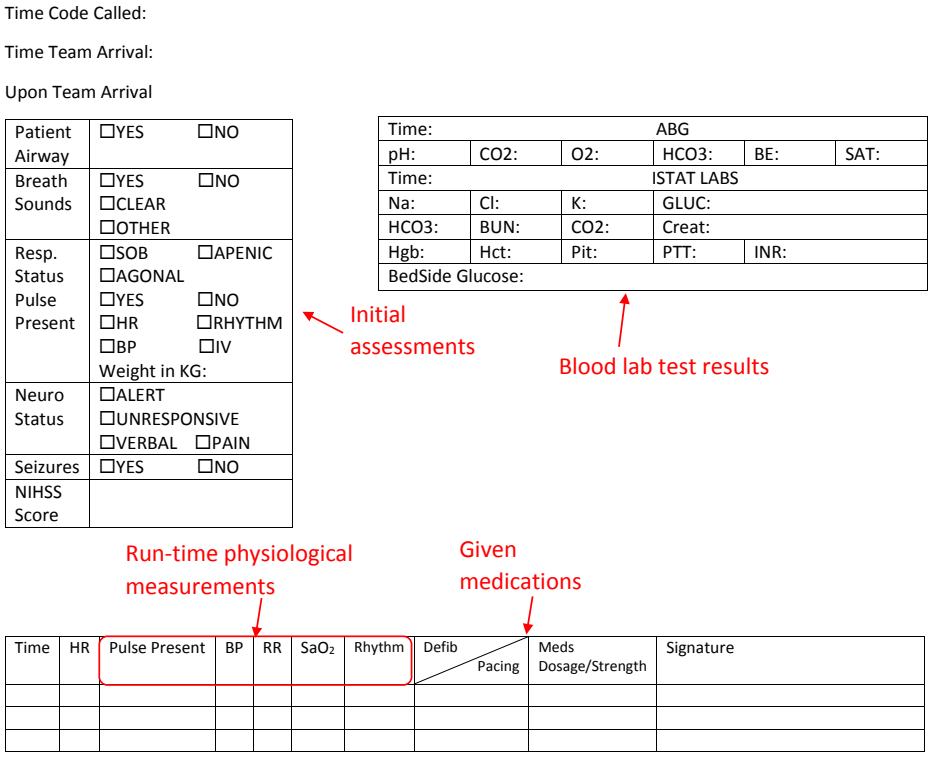


Figure 6.7: Code sheet

device. Based on the physical model, the system designers understood the environmental constraints and opportunities that need to be considered for a successful design.

Design Requirements

Based on the information we have documented here in contextual design work models, we specified a list of design requirements in collaboration with physicians and nurses. The most important design requirements were those specifically related to the goal of reducing cognitive load. Each of the requirements below is directed toward reducing the cognitive load and supporting the medical team in better adherent to the best practice guidelines.

Reducing cognitive load due to **M1. assemble clinical information**:

R1 **Integrate clinical information to support both individual and team situation awareness:** The system should integrate and externalize clinical information, including physiological information, workflow, and treatment history, in order to facilitate the communication between

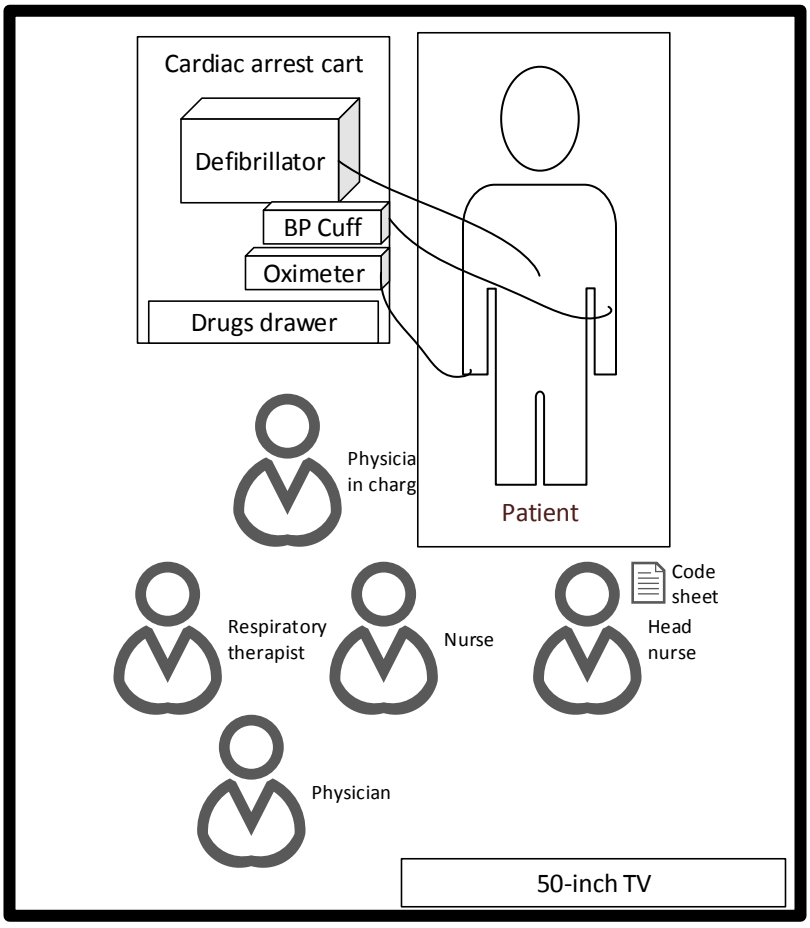


Figure 6.8: Physical model

medical staff, as described in the flow model (Fig. 6.3).

Reducing cognitive load due to **M2. recalls**:

R2 **Display workflows according to physician's diagnosis and patient conditions:** The medical staff should recall workflows and follow them to the greatest extent. Consequently, according to the physician's diagnosis and currently patient conditions, the system should display the corresponding workflow and provide step-by-step guidance. In addition, a warning should be raised if a potential deviation from a workflow is detected.

R3 **Support the medical team in validating treatment preconditions:** As described in the previous section, the system should continuously monitor patient conditions and validate the precon-

ditions of the treatments. The status of the treatments should be displayed to let physicians and nurses be aware of the applicable treatments.

R4 **Support the medical team in recalling pending medication orders:** The system should display the medication that has been ordered but not yet administered to the patient. Consequently, the chance of missing or misunderstanding medication orders can be reduced.

R5 **Provide diagnosis and treatment log:** The history of diagnoses and performed treatments is valuable information for physicians in deciding upon future treatment plans.

Reducing cognitive load due to **M3. real-time tracking**:

R6 **Support the medical team temporal awareness of treatments in progress:** As described in the sequence model (Fig. 6.4), certain treatments are time sensitive, so the system should keep track of the time progress of the treatments and reminds physicians if any treatment is performed too early or should be performed.

R7 **Support the medical team in attending to abnormal physiological measurements:** Any abnormal physiological measurements should be brought to the attention of physicians so the medical team can respond to them as appropriate. Moreover, the system should allow physicians to configure the thresholds of the physiological measurements defining what is meant by abnormal, as these thresholds are context-dependent, i.e. depending on the patients' conditions including age, sex, and medical history.

6.2.3 Prototyping

We started to prototype our system in June 2013. The system architecture and the initial user interface prototypes are shown in Fig. 6.9. The system separates the concern of user interfaces, which are an *integrated patient condition and workflow display* (details in Section 6.3.1) and a *nurse's tablet* (details in Section 6.3.2), and decision logic, which is a *best practice workflow manager* (details in Section 6.3.3). The *integrated display* and *workflow manager* are setup on a laptop and communicate with the nurse's tablet through Wi-Fi. At Carle Foundation Hospital, most of the patient rooms are equipped with a

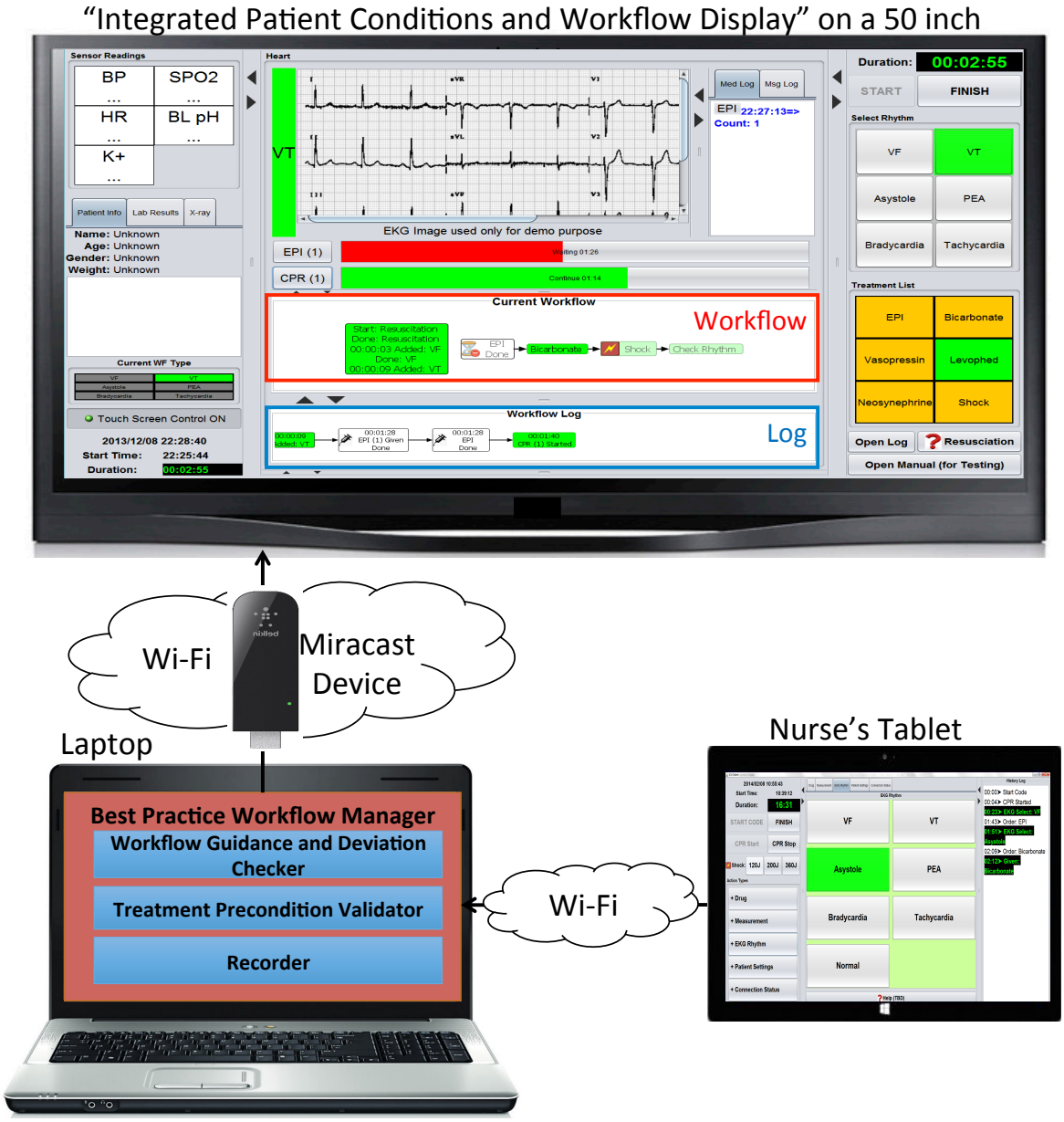


Figure 6.9: System Architecture and User Interface Prototypes

50-inch television, we use a Belkin Miracast, which is a wireless display device for mirroring a computer screen to a television without requiring any physical cables, to show the *integrated display* on the television.

The integrated display is designed to provide concise, comprehensive, and integrated workflow and clinical information for all the team members to maintain overall awareness of the patient conditions

and treatment procedures. The head nurse has a unique task to record the diagnosis and treatments, and s/he also needs to answer the questions raised from physicians. The *nurse's tablet* is for the head nurse to input and record diagnosis and treatment orders. In addition, the head nurse can also review the previous diagnosis and treatments. The *workflow manager* checks the status of physiological measurements, validates treatment preconditions, provides guidance, and raises warnings on the *integrated display* to help medical staff follow the guidelines. At the current development stage, the workflow manager uses Medical Device Plug and Play (MDPnP) middleware [37] to interface with the distributed medical devices and receive physiological measurements, such as EKG, heart rate, and blood pressure, from them. In the next section, we will discuss how the system is involved through simulations and interviews.

6.2.4 First Feedback Simulation at Carle ICU

Figure 6.10: Simulation environment

After prototyping the first version of our medical BPG system, we needed physicians and nurses' feedback on using the system. Nevertheless, Carle Foundation Hospital has strict regulations that do

not allow non-medical personnel to participate in or even observe real clinical practices. In addition, since the developed system has not yet been approved by the FDA (Food and Drug Administration), we cannot deploy the system for treating actual patients. We therefore conducted a scenario-based simulation, which was videotaped, with actual physicians and nurses from Carle ICU. The simulation was performed in a safe environment where no actual patients were involved and where we could observe the interactions between medical staff and the system. The simulation environment is shown in Fig. 6.10. In addition to members of our system design team, four medical staff members were involved in the simulation: the director of ICU, our consultant physician, Resident A, and Head nurse A, as shown in Table 6.1. We started by giving a 10-minute introduction to explain the displayed clinical information, the system behavior, and the interactions between the system components. Unlike many engineered systems, medicine has very few "equations" to describe or predict patients' physiological changes to diseases and treatments [32]. Therefore, instead of developing an automated scripted scenario, aspects of the scenario were under the control of the director of ICU with expert knowledge and experience. The director of ICU decided how the patient's physiological measurements changed over time, in a manner consistent with previous cases and representative alternatives. The consultant physician, resident A, and head nurse A interacted with the prototype system in deciding upon treatment plans.

After the simulation, we conducted an interview with all the participants together. The interview took about thirty minutes. We asked their opinions about each individual displayed component, such as workflow, vital signs panel, and treatment progress bars. In addition, we also asked a list of questions, including,

1. Can the generated warnings help them be better adherent to the AHA guidelines?

2. What are the difficulties they faced when using our system?

3. What are the additional features they expected the system to have?

4. Do they think the displayed information is user-friendly?

In summary, the physicians and nurses thought the displayed information was helpful and improved their situation awareness. They believed that system-generated warnings were useful for reducing

potential medical errors. However, they faced some difficulties to gather the clinical information as some interaction was clumsy and expected the system being more user-friendly. First, the previous medical events, i.e. diagnosis and treatments, in the log were organized as a sequence of boxes, as shown in the blue box in Fig. 6.9. The physicians, however, thought that it would be easier for them to retrieve the information by overlaying the medical events on a timeline. Second, a physician pointed out that

> *I cannot relate the displayed workflow to the AHA guidelines [35], because the flowchart in the guidelines is shown in a vertical way but your workflow is shown in a horizontal way (as shown in the red box in Fig. 6.9). At a glance, I did not know what it is and just ignored it.*

We addressed those issues and improved the user interface design accordingly. In addition, on the basis of information obtained via simulation, we identified a set of additional design requirements for improving usability.

R8 **Automatic code sheet generation:** As described in the previous section, the head nurse needs to manually fill information into the code sheet (Fig. 6.7), which considerably increases his/her workload. Therefore, the system should automatically generate the code sheet based on the recorded diagnosis and treatments.

R9 **Default drug dosage:** Since many drugs have standard dosage in the AHA guidelines, the system should use the guideline dosage as a default and allow the head nurse to change it if so desired. In this way, head nurse does not need to input drug dosage every time a drug is ordered.

6.2.5 Purchasing an American Heart Association (AHA) online training tool

Given the many limitations of our initial simulation-based approach to system evaluation, and also to privacy regulations, we are unable to directly observe how physicians and nurses treat cardiac arrest patients. In order to enhance the fidelity of future simulation-based evaluations, we purchased an online training tool[5] which was developed by the AHA to train medical staff to assess patients, formulate a

[5] https://www.onlineaha.org/system/scidea/courses/15/more_info/90-1405_HC_ACLS.pdf

treatment plan based on the guidelines, and provide treatments. Through interacting with this training tool, we gained an improved understanding of the commonly used drugs, the preconditions of the drugs, and how the workflows are executed. Afterwards, the system was improved by including more comprehensive drug and drug administration preconditions lists. In addition, the best practice workflows were also revised in a manner consistent with those used in the training tool.

6.2.6 2nd and 3rd Feedback Simulation

In August and December 2014, we conducted two additional simulations to gather more feedback from physicians and nurses. The settings of the simulations are similar to the 1st one, and the participants and duration are described in Table 6.1. It is worth pointing out that in the 2nd and 3rd simulation we invited different groups of physicians and residents to use our system, because they may provide feedback from a different perspective. Unfortunately, due to the scheduling issue, in the 3rd simulation we were unable to invite another head nurse to join the simulation. After the simulations, we conducted interviews with all the participants together. We present the detailed analysis and improvement in the next *Iteration and Extension section*.

During the interviews, in addition to give feedback about the system design, the medical staff also provided some clinical cases they faced and sought for IT supports. In particular, in the 2nd simulation, a physician mentioned that

> *Yesterday, we treated a patient with bradycardia, which is the heart rate abnormally slower than normal heart rate. The patient's heart rate kept slowing down, and we knew it would turn to asystole (flatline) in a short time. We must do something to prevent this. Moreover, we also need to know all the performed treatments, so we can know whether the patient responses to the treatments or not. If not, we should consider alternative treatments*

In the 3rd simulation, one physician pointed out that in real clinical practices, blood test results are critical for physicians to decide treatment plans. A nurse needs to draw patient's blood and send the blood to the lab. If the blood test results do not come back, it may be because the nurse forgets to send the blood to the lab or for some other reasons. The system should alert the medical staff of this issue, so they can adjust the treatment plans accordingly.

6.2.7 Iteration and Extension

The participants of 2nd and 3rd simulations were satisfied with displayed information and agreed that the system can improve their situation awareness. The head nurses also thought that the design of the nurse's tablet is very intuitive and easy to use. In addition, the system team improved the system based on the clinical cases mentioned by the physicians during the interviews. Those clinical cases motivated two major design requirements.

R10 **Support the medical team in tracking the trends of physiological measurements:** The system should take the trend of physiological measurements into consideration and inform the medical team of applicable treatments for treating the abnormalities.

R11 **Display brief summaries of the previous diagnosis and performed treatments:** The system should provide brief summaries of the performed treatments, so at a glance, physicians can know, for instance, how many doses of epinephrine have been given to the patient and decide treatment plans accordingly.

In the following sections, we will present the final design of the developed medical Best Practice Guidance system and the evaluation results.

6.3 Medical Best Practice Gudiance System Design

Over two years, we have conducted more than 10 demonstrations and interviews to about 10 physicians and nurses from Carle Foundation Hospital. We iteratively improved our system based on their feedback, including adding new features, improving usability, and customizing the user interface. Due to space limit, we are not able to present their feedback and improvements for every iteration. In this section, we only provide detailed information on the final design of medical Best Practice Guidance system.

6.3.1 Integrated Patient Conditions and Workflow Display

The integrated display, shown in Fig. 6.11, provides concise and comprehensive clinical information to fulfill design requirement *R1*. The integrated display consists of the following main components:

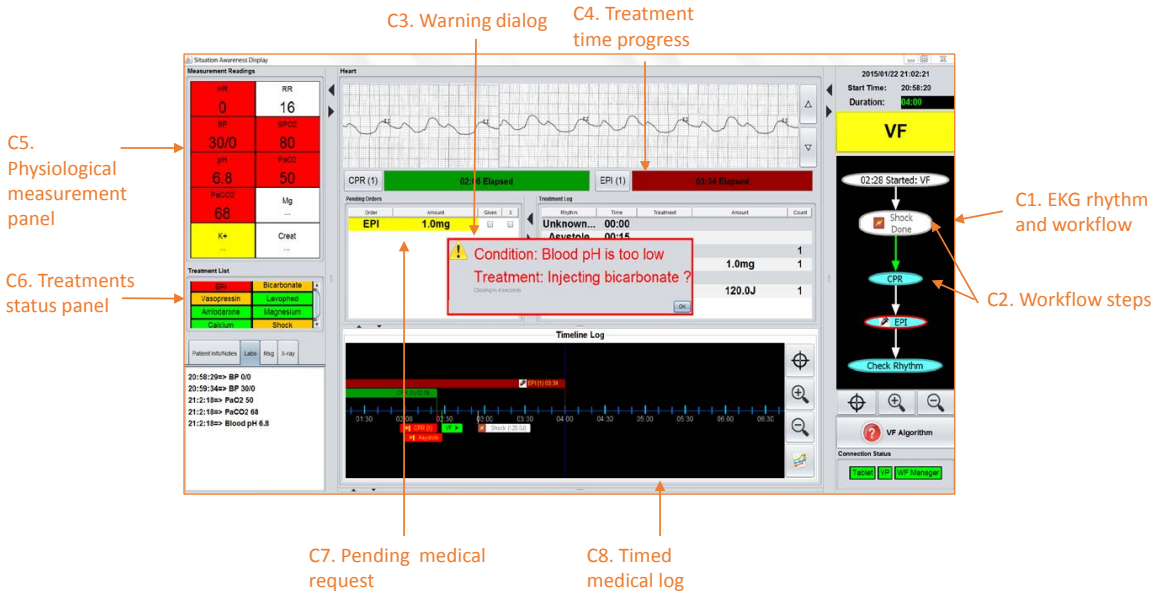


Figure 6.11: Integrated Patient Conditions and Workflow Display

C1. Workflow for treating abnormal heart rhythm: The system displays the workflow in the context of cardiac arrest resuscitation based on patient conditions and physician's diagnosis inputted by the nurse tablet.

C2. Workflow steps: The status of a workflow step is categorized into done, in progress, and should be performed. Color-coding is used to help discriminate between these categories. In the case shown, the patient has been defibrillated, so the Shock state is shown in white (done). The EPI (epinephrine) state has an outline in red. Red is used to draw physicians and nurses' attention to the fact that it is time to give EPI (should be performed). The CPR state is in blue, which indicates that CPR has not yet been performed. Medical teams are supported in determining the most appropriate next step to take by displaying this workflow information in a perceptually salient manner, in view of all team members.

C3. Warning dialog: If any potential deviation from the workflow is detected, for example, a treatment should be performed but physicians have not yet ordered it, or any patient's physiological measurement becomes abnormal, a warning dialog is shown to alert the medical team. In addition, if best practice guidelines suggest certain treatments for correcting the abnormal physiological measurement, the suggested treatments are also displayed. In the case shown, the team is being reminded that this patient's blood pH level is too low and sodium bicarbonate is recommended. In order to prevent the warning

dialog from blocking other critical information, the dialog will only appear for a short time, i.e. 10 sec, and be recorded. Physicians and nurses can retrieve the warning information from the log.

C4. Treatment time progress bars: Time progress bars are used to help the medical team meet the timing requirements of each action. At a glance, all the team members can easily keep track of the temporal progress of the treatments. Therefore, requirement *R6* is satisfied.

C5. Physiological measurements panel: The system continuously monitors the patient's physiological measurements, e.g. heart rate and blood pressure, and the status of the measurements are color-coded, which fulfills requirement *R7*. White means normal; red means seriously abnormal; yellow means caution.

C6. Treatment status panel: The panel shows the status of commonly used treatments, including epinephrine and sodium bicarbonate. The status of treatments is color-coded. Green means all the preconditions are satisfied. Yellow means some physiological measurements for validating the preconditions are missing; therefore, the medical staff should perform the treatment with caution. Red means some preconditions are not satisfied. Based on this treatment status panel, the medical staff can recognize the applicable treatments. Consequently, requirement *R3* is satisfied.

C7. Pending medical requests table: When a physician orders a treatment, the ordered treatment and dosage will be shown in the table until it is performed, which fulfills requirement *R4*. If a treatment order is pending for longer than a nominal time interval, a warning dialog is shown to remind the medical team of this fact.

C8. Timed medical log: The timed medical log records all the diagnoses and treatments that have been performed and displayed in a time sequence. Together with the recorded patient's physiological measurements, it allows physicians to assess the effectiveness of the current treatment approach. Consequently, requirement *R5* is satisfied.

6.3.2 Nurse's Touch Tablet

The nurse's tablet, as shown in Fig. 6.12, is designed to allow the head nurse to input medical information and quickly gather information regarding the established diagnosis and performed treatments. The nurse tablet consists of the following major components:

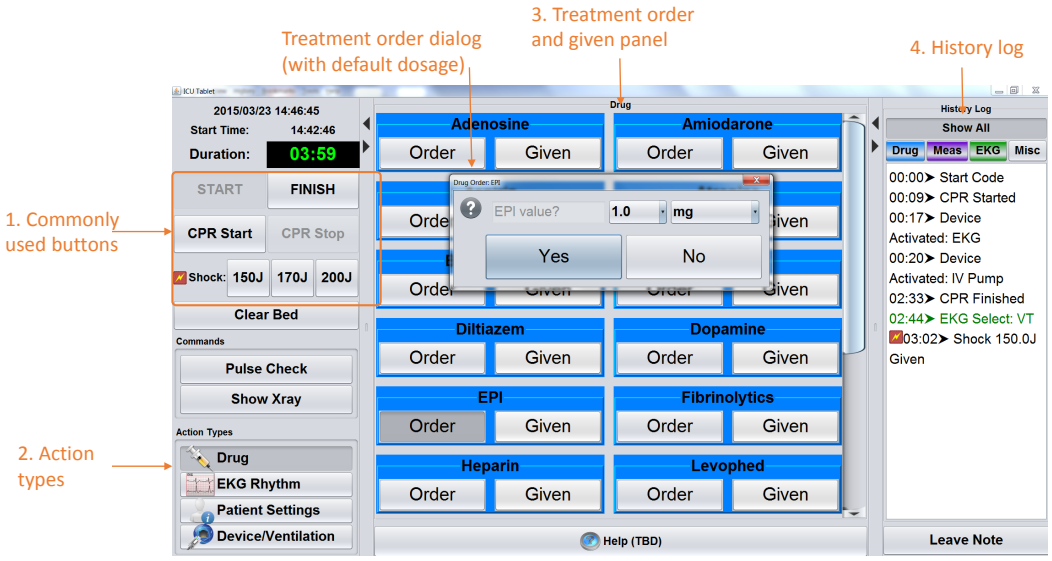


Figure 6.12: Nurse's Tablet

1. Commonly used buttons: There are some procedures or treatments, e.g. CPR and defibrillation, frequently performed during cardiac arrest resuscitation. In order to let the head nurse quickly input that information, buttons pertaining to these procedures are grouped together.

2. Action types: The functionality of the nurse's tablet is categorized into four action types: drug, EKG rhythm, patient settings and device/ventilation settings. When the head nurse selects any of the group, a corresponding panel is displayed in the middle. The drug panel is used to input drug orders and dosage. The EKG rhythm panel is used to input the rhythm diagnosis. Patient settings panel is used to input patent's personal information and medical history. Device/ventilation settings are used to configure the thresholds of the physiological measurements, which satisfies requirement *R7*. We use the drug panel as an example.

3. Treatment order and given panel: This panel is used for entering drug orders and given information. Our system takes two steps, i.e. "order" and "given", to administer a drug. First, a physician in charge gives a verbal drug order. Second, a nurse takes the drug from the drug drawer, checks the drug name and dosage, and injects it into the patient. Therefore, there are two buttons for each drug. When the head nurse pushes the "Order" button of a drug, an order dialog is shown with guideline dosage as the default one, which fulfills requirement *R9*. When a drug is given to a patient, the head nurse needs

to pushes the "Given" button and the system will record the given time and dosage.

4. History log: This log is for the head nurse to have a clear understanding of the established diagnosis and previously performed treatments, which fulfills requirement *R11*.

6.3.3 Best Practice Workflow Manager

Our best workflow manager explicitly codifies medical workflows and treatments in order to provide guidance for the medical staff. Specifically, the best practice workflow manager has three major functionalities. First, it selects and presents the recommended workflow according to the medical information, e.g. diagnosis and treatments, entered on the nurse's tablet. Medical workflows are codified from AHA guidelines using timed automaton. The workflow manager reminds the medical staff of the assessments and treatments should be performed according to the workflow states and transitions. In addition, if any potential deviation from the guidelines is detected, for example, a treatment should be performed but physicians have not yet ordered it, the workflow manager will create a warning on the integrated display. Consequently, together with the *C1* and *C2* of the integrated display, the medical staff can easily comprehend and adhere to the workflows, which fulfills the requirement *R2*. It is worth mentioning that the codified workflow has been validated by the physicians through clinical simulations. The details of the simulations will be discussed in Section 6.5.

Second, in order to satisfy requirement *R3*, the workflow manager continuously monitor the patient conditions against the treatment preconditions. The status of the treatments is color-coded and displayed on the *C6* of the integrated display. In addition, if the workflow manager receives a treatment order from the nurse's tablet and the treatment preconditions are not satisfied, a warning will be raised to alert the medical team to this precondition violation. In our current system design, the treatment information are specified by the physicians before the procedure starts, but the information only need to be inputted once and will be stored in the system for further reference. We are currently working to interface our system with a pharmaceutical database to automatically retrieve treatment information.

Third, the workflow manager maintains a log of the established diagnosis and all the performed treatments. The logged information is displayed on *C8* of the integrated display. Physicians and nurses can review this information to help them decide upon future treatment plans. In addition, at the end of

cardiac arrest resuscitation, the workflow manager automatically generates the code sheet based on the recorded information, which fulfills requirement *R8*.

Furthermore, the workflow manager also keeps track of the patient's physiological measurements. If the any measurement is out of the normal range specified by the physicians, it will change the corresponding measurement color shown on *C5* of the integrated display. In addition, more incipient changes of patient conditions may be sooner recognized by identifying trends of measurements. Therefore, the workflow manager monitors the changes of the measurements over time and alerts the medical staff when any measurement is moving toward abnormal. For instance, when a patient's heart rate is slowing, a warning dialog is displayed with a list of drugs which are commonly used to increase a patient's heart rate. Consequently, design requirement *R7* and *R11* are satisfied.

In this work, we focus on the human-centric design and reducing medical staff's cognitive load, more details of the protocols utilized in workflow manager can be found in our previous work [93, 95, 96].

6.4 Design Rationale and Potential Risks

In this section, we discuss the design rationale behind our medical BPG system and the potential limitations.

Our system selected the AHA (American Heart Association) guidelines as the basis to provide guidance, although AHA guidelines may be insufficiently detailed and incomplete [74]. The AHA guidelines, nevertheless, provide a set of essential and well-accepted primitives for treating cardiac arrest patients. Clinical outcomes can be generally improved if medical staff adheres to the AHA guidelines [91]. Recognition of this fact is the main reason that Carle Foundation Hospital intends to use the developed system to improve adherence. However, patient conditions may change rapidly and develop complications, such as brain damage, so it is very difficult to develop a guideline to take all the possible patient conditions and complications into consideration [68]. Therefore, in clinical practices, the medical staff should follow those essential and well-accepted guideline primitives to the greatest extent yet actively decide upon the treatment plans that best meet the current patient conditions. Our system, consequently, externalizes the AHA guidelines and essential clinical information in a

sensible and organized manner without precluding the medical staff to also take into account additional information in deciding upon actual care. As one Carle physician explained:

> *The basis for AHA, however, is to emphasize those primary principles, so that, at least, they will be followed; this fact allows the physician/nurse/team to then address other patient conditions. The real-time information provided by the system is very useful in that sense, because it significantly reduces our burden from keeping track of the time progress and the performed treatments and lets us more focus on the patient conditions.*

We now discuss the limitations of our system. First, the patient conditions monitored by our system are incomplete due to the limitations of available medical devices. For example, the quality of CPR is an important factor, which can be measured by an exhaled carbon dioxide monitor. [6]. Unfortunately, most of the patient rooms at Carle Foundation Hospital are not equipped with this medical device, and thus our system cannot display exhaled carbon dioxide information. Our system relies on the medical staff to manually monitor the patient conditions, especially the conditions that cannot be monitored by the system. Second, our system can only issue warnings when the deviation from best practice workflows is detected but not stop such deviation. Therefore, deviations may still occur if the medical staff neglects the warnings. The above two types of limitations are disclosed to the medical staff, and we are working with them to design training activities in order to address the limitations. Third, if the medical staff over-relies on the system, in the occurrence of the system failure the medical staff could potentially become confused. Therefore, our system design takes fault-tolerance into consideration and provides redundant hardware and a logging and recovery mechanism, so the system can be resumed from the point of failure.

In this work, we use cardiac arrest resuscitation as our case study. However, we argue that the presented analysis of cognitive load and the developed technology can be applied to other medical scenarios, such as stroke and sepsis. Take stoke management as an example. Medical staff also suffers from high cognitive load due to recall, clinical information assembling and real-time tracking. Particularly, in addition to recall the workflow steps, the medical staff also needs to validate the treatment preconditions and keep track of the temporal progress of the treatment, for instance, initial assessments and

[6]The exhaled carbon dioxide indicates if CPR is effective to improve blood flood to excrete CO_2 from the patient.

treatment, including preforming CT scan, maintaining airway, and managing blood pressure, should be performed within 60 minutes of patient arrival [53]. We have worked closely with Carle Foundation Hospital and adapted the developed medical Best Practice Guidance system for stroke management.

6.5 Cognitive Load Evaluation

6.5.1 Quantitative Evaluation

The remaining challenge is to quantitatively evaluate and compare the medical staff's cognitive load with and without the developed medical BPG system. We therefore conducted three scenario-based simulations with total 12 actual physicians and nurses from Carle Foundation Hospital on a medical SimMan 3G Manikin, which is used for training purpose.

In each simulation, a medical team consists of four members of medical staff: a physician in charge, a head nurse, and two assistant physician/nurse. The physician in charge will be responsible for making diagnoses and ordering treatments. The head nurse recorded the physiological measurements and performed treatments. The assistant physicians/nurses performed the treatments on a medical manikin and remind the physician in charge of any abnormal physiological measurements. The coordinator of the Carle life support program was in charge of adjusting the manikin's responses in a manner consistent with treatments performed by the medical team. The medical team performed cardiac arrest resuscitation twice in each simulation: the first time without the developed system, and the second time with it. After the medical team finished the two scenarios, they evaluated their workload by using NASA Task Load Index (NASA-TLX) questionnaire.

NASA-TLX [42] is a widely-used and subjective assessment tool that rates perceived workload in order to evaluate users' effectiveness and performance. It has been used in various domains, including aviation, healthcare and other complex socio-technical domains. There are six metrics: mental demand, physical demand, temporal demand, performance, effort, and frustration. The descriptions of each metric are summarized in Table 6.2. The physicians and nurses gave a score from 1 to 20 for each metric, and the lower score means the lower demand, better performance, and less effort and frustration.

In Fig. 6.13, we showed the average score for each of the six metrics from 12 physicians and nurses. Our system significantly reduces medical staffs' mental demand and temporal demand, because the

Metrics	Description
Mental Demand	How much mental or perceptual activity was required?
Physical Demand	How much physical activity was required?
Temporal Demand	How hurried or rushed was the pace of the task?
Performance	How successful were you in performing the task? (Lower score means better performance)
Effort	How hard did you have to work to accomplish the task?
Frustration level	How insecure, discouraged, irritated, and stressed, did you feel during the task?

Table 6.2: Descriptions of cognitive load metrics

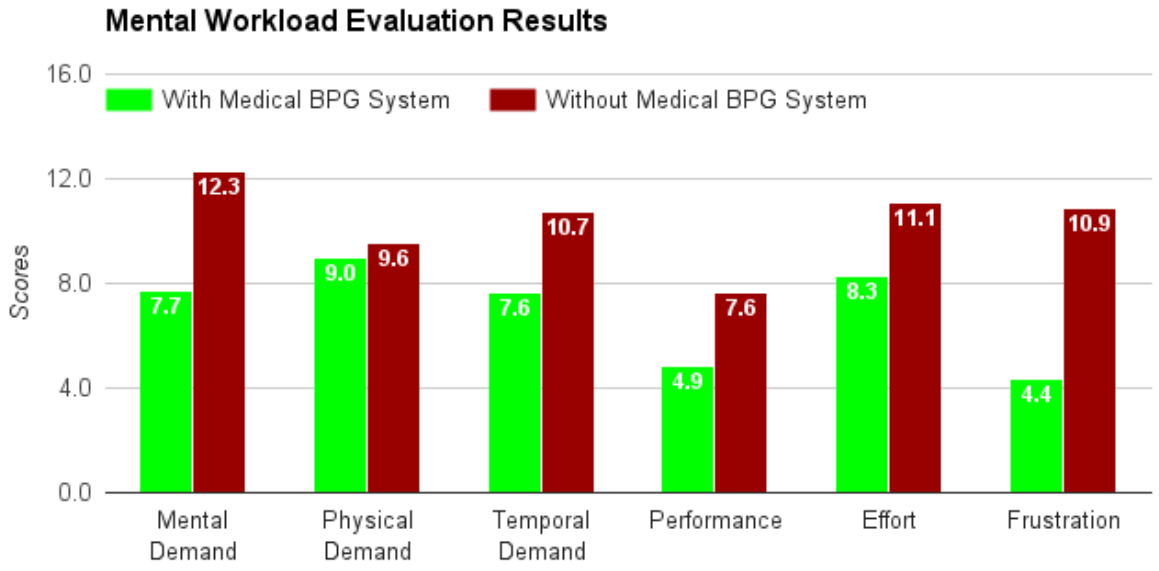


Figure 6.13: Cognitive Load Evaluation Results from 12 physicians and nurses.

critical clinical information and the temporal progress of the treatments are presented on the integrated display. Therefore, the medical staff has less clinical information to recall and can easily track the progress of treatments and patient conditions in real-time. Moreover, with the developed system, the medical staff believed their performance is significantly improved. The performance improvement will be further discussed in the next section. In addition, the effort and frustration levels are also low, which

suggests that the displayed information is critical and easy for medical staff to comprehend. However, our system does not significantly reduce their physical demand as we expected, because they still need to physically perform treatments, i.e. CPR, administering epinephrine. In summary, the evaluation showed that the developed medical BPG system can significantly reduce medical staff's cognitive load and improve their performance.

6.5.2 Qualitative evaluation

In addition to reduce medical staff's cognitive load, the medical staff also believed that our medical BPG system can improve their performance in terms of adherence, situation awareness, and record/review. We have conducted a series of interviews to gather their opinions.

Adherence to best practice guidelines

The medical staff agreed that the developed system could improve the adherence to best practice guidelines. One of the physicians mentioned:

> *The displayed workflow and the guidance of the assessments and treatments that should be performed could be very useful for residents. For experienced physicians and nurses, we tend to remember those things, but it is still a good idea for us to take a look just in case we forget.*

In addition, validating treatment preconditions is very useful for the medical staff. They commented that

> *The warning of blood pH to low when epinephrine is ordered is very helpful, because we tend to forget it, especially in a rush. Other warnings, like not to shock a patient with unshockable rhythm, may also be helpful for residents.*

In summary, our system improves adherence through providing step-by-step guidance of workflows and validating the preconditions of treatments.

Team Situation Awareness

Another important aspect is to improve team situation awareness. First, our system improves temporal awareness through the use of treatment time progress bars. Because these progress bars are color-coded, medical teams can easily keep track of the temporal progress of the treatments. One physician commented:

> *The time progress bars are very helpful. I can know how long I should keep performing CPR or whether to give another dose of epinephrine or not. In our working environment, it is hard to always accurately keep track of the time.*

Second, the table for pending medical request was originally designed to reduce the chance of misunderstanding or neglecting of treatment orders. The coordinator of the Carle life support program added that

> *That table helps us think ahead and be prepared for it. Physician in charge can request certain drugs, such as epinephrine, even the injection interval has not yet reached. Nurse can prepare the drug in advance. When it is time to give the drug, nurse can inject it with short time delay.*

Third, our system shows the candidate treatments and the status of the preconditions check, which can help the medical team be aware of the applicable treatments. One head nurse believed that the treatments status panel is very useful, especially for inexperience residents. She said that:

> *Personally, I like that (treatments status panel) feature. I involved in some situations, and the second year residents just got lost. With that display, they can start to think what the treatments should be considered are.*

In short, the medical staff believe that they can better comprehend the situation and treat patients.

Records and review

One important feature appreciated by the medical staff is to record performed diagnosis and treatments and automatically generate a code sheet. As a nurse commented on this

Keeping an accurate code sheet is hard, because so many things happen at the same time. A physician just starts doing CPR, while another nurse gives epinephrine to the patient. It is not easy to always keep track of the treatments and time correctly. Your (nurse's) tablet can save me a lot of effort.

A physician added that

The records are also important for reviewing and improving purposes, especially for residents. Based on the records, I or other senior doctors can review what residents have done and point out what they may consider doing differently next time.

In summary, physicians and nurses believed this feature can reduce their effort, improve the accuracy of the code sheet, and potentially improve the clinical outcome in the future.

6.5.3 Limitations

In this section, we discus the limitations of the evaluations. First, the conducted evaluations are a pilot study for collecting safety and effectiveness data. Therefore, in spite of the promising results collected from a relatively small group of participants, the evaluation results should not be interpreted with statistical significance. Second, due to the nature of the training classes at Carle Foundation Hospital, the evaluations were not conducted with strict experimental and control groups. Consequently, the cardiac arrest scenarios that the two groups experienced were different and leaded to different cognitive load. The system designers asked the coordinator to give the experimental group a more challenging scenario, which favors the control group. Third, the evaluations may be subject to confirmation bias[7], since the ICU director, the consultant physician, and the coordinator of Carle life support program were involved in the system development and provided feedback. Nevertheless, the participants of the evaluations were not involved in the development. Therefore, the confirmation bias should be reduced.

[7]Confirmation bias is a tendency to search for or interpret information in a way that confirms one's preconceptions, leading to statistical errors [75].

6.6 Summary and Future Work

In this chapter, we first modeled the individual medical staff's responsibility and interactions in cardiac arrest resuscitation and decomposed their overall tasks into a set of distinct cognitive tasks. We documented our findings using the contextual design methodology. We then developed a medical Best Practice Guidance (BPG) system to reduce medical staff's cognitive load and thus help them better adhere to the workflows in real-time. The developed system was evaluated by the medical staff under clinical simulations with a medical manikin. The evaluation results showed significant reduction of cognitive load and performance improvement. As the ICU director of Carle Foundation Hospital has remarked,

> *"It is expected that the use of the system will result in rapid and consistent timing of medical interventions, stricter adherence to standardized medical treatment guidelines, more accurate record keeping, and improved team situation awareness"*

Therefore, the developed system has been recommended and approved by Carle Foundation Hospital for clinical trials.

As future work, in order to more extensively and objectively evaluate the medical BPG system, we are closely working with Carle Foundation Hospital and will conduct a series of additional simulations. The medical staff's performance will be evaluated in respect of two aspects: number of deviations and time to response to critical events. We believe that these measures will be useful for objectively evaluating the quality of our system and provide diagnostic feedback for adapting the system to other medical applications.

Chapter 7

Conclusion

Medical CPHS are both safety-critical and highly complex, because medical staff need to coordinate with distributed medical devices and supervisory controller to monitor and control multiple aspects of the patient's physiology. Complexity is one of the major reasons for medical errors in medical cyber-physical-human systems (CPHS). This thesis makes novel contribution to reduce and control complexity in respect of verification complexity, cyber medical treatment complexity, and cognitive load complexity. Architectural patterns and protocols are developed to limit asynchronous communication and provide consistent control for distributed medical devices and controllers, which in turn reduces the verification complexity. A treatment validation and a workflow adaptation protocols are designed to assist medical staff to correctly perform treatments by checking preconditions, monitoring side effects, and checking expected responses. We also developed a best practice guidance system to provide context-dependent information for reducing medical staff's cognitive load and further foster real-time adherence to the best practice guidelines.

Beyond the scope of this thesis, there are several open challenges.

- We currently verify the developed architectural patterns and protocols by developing system models and checking the correctness and safety properties of the models. However, there is a gap between system models and implementations. In other words, a correct system model does not guarantee a correct implementation. In order to address this issue, we plan to use a source code model checker *CBMC* [28], which is bounded model checker for C and C++ programs, to

verify the implementation of the architectural patterns and protocols.

- We believe that the developed best practice guidance system can also improve patient hand-over and transfer process. The developed system keeps track of the patient conditions and performed treatments during the transportation, so with the help from our system, physicians and nurses can resume the workflow and seamlessly continue the treatment. We plan to use sepsis as our case study and develop the system to provide end-to-end workflow guidance and seamless hand-over.

- In order to conduct clinical trials at Carle Foundation Hospital, the developed system must be approved by Food and Drug Association (FDA). In addition to follow best practice software engineering process, FDA also requires the system designers to perform risk analysis and software traceability. For risk analysis, we plan to perform fault-tree analysis to identify how an undesired or hazardous system state could be reached by a combination of software and/or hardware components faults. Then, a set of fault-tolerance requirements are derived from our fault-tree analysis to reduce the likelihood and severity of the risks. Traceability supports many software engineering activities, such as compliance verification, traceback of code, regression test selection, and requirements validation For traceability, we plan to establish a hierarchical structure with the following layers (top-down): customer requirements, system requirements, software requirements, implementation, and test cases.

Bibliography

[1] Adil Ahmed, Subhash Chandra, Vitaly Herasevich, Ognjen Gajic, and Brian W Pickering. The effect of two different electronic health record user interfaces on intensive care provider task load, errors of cognition, and performance*. *Critical care medicine*, 39(7):1626–1634, 2011.

[2] A. Al-Nayeem, Sun, Xiaokang Qiu, and L. Sha. A Formal Architecture Pattern for Real-time Distributed Systems. *Proceedings of the 30th IEEE RTSS*, 2009.

[3] R. Alur, D. Arney, E.L. Gunter, I. Lee, J. Lee, W. Nam, F. Pearce, S. Van Albert, and J. Zhou. Formal specifications and analysis of the computer-assisted resuscitation algorithm (cara) infusion pump control system. *International Journal on Software Tools for Technology Transfer (STTT)*, 5(4):308–319, 2004.

[4] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[5] VA Arlington. National science foundation investing in america's future: Strategic plan fy 2006–2011. 2006.

[6] D. Arney, M. Pajic, J.M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky. Toward patient safety in closed-loop medical device systems. In *Proceedings of the 1st ACM/IEEE ICCPS*. ACM, 2010.

[7] Philip Aspden, Julie Wolcott, J Lyle Bootman, Linda R Cronenwett, et al. *Preventing medication errors: quality chasm series*. National Academies Press, 2006.

[8] H. Attiya, G. Ramalingam, and N. Rinetzky. Sequential verification of serializability. *ACM Sigplan Notices*, 45(1):31, 2010.

[9] Christel Baier, Joost-Pieter Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[10] Jakob E Bardram and Claus Bossen. Mobility work: The spatial dimension of collaboration at a hospital. *Computer Supported Cooperative Work*, 14(2):131–160, 2005.

[11] Jakob E Bardram, Thomas R Hansen, and Mads Soegaard. Awaremedia: a shared interactive display supporting social, temporal, and spatial awareness in surgery. In *Proceedings of conference on Computer supported cooperative work (CSCW'06)*, pages 109–118. ACM, 2006.

[12] D.W. Bates and A.A. Gawande. Improving safety with information technology. *New England Journal of Medicine*, 348(25):2526–2534, 2003.

[13] G. Behrmann, A. David, and K.G. Larsen. A tutorial on uppaal. *Lecture Notes in Computer Science*, pages 200–236, 2004.

[14] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on UPPAAL. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[15] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. *Lectures on Concurrency and Petri Nets*, pages 87–124, 2004.

[16] Marc Berg. Accumulating and coordinating: occasions for information technologies in medical work. *Computer supported cooperative work (CSCW)*, 8(4):373–401, 1999.

[17] Hugh Beyer and Karen Holtzblatt. *Contextual design: defining customer-centered systems*. Elsevier, 1997.

[18] Ann M Bisantz, Rollin J Fairbanks, and Catherine M Burns. Cognitive engineering for better health care systems. *Cognitive Systems Engineering in Health Care*, page 1, 2014.

[19] Claus Bossen. Representations at work: A national standard for electronic health records. In *Proceedings of the conference on Computer supported cooperative work(CSCW'06)*, pages 69–78. ACM, 2006.

[20] Claus Bossen and Lotte Groth Jensen. How physicians 'achieve overview': A case-based study in a hospital ward. In *Proceedings of the Conference on Computer supported cooperative work (CSCW'14)*, pages 257–268. ACM, 2014.

[21] John Seely Brown and Paul Duguid. Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation. *Organization science*, 2(1):40–57, 1991.

[22] Catherine M Burns, Yukari Enomoto, and Kathryn Momtahan. A cognitive work analysis of cardiac care nurses performing teletriage. *Applications of cognitive work analysis*, pages 149–174, 2008.

[23] D.N. Card and W.W. Agresti. Measuring software design complexity. *Journal of Systems and Software*, 8(3):185–197, 1988.

[24] Chin-Liang Chang and Richard Char-Tung Lee. *Symbolic logic and mechanical theorem proving*. Academic press, 2014.

[25] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994.

[26] Carey D Chisholm, Edgar K Collison, David R Nelson, and William H Cordell. Emergency department workplace interruptions are emergency physicians "interrupt-driven" and "multitasking" ? *Academic Emergency Medicine*, 7(11):1239–1243, 2000.

[27] Edmund Clarke. The birth of model checking. *25 Years of Model Checking*, pages 1–26, 2008.

[28] Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, volume 2988 of *Lecture Notes in Computer Science*, pages 168–176. Springer, 2004.

[29] E.M. Clarke, O. Grumberg, and D.E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.

[30] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Mart-Oliet, J. Meseguer, , and C. Talcott. *Maude manual*, 2007.

[31] E.W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, 1974.

[32] L. J. Easdown, A. Banerjee, and M.B. Weinger. Simulation to assess human responses to critical events. In John D Lee, Alex Kirlik, and MJ Dainoff, editors, *The Oxford Handbook of Cognitive Engineering*. Oxford University Press, 2013.

[33] P. Feiler. Open source aadl tool environment (osate). In *AADL Workshop, Paris*, 2004.

[34] P.H. Feiler. The architecture analysis & design language (AADL): An introduction. Technical report, Software Engineering Institute, Cranegie-Mellon University, 2006.

[35] John M Field, Mary Fran Hazinski, Michael R Sayre, Leon Chameides, Stephen M Schexnayder, Robin Hemphill, Ricardo A Samson, John Kattwinkel, Robert A Berg, Farhan Bhanji, et al. Part 1: executive summary 2010 american heart association guidelines for cardiopulmonary resuscitation and emergency cardiovascular care. *Circulation*, 122(18 suppl 3):S640–S656, 2010.

[36] Geraldine Fitzpatrick and Gunnar Ellingsen. A review of 25 years of cscw research in healthcare: Contributions, challenges and future agendas. *Comput. Supported Coop. Work*, 22(4-6):609–665, August 2013.

[37] J.M. Goldman, S. Whitehead, S. Weininger, and MD Rockville. Eliciting clinical requirements for the medical device plug-and-play (MD PnP) interoperability program. *Anesthesia & Analgesia*, 102:S1–54, 2006.

[38] 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. *draft ASTM TC F*, 29, 2009.

[39] J. Gray and A. Reuter. *Transaction processing: concepts and techniques*. Morgan Kaufmann, 1993.

[40] Jeremy M Grimshaw and Ian T Russell. Effect of clinical guidelines on medical practice: a systematic review of rigorous evaluations. *The Lancet*, 342(8883):1317–1322, 1993.

[41] John R Hajdukiewicz, Kim J Vicente, D John Doyle, Paul Milgram, and Catherine M Burns. Modeling a medical environment: an ontology for integrated medical informatics design. *International journal of medical informatics*, 62(1):79–99, 2001.

[42] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. *Advances in psychology*, 52:139–183, 1988.

[43] Mark Hartswood, Rob Procter, Mark Rouncefield, and Roger Slack. Making a case in medical work: implications for the electronic medical record. *Computer Supported Cooperative Work (CSCW)*, 12(3):241–266, 2003.

[44] T. Henzinger, S. Qadeer, and S. Rajamani. You assume, we guarantee: Methodology and case studies. In *Computer Aided Verification*, pages 440–451. Springer, 1998.

[45] Jan Horsky, David R Kaufman, Michael I Oppenheim, and Vimla L Patel. A framework for analyzing the cognitive complexity of computer-assisted clinical ordering. *Journal of biomedical informatics*, 36(1):4–22, 2003.

[46] M. Jacobs, JC Verdeja, HS Goldstein, et al. Minimally invasive colon resection (laparoscopic colectomy). *Surgical laparoscopy & endoscopy*, 1(3):144, 1991.

[47] Tizneem Jiancaro, Greg A Jamieson, and Alex Mihailidis. Twenty years of cognitive work analysis in health care a scoping review. *Journal of Cognitive Engineering and Decision Making*, page 1555343413488391, 2013.

[48] Woochul Kang, Po-Liang Wu, Lui Sha, Richard B. Berlin, and Julian M. Goldman. Towards safe and effective integration of networked medical devices using organ-based semi-autonomous hierarchical control. Technical report, University of Illinois at Urbana Champaign, 2012.

[49] Woochul Kang, PoLiang Wu, Maryam Rahmaniheris, Lui Sha, Richard B Berlin, and Julian M Goldman. Towards organ-centric compositional development of safe networked supervisory medical systems. In *Computer-Based Medical Systems (CBMS), 2013 IEEE 26th International Symposium on*, pages 143–148. IEEE, 2013.

[50] C. Kim, M. Sun, S. Mohan, H. Yun, L. Sha, and T.F. Abdelzaher. A framework for the safe interoperability of medical devices in the presence of network failures. In *Proceedings of the 1st ACM/IEEE ICCPS*, pages 149–158. ACM, 2010.

[51] C. Kim, M. Sun, H. Yun, and L. Sha. A Medical Device Safety Supervision over Wireless. *Reliable and Autonomous Computational Science*, pages 21–40, 2010.

[52] Andrew L. King, Lu Feng, Oleg Sokolsky, and Insup Lee. A modal specification approach for on-demand medical systems. In *Proceedings of the Third International Symposium on Foundations of Health Information Engineering and Systems*, 2013.

[53] Catharina JM Klijn and Graeme J Hankey. Management of acute ischaemic stroke: new guidelines from the american stroke association and european stroke initiative. *The Lancet Neurology*, 2(11):698–701, 2003.

[54] Linda T Kohn, Janet M Corrigan, Molla S Donaldson, et al. *To err is human: building a safer health system*, volume 627. National Academies Press, 2000.

[55] Digna R Kool and Johan G Blickman. Advanced trauma life support®. abcde from a radiological point of view. *Emergency radiology*, 14(3):135–141, 2007.

[56] Diana S Kusunoki, Aleksandra Sarcevic, Zhan Zhang, and Randall S Burd. Understanding visual attention of teams in dynamic medical settings through vital signs monitor use. In *Proceedings of the conference on Computer supported cooperative work (CSCW'13)*, pages 527–540. ACM, 2013.

[57] Edward A. Lee. Cyber physical systems: Design challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008.

[58] Insup Lee, George J. Pappas, Rance Cleaveland, John Hatcliff, Bruce H. Krogh, Peter Lee, Harvey Rubin, and Lui Sha. High-confidence medical device software and systems. *IEEE Computer*, 39(4), Apr. 2006.

[59] John D Lee, Alex Kirlik, and MJ Dainoff. *The Oxford Handbook of Cognitive Engineering*. Oxford University Press, 2013.

[60] Wen-Shing Lee, DL Grosh, Frank A Tillman, and Chang H Lie. Fault tree analysis, methods, and applications ? a review. *Reliability, IEEE Transactions on*, 34(3):194–203, 1985.

[61] Nancy G Leveson and Clark S Turner. An investigation of the therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[62] David E Long. *Model checking, abstraction, and compositional verification*. PhD thesis, Citeseer, 1993.

[63] Blair MacIntyre, Elizabeth D Mynatt, Stephen Voida, Klaus M Hansen, Joe Tullio, and Gregory M Corso. Support for multitasking and background awareness using interactive peripheral displays. In *Proceedings of the symposium on User interface software and technology*, pages 41–50. ACM, 2001.

[64] Orit Manor-Shulman, Joseph Beyene, Helena Frndova, and Christopher S Parshuram. Quantifying the volume of documented clinical information in critical illness. *Journal of critical care*, 23(2):245–250, 2008.

[65] Adam Marcus. Once a tech fantasy, plug-and-play or edges closer to reality. *Anesthesiology News*, 33(1), 2007.

[66] Thomas J McCabe. A complexity measure. *Software Engineering, IEEE Transactions on*, (4):308–320, 1976.

[67] Erin McCann. Deaths by medical mistakes hit records. *Healthcare IT News*, 2014.

[68] Brian S Mittman, Xenia Tonesk, and Peter D Jacobson. Implementing clinical practice guidelines: social influence strategies and practitioner behavior change. *QRB. Quality review bulletin*, 18(12):413–422, 1992.

[69] J.B. Moseley, K. O'Malley, N.J. Petersen, T.J. Menke, B.A. Brody, D.H. Kuykendall, J.C. Hollingsworth, C.M. Ashton, and N.P. Wray. A controlled trial of arthroscopic surgery for osteoarthritis of the knee. *New England Journal of Medicine*, 347(2):81–88, 2002.

[70] Sirajum Munir, John A Stankovic, Chieh-Jan Mike Liang, and Shan Lin. Cyber physical system challenges for human-in-the-loop control. In *Presented as part of the 8th International Workshop on Feedback Computing, Berkeley, CA*, 2013.

[71] Christopher Nemeth, Michael O?Connor, P Allan Klock, and Richard Cook. Discovering healthcare cognition: the use of cognitive artifacts to reveal cognitive work. *Organization studies*, 27(7):1011–1035, 2006.

[72] Christopher Nemeth, Robert L Wears, Sachin Patel, Greg Rosen, and Richard Cook. Resilience is not control: healthcare, crisis management, and ict. *Cognition, Technology & Work*, 13(3):189–202, 2011.

[73] Christopher P Nemeth, Richard Cook, Michael O Connor, P Allan Klock, et al. Using cognitive artifacts to understand distributed cognition. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 34(6):726–735, 2004.

[74] Mark D Neuman, Jennifer N Goldstein, Michael A Cirullo, and J Sanford Schwartz. Durability of class i american college of cardiology/american heart association clinical practice guideline recommendations. *JAMA*, 311(20):2092–2100, 2014.

[75] Raymond S Nickerson. Confirmation bias: A ubiquitous phenomenon in many guises. *Review of general psychology*, 2(2):175, 1998.

[76] M. Pajic, Z. Jiang, I. Lee, O. Sokolsky, and R. Mangharam. From verification to implementation: A model translation tool and a pacemaker case study. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2012 IEEE 18th*, pages 173–184. IEEE, 2012.

[77] M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. Goldman, and I. Lee. Model-driven safety analysis of closed-loop medical systems. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, 2012.

[78] Emily S Patterson, Richard I Cook, David D Woods, and Marta L Render. Examining the complexity behind a medication error: generic patterns in communication. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 34(6):749–756, 2004.

[79] Sharoda A Paul and Madhu C Reddy. Understanding together: sensemaking in collaborative information seeking. In *Proceedings of the conference on Computer supported cooperative work (CSCW'10)*, pages 321–330. ACM, 2010.

[80] Mandy Perrin and Anthony Fletcher. Laparoscopic abdominal surgery. *Continuing Education in Anaesthesia, Critical Care & Pain*, 4(4):107–110, 2004.

[81] Charles Perrow. *Normal accidents: Living with high risk technologies*. Princeton University Press, 2011.

[82] Anand Ranganathan and Roy H. Campbell. What is the complexity of a distributed computing system? *Complexity*, 12(6):37–45, 2007.

[83] John Rushby. Systematic formal verification for fault-tolerant time-triggered algorithms. *IEEE Transactions on Software Engineering*, 25:651–660, September 1999.

[84] Lui Sha. Using Simplicity to Control Complexity. *IEEE Software*, 18(4):20–28, Jul./Aug. 2001.

[85] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2006.

[86] Xiping Song, Beatrice Hwong, Gilberto Matos, Arnold Rudorfer, Christopher Nelson, Minmin Han, and Andrei Girenkov. Understanding requirements for computer-aided healthcare workflows: experiences and challenges. In *Proceedings of the International conference on software engineering (ICSE'06)*, pages 930–934. ACM, 2006.

[87] N. Stevens, A.R. Giannareas, V. Kern, A. Viesca, M. Fortino-Mullen, A. King, and I. Lee. Smart alarms: multivariate medical alarm integration for post cabg surgery patients. In *Proceedings of the 2nd ACM SIGHIT symposium on International health informatics*, pages 533–542. ACM, 2012.

[88] John C Thomas and John T Richards. Achieving psychological simplicity: Measures and methods to reduce cognitive complexity. *Human-Computer Interaction: Design Issues, Solutions, and Applications*, page 161, 2009.

[89] A. Valmari. The state explosion problem. *Lectures on Petri nets: advances in Petri nets*, page 429, 1998.

[90] Jiacun Wang. Emergency healthcare workflow modeling and timeliness analysis. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 42(6):1323–1331, 2012.

[91] Diane B Wayne, Aashish Didwania, Joe Feinglass, Monica J Fudala, Jeffrey H Barsuk, and William C McGaghie. Simulation-based education improves quality of care during cardiac arrest team responses at an academic teaching hospital: a case-control study. *Chest Journal*, 133(1):56–61, 2008.

[92] David D Woods, Emily S Patterson, and Richard I Cook. Behind human error: taming complexity to improve patient safety. *Handbook of Human Factors and Ergonomics in Health Care and Patient Safety. London: Lawrence Erlbaum*, pages 459–76, 2007.

[93] Po-Liang Wu, Woochul Kang, Abdullah Al-Nayeem, Lui Sha, Richard B Berlin Jr, and Julian M Goldman. A low complexity coordination architecture for networked supervisory medical systems. In *Proceedings of the 4th ACM/IEEE ICCPS*, pages 89–98. ACM, 2013.

[94] Po-Liang Wu, Min-Young Nam, Jeonghwan Choi, Alex Kirlik, Lui Sha, and Richard Berlin. Medical best practice guidance, navigation and control (GN&C) system for supporting situation awareness. Technical report, University of Illinois at Urbana Champaign, 2015.

[95] Po-Liang Wu, Dhashrath Raguraman, Lui Sha, Richard B. Berlin, and Julian M. Goldman. A treatment validation protocol for cyber-physical-human medical systems. In *Software Engineering and Advanced Applications (SEAA), 2014 EUROMICRO Conference on*. IEEE, 2014.

[96] Po-Liang Wu, Lui Sha, Richard B. Berlin, and Julian M. Goldman. Safe workflow adaptation and validation protocol for medical cyber-physical. In *Software Engineering and Advanced Applications (SEAA), 2015 EUROMICRO Conference on*. IEEE, 2015.

[97] H. Yun, P.L. Wu, M. Rahmaniheris, C. Kim, and L. Sha. A reduced complexity design pattern for distributed hierarchical command and control system. In *Proceedings of the 1st ACM/IEEE ICCPS*, pages 42–49. ACM, 2010.

Appendix A

Introduction of Model Checking and UPPAAL

A.1 Model Checking

During the last two decades, research in formal methods has led to the development of several very promising verification techniques, such as model checking [27] and theorem proving [24], that facilitate correctness and safety of system designs.. These techniques are accompanied by powerful software tools that can be used to automate various verification steps, simulate system behaviors and generate source code. Several studies have shown that formal verification procedures would have revealed the software errors [9].

Model checking is a formal verification technique that explores all possible system states in a brute-force manner. In this way, it can be shown that a given system model truly satisfies a certain property. A model checking problem can be stated as the following:
Let M be a Kripke structure (i.e., finite state machine specifications). Let f be a formula of temporal logic (i.e., the properties to be verified). Find all states s of M such that $M, s \vDash f$. [27] In general, the steps of applying model checking can be summarized as the following:

1. Build a model for the system, typically as a set of timed automata.
2. Formalize the properties to be verified using temporal logic.
3. Use the model checker to generate the space of all possible states and to exhaustively check whether the properties hold in all of the possible dynamic behaviors of the model

Model Checking has a number of advantages, and followings are some of these advantages:

- **Diagnostic counterexamples:** If any property is not satisfied, the model checker will produce a counterexample of execution trace that shows why the property does not hold. System designers can examine the counterexample and debug the systems.
- **Expressiveness of temporal logic:** Temporal logic can easily express many of the properties that are needed for reasoning about correctness and safety of systems.
- **Partial verification:** Properties can be verified individually. Thus, system designers can focus on the essential properties first even without a complete system design.

However, model checking also has several disadvantages:

- **State explosion problem:** The number of system states needed explore may easily exceed the amount of available computer memory. Despite the development of several very effective methods, such as abstraction and compositional verification [62], to address this problem, models of realistic systems may still be too large.
- **Verifying a system model and not the actual system:** Model checking only verifies the system model (i.e. finite state machines) instead of real system implementation. Therefore, it is possible that the system model is correct but the implementation contains serious errors. Complementary techniques, such as testing, are required.
- **Completeness of correctness and safety properties:** Model checking relies on system designers to specify the verification properties, and there is no guarantee that the specified properties are complete and accurate.

Despite the above limitations, model checking can still provide a significant increase in the level of confidence of a system design.

A.2 UPPAAL

UPPAAL [14] is a integrated tool for both validation (via graphical simulation) and verification (via automatic model-checking) of real-time systems. In UPPAAL, systems are modeled using timed-automata, which are finite state machines with clocks.

Definition 11 *A timed automaton is defined as a tuple* $< Q, \Sigma, C, A, E, q_0, I >$*, where*

Q *is a set of the states of W.* Σ *is a set of actions of W.*

C *is a set of the clocks.*

A *is a set of actions.*

$E \subseteq Q \times A \times \Sigma \times B(C) \times P(C) \times Q$ *is a set of edges, called transitions of W, where*

$B(C)$ *is the set of boolean clock constraints involving clocks from C, and* $P(C)$ *is the powerset of C.*

$I: Q \leftarrow B(C)$ *assigns invariants to states.* $q_0 \in Q$ *is the initial state.*

A timed-automaton is represented as a graph which has states as nodes and transitions as edges between states. States can be in one of the four different types: **initial**, **urgent**, **committed**, and **normal**. Each timed-automaton must have exactly one **initial** state marked as double-circle. In an **urgent** state (marked with *u*), clock is frozen. In other words, time is not allowed to pass when the system is in an urgent state. In a **committed** state (marked with *c*), clock is also frozen like urgent state, but the next transition must be an outgoing transition from a **committed** state.

Edges are annotated with **guards**, **updates**, and **synchronizations**. A **guard** is an expression which uses the variables and clocks of the model in order to indicate whether a transition is enabled or not. An **update** is an expression that is evaluated when the corresponding edge is fired. The **synchronization** is the basic mechanism used to coordinate the action of two or more UPPAAL processes. Specifically, the synchronization enforces two (or more) processes to take a transition at the same time. A channel should be declared as *chan c*; one process will have an edged annotated with *c!* and the other(s) process(es) another edge annotated with *c?*. A synchronization pair is chosen non-deterministically if several combinations are enabled.

UPPAAL uses a simplified version of timed computation tree logic (TCTL) as the query language to specify verification properties. Like in TCTL, the UPPAAL query language consists of path formula

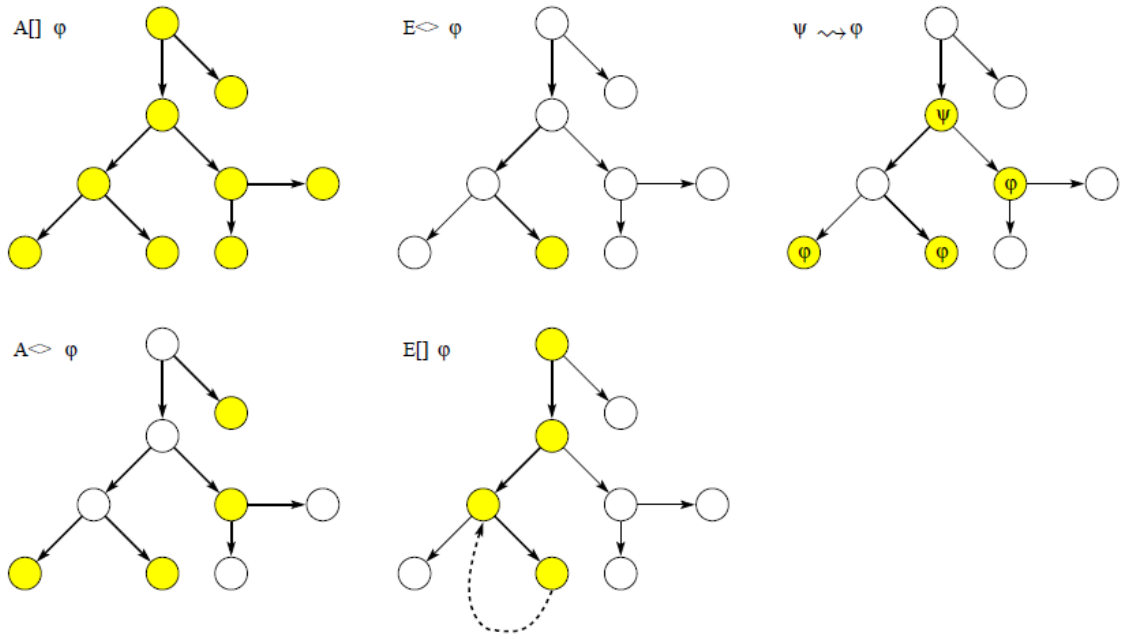


Figure A.1: Path formulae supported in UPPAAL. The filled states are those for which a given state formula φ holds. Bold edges are used to show the paths the formula evaluate on. This figure is from [14].

and state formula. State formula describe individual states, whereas path formula quantify over paths or traces of the model. A graphical representation of the query language is in Fig A.1. The specific types of properties that can be expressed in the UPPAAL query language can be classified as:

- **Reachability properties:** Rechability properties express that if there exist a path starting at the initial state, such that a property, φ, is eventually satisfied along that path. This property is specified with the syntax E<> φ.
- **Safety properties:** Safety properties express that if a property, φ, is satisfied in all the states of an execution path. There are two possible cases. A[]φ expresses that φ is satisfied in all reachable states. E[]φ expresses that there exist a maximal path such that φ is always satisfied.
- **Liveness properties:** Liveness properties express that a property, φ, will eventually be satisfied. This property is specified with the syntax A<> φ. Another useful form is the *leads to* or *response* property, expressed as $\varphi - - > \psi$, which means that whenever φ is satisfied, then eventually ψ

will be satisfied.

- **Deadlock properties:** Deadlock properties express that if a state is in a deadlock. A state is in a deadlock if there are no outgoing transitions neither from the state itself or any of its delay successor.

Appendix B

Introduction of Integrated Clinical Environment (ICE) Architecture

Integrated Clinical Environment (ICE) architecture [37] provides a centralized supervisory framework for integrating networked medical devices to enable the development of supervisory protocols. Logically, ICE is a hierarchical control architecture, as illustrated in Figure B.1.

Medical Devices are responsible for measuring the patient conditions(e.g. heart rate, blood pressure) and providing treatments or interventions (e.g. drug administration, assisted ventilation). **Adapters** facilitate communication between the medical devices and the Supervisor through wireless network. **Medical Device Plug and Play (MDPnP) controller** is to provide an integrated communication interface for the Supervisor with responsibilities including data routing, format translation, and quality of service (QoS) enforcement. **Supervisory protocols** are responsibility for encoding clinical knowledge and coordinating with medical and medical devices. In this thesis, we take advantage of such an integrated framework and develop the validation and coordination protocols on as part of the supervisory protocols in order to reduce medical errors and improve patient safety.

The high-level scenario can be described as the following. Medical staff issue a treatment command through user interface. The supervisory protocols validate the treatment command against the patient conditions provided exported by the MDPnP controller. If the supervisory protocols detect any potential safety hazards, an exception is raised to the medical staff. Otherwise, the supervisory protocols

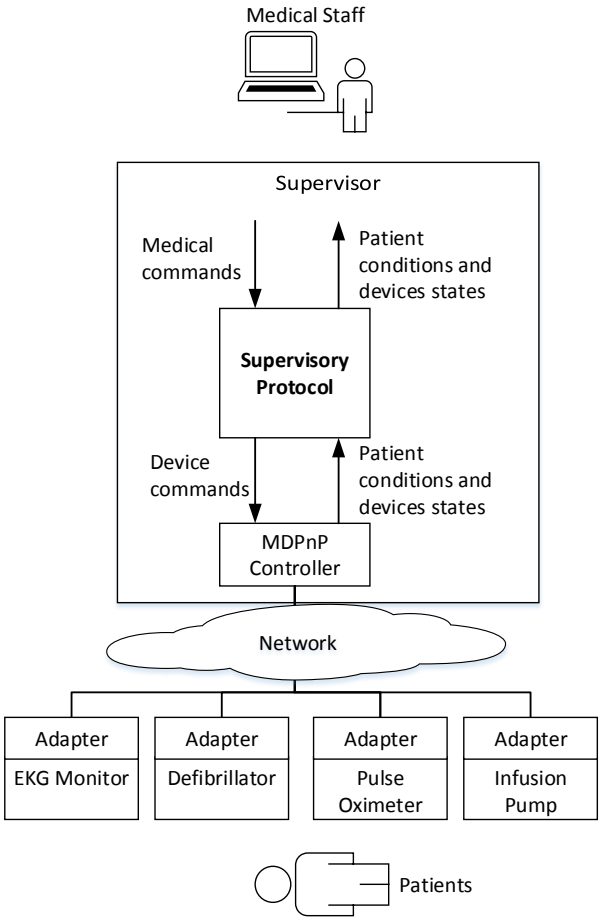


Figure B.1: The Integrated Clinical Environment (ICE) Architecture

request the medical devices to perform treatments through MDPnP controller. On the other hand, the supervisory protocols dynamically monitor the patient conditions and alert the medical staff to adjust the treatment if a patient adverse event is raised.