Medical Device Interoperability With Provable Safety Properties

David Eric Arney
University of Pennsylvania, dave@davearney.org

Medical Device Interoperability With Provable Safety Properties

Abstract
Applications that can communicate with and control multiple medical devices have the potential to radically improve patient safety and the effectiveness of medical treatment. Medical device interoperability requires devices to have an open, standards-based interface that allows communication with any other device that implements the same interface. This will enable applications and functionality that can improve patient safety and outcomes.

To build interoperable systems, we need to match up the capabilities of the medical devices with the needs of the application. An application that requires heart rate as an input and provides a control signal to an infusion pump requires a source of heart rate and a pump that will accept the control signal. We present means for devices to describe their capabilities and a methodology for automatically checking an application's device requirements against the device capabilities.

If such applications are going to be used for patient care, there needs to be convincing proof of their safety. The safety of a medical device is closely tied to its intended use and use environment. Medical device manufacturers create a hazard analysis of their device, where they explore the hazards associated with its intended use. We describe hazard analysis for interoperable devices and how to create system safety properties from these hazard analyses. The use environment of the application includes the application, connected devices, patient, and clinical workflow. The patient model is specific to each application and represents the patient's response to treatment. We introduce Clinical Application Modeling Language (CAML), based on Extended Finite State Machines, and use model checking to test safety properties from the hazard analysis against the parallel composition of the application, patient model, clinical workflow, and the device models of connected devices.

Degree Type
Dissertation

Degree Name
Doctor of Philosophy (PhD)

Graduate Group
Computer and Information Science

First Advisor
Insup Lee

Keywords
formal methods, interoperability, patient safety

Subject Categories
Computer Sciences

MEDICAL DEVICE INTEROPERABILITY WITH PROVABLE SAFETY PROPERTIES

David Arney

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

2019

Acknowledgments

I have been privileged to work with and learn from many amazing people. This work has benefited from conversations with literally hundreds of people at the MD PnP Lab and conferences and workshops around the world

I'd like to thank the clinicians that have allowed me the privilege of accompanying them as they care for patients and for sharing their challenges, frustrations, and hopes. Soojin Park, Marco Zenati, John Walsh, and Margaret Mullen-Fontino among many others, as well as Geoff Rance, Bill Driscoll, Ryan Ford, Luis Melendez, and Rick Schrenker for sharing the challenges of clinical biomedical engineering

Julian Goldman for his support, vision of using interoperability to improve patient safety and outcomes, and founding the MD PnP Program at Massachusetts General Hospital where much of this work was done

Jeff Plourde, Jeff Peterson, and all of my colleagues at MD PnP for sharing the dream

Carlen Blackstone from Emmaus High School and Raymond McDowell, Alyce Brady, Chris Latiolais from Kalamazoo College for their teaching and mentorship

My fellow students in the PRECISE group, especially Andrew King, BaekGyu Kim, Miroslav Pajic, and Arvind Easwaran, who helped me puzzle out the initial semantics for

CAML

At the FDA, Sandy Weininger, Paul Jones, and Raoul Jetley

My fellow Fellows and Affiliates at the Berkman Klein Center, and especially the Healthcare Working Group, for support, encouragement, and wonderful discussions

The members of my committee for their mentorship

My advisor, Insup Lee, for his guidance and support

And finally my family. Laurie, Zachary, Eleanor and all of you who have helped and supported me on this journey

ABSTRACT

MEDICAL DEVICE INTEROPERABILITY WITH PROVABLE SAFETY PROPERTIES

David Arney

Insup Lee

Applications that can communicate with and control multiple medical devices have the potential to radically improve patient safety and the effectiveness of medical treatment. Medical device interoperability requires devices to have an open, standards-based interface that allows communication with any other device that implements the same interface. This will enable applications and functionality that can improve patient safety and outcomes.

To build interoperable systems, we need to match up the capabilities of the medical devices with the needs of the application. An application that requires heart rate as an input and provides a control signal to an infusion pump requires a source of heart rate and a pump that will accept the control signal. We present means for devices to describe their capabilities and a methodology for automatically checking an application's device requirements against the device capabilities.

If such applications are going to be used for patient care, there needs to be convincing proof of their safety. The safety of a medical device is closely tied to its intended use and use environment. Medical device manufacturers create a hazard analysis of their device, where they explore the hazards associated with its intended use. We describe hazard analysis for interoperable devices and the creation of system safety properties from these hazard analyses. The use environment of the application includes the application, connected devices, patient, and clinical workflow. The patient model is specific to each application and represents the patient's response to treatment. We introduce Clinical Application Modeling Language (CAML), based on Extended Finite State Machines, and use model checking to test safety properties from the hazard analysis against the parallel composition of the application, patient model, clinical workflow, and the device models of connected devices.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Medical devices are increasingly being connected together to collect vital signs, propagate alarms, and feed data to electronic health record systems. There are many standards organizations, companies, and non-profit organizations working on medical device interoperability and, ironically, they all seem to have slightly different definitions of the word. Since interoperability means so many different things to different people, we need to start by defining what we mean by it here.

Conceptually, interoperability means that pieces from different sources should work together. The idea is that if you need a part to build something you can give requirements for your part and, as long as you've correctly specified the important aspects, any part from any manufacturer that meets your specifications should work. This concept has been standard practice in manufacturing for about the last 200 years, and is a basic tenet of object-oriented software, but is only recently making inroads in the medical domain.

There is an important distinction between interoperable devices as defined here and interconnected devices. Device connectivity simply means that there is a way to connect a device to something else. In most cases, this ‘something else’ is made by the same manufacturer. We use the term interoperability only for devices with an open, standards-based interface that allows communication with any other device or software application that implements the same interface. All interoperable devices are connected, but not all connectivity solutions are interoperable.

Interoperability Standards. The concept of interoperable medical devices is not particularly new, but adoption has been very slow. Early electronic medical devices in the 1950s had analog output ports, and some devices sold today still have such ports. Analog ports do allow devices to connect to a variety of systems, but they are limited to carrying a single variable per port and they do not include meta-data such as the device type, unique ID, or signal processing parameters. Manufacturers each decide what voltage range to use, whether to have a linear or logarithmic relationship between the voltage and the mapped variable, and many other aspects of the analog signal, so even seemingly simple analog interfaces are beset with the same semantics and protocol compatibility problems as digital interfaces. In the 1970’s the Medical Information Bus (MIB) standard offered a degree of interoperability, but failed in the marketplace. The ISO 1073 family of standards grew out of the MIB work and inspired, in turn, the ISO/IEC 11073 standards that are still being developed today. While there are a limited number of implementations of the 11073 standards, there have been offshoots that are more widely used. Most notable of these are the Continua Alliance, which adapted and reused portions of 11073 to create a new standard for personal health (home use) medical devices, and the reuse of the 11073-10101 terminology set by several other groups. Continua’s Personal Health

Domain (PHD) adaptation of 11073 has since been adopted as an official part of 11073.

Data encoding, and especially terminology, has been a major issue in medical device connectivity. There are many groups working on medical terminologies and ontologies. 11073 includes portions of medical terminology, along with communications protocols. Other major terminologies come from Health Level 7 (HL7), and the Systematized Nomenclature of Medicine–Clinical Terms (SNOMED-CT) developed by the College of American Pathologists and maintained by the International Health Terminology Standards Development Organization. In this thesis, we largely stay away from questions of terminology, using English-language terms to enhance readability. Standardizing terminology is largely a political process, though with a need for strong technical support. The group Integrating the Healthcare Enterprise (IHE) is currently working on a promising set of domain-specific terminologies under the Rosetta name. Their Rosetta table for ventilators, for example, provides a normalized technical vocabulary for terms used on ventilator electronic interfaces. Interoperability can not be achieved without much more work on terminology, whether through Rosetta or another pathway. At present, a common approach to build a terminology by adding individual terms with each manufacturer's definition of a vital sign. This pushes the problem to the eventual users of the standards, who now have to decide whether each of the multiple definitions of, for instance, respiratory rate, is appropriate for their application. Eventually, we would hope for a common vocabulary of terms with standardized, well-understood semantics; this will be necessary to enable the types of systems described here.

Data Logging and Applications. Medical device interoperability by itself is useless – interoperability is only helpful insofar as it enables applications and functionality that are otherwise impossible. Two important abilities enabled by device interoperability are centralized data logging and the ability to run software applications that interact with multiple devices.

Standards-based, normalized, time-synchronized recording of device messages and patient physiological data will enable significant shifts in the practice of medicine. A coordinated record of device interactions will enable better post-market surveillance of medical devices, allowing for easier adverse event analysis and meaningful root cause analysis [13].

The real benefit of an ICE-style data logger will be in improvements to our understanding of basic medicine. "Evidence-based medicine" is the practice of basing medical decisions and treatments on scientific evidence. To engineers, scientists, and many practitioners in other domains, this sounds like an oxymoron – what else would doctors be doing? Clinical studies are expensive, and large scale clinical studies, especially of thing that are believed to be best practices, are difficult to fund and conduct. As we add the capability to routinely document high-resolution information about treatments and outcomes, we will finally be able to realize the promise of evidence-based medicine to improve patient safety and outcomes.

A major difference between the design of physiologic closed-loop control systems and control systems in other domains, such as chemical plants or aircraft, is our inability to predict in detail how a specific patient will respond to treatment. When an airplane is designed, engineers can mathematically model its behavior and even fly the new design in simulation before a single part is made. They can try different designs in a simulated environment and experiment with changing individual variables to see how they effect the plane's overall performance- for instance, changing the shape of a control surface may allow for sharper turns, but at the cost of increasing drag in level flight. When you or I go to the hospital, the doctors predict the outcome of potential treatments based on their knowledge and experience of treating similar patients. Drug dosages are based on gross measurements like height and weight, rather than on an actual measurement, or even estimate, of a patient's individual rate of absorption of the drug. This is like estimating the gas mileage of a car based solely on its length without taking into account its engine size, aerodynamics, or load. In medicine, this works because medications are give to effect. That is, the dose is increased until the desired effect is observed, with the physician closing the loop.

An ICE-style data logger offers the potential to build a large database recording the specific responses of individual patients to particular courses of treatment. This database will improve our knowledge of medicine in two ways. First, it would immediately benefit the study of treatments across populations. The database would be an unprecedented resource for outcomes research. Second, such a database could build a record of an individual's reaction to treatment over time. If a doctor knows that last time the patient was in the hospital, they tended to react in a particular way to a particular treatment, then the doctor can tailor the treatment to the patient the next time. Medical data logging is the subject of ongoing AAMI standards activity with a new standard nearing completion in Spring 2019. Data logging can be done locally in each connected device, in an appliance on the local network, or in the cloud[64].

Using the data log faces technical and political barriers. The political barriers are often much more formidable. Probably the hardest barrier to surmount is concerns about liability. Liability seems to be used (sometimes with excellent reason!) as an argument against the implementation of every new technology in healthcare. In the case of the data logger, there are liability concerns from device manufacturers, healthcare providers, and patients. Other barriers to adoption include: patient privacy and data security [12], data ownership, and liability issues.

The data logger will contain protected health information (PHI). Different jurisdictions have different rules, which are sometimes contradictory. Who is responsible for ensuring compliance? Suppose (and this is based on a real example too complex for a non-lawyer to explain) that California has a requirement that patient data related to workplace chemical exposure be retained for 10 years, and Oregon has a regulation that patient data related to a particular medical condition be destroyed within 5 years of a patient's death. If a patient living in New Mexico has a workplace chemical exposure in California, develops that condition, and is treated in Oregon by a HDO that stores its records on a server in Arizona, which regulation should they comply with?

Right now, in hospitals around the world, patients are connected to medical devices that record physiologic data and provide treatments. That data could feed into a data repository, but instead it scrolls by unrecorded. Development of an ICE-style data logger has the potential to revolutionize healthcare by enabling both a better understanding of the effectiveness of medical treatments across populations, as well as allowing for individualized treatment based on a patient's history.

Applications that can communicate with and control multiple medical devices have the potential to radically improve patient safety and the effectiveness of medical treatment. Many doctors, nurses, and other clinicians have great ideas that could improve the efficiency and effectiveness of medical devices. These ideas frequently require devices to share information or coordinate their actions. But these ideas are wasted because medical devices have very limited electronic interfaces and do not allow other devices to read their data or make changes to their settings.

Interoperable Devices. To get an application to work with a set of devices, we first need the devices to implement a standardized electronic interface. In other words, we need interoperable devices. We also need a platform to run the application, and we need the networking infrastructure to connect everything together. These pieces are included in the ASTM 2761-09 'ICE' standard. An overview of the system is shown in Figure 1.1, and the architecture is discussed in detail in Chapter 4.

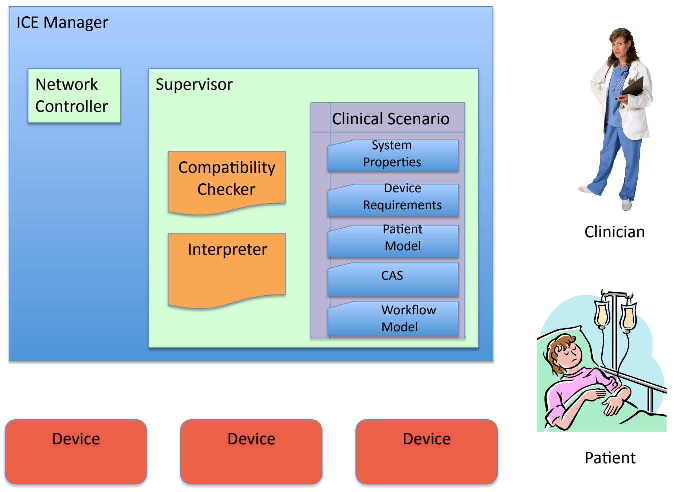


Figure 1.1: Components of the System

Medical systems are used within clinical workflows by clinicians to treat patients. An application that is safe and effective in one clinical context may be unsafe in the hands of a different user in a different environment. Medical context is an important part of demonstrating system safety [38]. In this work, we include clinician and workflow models to capture some of the essential aspects of the environment.

Overview. This thesis addresses two technical problems that present barriers to the adoption of interoperable medical devices.

First, we need to match up the capabilities of the medical devices in the system with the needs of the application. An application that requires heart rate as an input and provides a control signal to an infusion pump requires a source of heart rate and a pump that will accept the control signal. The ICE architecture requires devices to provide a device model that describes their capabilities. We present a format and set of required fields for the device model and a format for the application's device requirements along with a methodology for automatically checking the device requirements against the device models using model checking.

Second, if such applications are going to be used for patient care, there needs to be convincing proof of their safety. The safety of a medical device (and these applications are medical devices in their own right) is closely tied to their intended use and use environment. We provide an architecture and methodology for formulating and testing safety properties of such medical applications, taking into account the clinical workflow in which they are used.

We want to ensure that medical device plug-and-play systems are acceptably safe for their intended use in a particular use environment. We define safety in the usual way for medical devices, as freedom from unnecessary risk, where risk is the product of the probability and severity of a known hazard. The use environment for these systems includes the clinical workflow they are used in as well as the specific set of devices connected to the system at the time of use. The challenge is to represent the intended use of the application, the capabilities of the devices, the clinicians' actions in their workflow, and the safety properties in a way such that the system safety can be evaluated.

We begin with the hazard analysis for the medical application, which lists new hazards introduced by assembling the devices into a system with a coordinating application.

The essential inputs for this are:

1. hazard analysis, and resulting safety properties
2. clinical workflow model
3. application model
4. patient model
5. device models for each device
6. device requirements

The actual checks are done in two stages. First, we check device models against device requirements to see whether the set of connected devices at the time of checking satisfy the requirements of the application as defined by the application developers. Second, we check safety properties, again defined by the application developers, against the combination of clinical, application, device, and patient models. This tells us whether the system as assembled includes any of the known hazards as documented by the application developer and represented by the system safety properties.

Contributions. This thesis is structured around three primary contributions.

The first is a formal definition of components and interfaces for medical device interoperability. We describe the Clinical Application Modeling Language (CAML), a workflow language for describing clinical applications and caretaker workflows, which supports formalization of device models and an analysis of the types of device requirements supported by each type of device model. ASTM 2761, ISO 11073 and others introduce high level concepts around medical device interoperability. However, concepts like device models and safety requirements remain either undefined or too inexpressive to allow verifying properties using devices from multiple manufacturers. Ad-hoc data mapping and extensive integration work and testing remain the norm. We address this by defining a set of components and interfaces sufficient to enable proving realistic safety properties.

Next, we present a methodology for checking safety properties derived from hazards against the system composed of device models, clinical applications, patient model, and clinical workflows.

Finally, we discuss a hazard analysis for systems of connected medical devices from which we can draw common safety properties. Section 5.1 and Appendix A include a systematic analysis of system-level safety hazards for plug-and-play medical device systems, which serves as a source of properties to be checked.

Finally, we put these pieces together in a way that supports modeling real-world clinical use cases at a sufficient level of detail to support useful safety properties and support this with two detailed case studies.

Limitations, Gaps, and Future Work. Each chapter of this thesis ends with a section on limitations, gaps, and future work. These sections explain some of the constraints and assumptions under which the work described in the chapter was produced.

The limitations section adds more context around the work to explain simplifications and assumptions that were used. In some cases, work was constrained to one aspect of a problem to keep the scope manageable. We have identified what we consider some of the most important limitations and considerations for anyone thinking of following parts of this approach. Clinical applications we did not consider, different hardware and software architectures, and the use of different middlewares, among other factors, will impose additional constraints on system design and safety properties. Careful consideration of the limitations documented here and analysis to identify other limitations that may apply to novel systems, is necessary before applying any of these techniques to another system.

Gaps are areas where more needs to be solved; either areas that were considered out of scope for this work but relevant for future exploration, or where there was a gap between what we addressed and the way these systems would be built and deployed for clinical use.

The future work section explores promising areas for future examination. Future work items may involve the straightforward application of known techniques or more fundamental theoretical work that needs to be done to address the problem.

In this Chapter, we have introduced the problem of medical device interoperability. This Section discusses some of the limitations, gaps, and future work related to interoperability in general. Later Chapters, particularly Chapter 2 will discuss specific standards and implementations of interoperable medical systems and more specific limitations, gaps, and future work.

Medical interoperability is a rich problem space and no one thesis can address all the problems. This work is focused on acute care and bedside networks, especially on systems for real-time decision support, smart alarms, safety interlocks, and physiologic closed-loop control of devices. The term interoperability is also used in the medical space to talk about systems for connecting devices and other data sources to electronic medical records. These systems, as well as health and wellness systems for home use, are generally not built under the same kinds of quality systems as acute care devices, involve different analyses of hazards and risks, and are out of scope for this work.

In this thesis, we model the information that needs to be exchanged in the form of device models and requirements, application, clinician, and patient models and system safety properties and do not explicitly model the network components or network requirements. The focus is on what information needs to move and not the mechanisms for moving it, although network considerations are of course important in a networked real-time system.

Two other limitations will reappear and be described in multiple chapters, as they have aspects relevant to modeling, the architecture for proving safety properties, the techniques chosen to prove properties, and the case studies. This thesis discusses modeling and verifying properties of a single clinical process at a time. Although it is likely that clinicians will want to run multiple applications simultaneously on a single patient, composing applications and safety properties is left as future work.

Second, the system described here does not allow directly modeling or proving safety properties about continuous dynamics. Many important safety properties can be modeled without continuous dynamics, and abstracting away from them makes analyzing properties more tractable. However, some aspects of device and patient models, such as pharmacokinetics and pharmacodynamics, require continuous dynamics. Hybrid modeling of interoperable systems, whether in healthcare or not, is a promising direction for future work. This limitation is addressed in more detail in Chapter 3, with respect to code generation, and Chapter 6, with respect to the case studies.

Finally, we use English-language terms in place of a technical terminology set or ontology. Much good work has been done in formalizing medical, medical device, and system component terminology but much still remains to be done.

Chapter 2

Medical Device Interoperability

Installing a new mouse on a computer used to be a painful experience. First, you had to buy the right one for your computer. Once you had a mouse with the right physical connector, you could plug it in and start to configure the software. On a DOS or Windows machine, you would edit config.sys to tell the operating system that there was a mouse attached, and which serial port it was on (COM1, COM2, PS/2, etc.). Things were a bit easier on Macintosh and other manufacturers that produced both the operating system and hardware, but only because their computers would only work with a mouse you bought from them. Today, it's much easier – you can get a USB mouse from any manufacturer, plug it into your computer, whether it's a mac, any variant of Windows, or running Linux, and it just works.

Plug-and-Play. A "plug-and-play" (PnP) system is one where you can connect a device and have it work automatically. This is in contrast to systems where you might have to load a driver, change network settings, or otherwise customize the configuration.

Most PnP systems are designed to work with devices from multiple vendors and thus are based on standards. Standardization allows manufacturers to implement well-documented interfaces with confidence that their implementation will be compatible with others.

It's not usually feasible to test every possible combination of devices in a PnP system. Instead, devices are tested to make sure that their interface conforms to the standard and the system is designed in such a way that any conforming device will work correctly. Testing conformance to the standard is frequently done by a third party (i.e., not the manufacturer or user), which can be an independent test house or a consortium that certifies conformance to one particular standard.

USB devices are a common example of a plug-and-play system. When you buy a USB memory stick, you don't have to buy one specific to a particular computer manufacturer, and the memory stick manufacturer doesn't have to test it with each operating system variant, motherboard, or even each USB chipset. When a manufacturer creates a new USB device, they build it according to published specs and then send it to one of a number of independent test houses. If the device passes testing, the manufacturer can put a USB logo on it and sell it as a compatible device.

Plug and play is widespread in consumer electronics, where general design principles and best practices have been developed that can be reused in the medical domain. Although data communication protocols and the specific purpose of connectivity are different in different domains, PnP systems go through a similar sequence of events as devices are connected, used, and disconnected. Such systems also need to handle similar failure modes, such as the unexpected disconnection of a device. In this description, we use the term 'device', but the description applies to software applications as well as physical devices.

Plug-and-Play Lifecycle. The first phase of the use of a device in a PnP system is connection. The device must be connected to the network before it can be used. It may be connected with a wire, in which case connection means physically plugging in the wire, or it may be connected wirelessly. The connection phase also includes discovery, where the system becomes aware that a device capable of communication has been attached. Discovery may be followed by authentication and authorization. Authentication checks that a device is what it claims to be. This can be done by reference to a third-party authority, for instance if a device sends an identifying message that is signed by a certificate authority. Authentication may also include functional testing, where the system tries to use some functions of the connected device and checks that the expected response is returned – this is analogous to a power-on self test (POST). Authorization may be another component of connection. Authorization is another check, usually done to enforce security policies, that checks whether the device is allowed to participate in the system.

After the device is physically connected, discovered, authenticated, and authorized, it can be used. In the use phase, the system communicates with the device to accomplish some task. For a USB keyboard, use involves sending keystroke data. For a web service, use means receiving some data or a request, doing processing, and returning a result.

The final generic lifecycle phase for a PnP device is disconnection. Some devices can just be unplugged, others require a shutdown sequence before physical disconnection; perhaps the most familiar example of this is the warning message shown when a USB storage device is removed without shutting it down first. Most medical devices will require a shutdown sequence. If the proper sequence is not followed, they will go into a ‘safe’ or ‘fallback’ mode designed to protect the patient if the device is accidentally connected from a controller.

Safety-critical systems like airplanes, power plants, or automobiles have traditionally been designed as monolithic, closed systems by a single manufacturer or systems integrator. Centralized development of the architecture and all components enables the manufacturer, at least in theory, to exercise complete control over all aspects of the design and implementation. There is a widely held belief that the ability of a single systems integrator to carefully document and test each component at each stage of integration makes it much easier to obtain regulatory approval [72]. Almost all medical devices are developed in this manner, with manufacturers citing safety and ease of regulatory clearance as reasons. It is also generally not seen as a drawback that this approach closes the system to third-party developers. Some manufacturers are beginning to add end-to-end encryption with the explicit aim of making it impossible for any other devices to acquire data from their systems. Despite being designed as closed ecosystems, these safety-critical systems still have very well specified interfaces between components. The central premise we are exploring here is whether these interfaces can be specified well enough that a system can be assembled in a plug-and-play manner and be proven to be safe.

Medical Device Interoperability. Medical device interoperability has the potential to reduce healthcare costs, improve patient outcomes and improve patient safety. Achieving interoperability requires that medical devices (including software applications) and other equipment share the same information model and communication protocol. This enables applications to work with any source of compatible data regardless of the manufacturer or specific device type.

A system using standalone medical devices and computers is a distributed system. There is a long and rich history of work in the field of distributed systems that can directly inform the development of interoperable medical cyber-physical systems (MCPS) [51]. One broadly accepted tenet of this work is that network architecture can be broken down into a number of layers; this is perhaps most commonly illustrated by the Open Systems Interconnection (OSI) Seven Layer Model [25]. Breaking network architecture into these layers allows designing and reasoning about them independently – a transport layer can operate on many different network and data link layers that in turn can work with a multitude of physical layers. Middleware is software that implements some of these middle layers between an application and networking hardware. There are a great number of middleware implementations with widely varying capabilities that implement various subsets of the seven layers. Choosing an appropriate middleware for a particular domain is thus a complex undertaking that requires an understanding of what applications need and expect from the network.

Systems Engineering similarly has a long history with many lessons that can inform MCPS development. The most important lesson here is that user needs must be used to validate system designs. Broadly stated, technical requirements are used for verification (that "the system was built right") and user requirements are used for validation (that "you built the right system"). A technically flawless and provably safe system that does not satisfy user needs will not be used.

Design Pillars and Clinical Requirements are intended to give metrics by which competing interoperability options can be judged. Design pillars set out high-level non-functional goals that we found to be useful in building the OpenICE implementation and that we argue are necessary for building safe and adoptable interoperability. These design pillars emerged from over ten years of collaboration with a large team building MCPS implementations. Clinical requirements aim to capture the needs of the clinical community.

2.1 Medical Interoperability Design Pillars

We have given the name Design Pillars to the set of non-functional requirements that summarize the approach that we believe is necessary to achieve safe, adoptable medical device interoperability. Other standards, including ISO 14971, IEC 60601, and FDA Guidance Documents on Risk Management [75] also include important guidance for interoperable systems. This list has a different focus, aiming to capture the normally unwritten goals and philosophy needed to achieve successful interoperability.

In developing OpenICE and other MCPS implementations, we have worked to gather input from as many experts and stakeholders as possible. Community engagement and outreach remain a priority. Community input and feedback on prior work inspired the design pillars that guided our middleware selection. Clinical focus groups helped us to select a set of clinical scenarios, from which we developed clinical requirements that captured the needs and wants of the clinical community. The clinical requirements and input from the engineering community also informed the middleware choice and helped in formulating the technical requirements we used in implementing OpenICE. Using OpenICE to build demos that we present at conferences and meetings provides a measure of clinical validation that our requirements were appropriate and feedback from users of OpenICE in clinical settings provides additional input on requested features and further requirements.

Over the years, we have gathered input on OpenICE from many stakeholders besides clinicians including industry partners, academic collaborators, and regulators including the US FDA. OpenICE has been particularly informed by input from the OR.net community [40] [41] [43] [42] and the developers of the Medical Device Coordination Framework, MDCF [45]. Inspiration has also come from work on medical device communication patterns [70], on ecosphere [44] and architectural principles for platforms, and on component-based app design [52].

The guiding element of our philosophy in developing OpenICE is that the system should be as simple as possible to meet requirements (but no simpler). As an example, when we began using the 11073-10101 terminology set, we started by adding only the elements of the data structure that were necessary to support our use cases. If there was not a specific reason to include a data element, we did not include it. We have added additional elements as needed, but our data structure is significantly smaller and less complex than the full 11073-10101 data structure. We have found OpenICE to be sufficiently expressive for building a variety of use cases even without the omitted parts.

Another way of stating this is that additions or modifications to the system should be driven directly by their necessity to meet specific requirements. A related aspect of our philosophy is that nothing should be created in the system without a strong understanding of how it will be used. Testability is also very important, and additions to the system should be amenable to testing. This has led to various diagnostic screens and software simulators, as well as unit tests built into the build process.

We developed a set of design pillars [11] to guide the development of OpenICE. These pillars are categorized as design goals, development process, or system attributes. Design goals highlight some of the high-level aims behind the design. They attempt to capture the principles behind the design choices. Development process pillars focus on the way that the system is built, and system attributes describe system-level requirements for what we want to build.

2.1.1 Design Pillars About Interoperability Goals

One ICE: The ASTM ICE (Integrated Clinical Environment) standard establishes a common set of terminology for parts of the system, but it does not specify technical details of the functionality or require a particular communication protocol. Since publication of the standard, there have been a wide variety of opinions about how the ICE architecture should be realized. We realized early on that efforts would be diluted without reaching a consensus on the design. We felt that the selection of a single middleware standard would allow these dispersed teams to build components that could coexist on the ICE platform and that will successfully interoperate when assembled. This design pillar points out that for ICE to be a viable standard, it must be internally interoperable. That is, it must be possible to demonstrate interoperability between components built at different sites. Using a common middleware and data model means that components can be written in different programming languages and run on different operating systems while still being interoperable. Silos of interoperability work that cannot successfully interoperate are self-defeating.

Visibility of runtime configuration: ICE implementations must surface the state of the system. The connection state of devices should be readily available to a user. When the system is in an undesirable state, for example lacking connectivity to a critical medical device, it is important that information be made available. A system operating with hidden states will never earn the confidence of clinicians, but neither will a system cluttered with unnecessary information. The platform must also allow for the plug-and-play assembly of medical devices and because of this the configuration at runtime is the only source of information about how the system is configured. In any system of systems, variability accrues with the addition of components which must be reflected to clinicians so they know exactly the configuration of the system they are using.

External Connectivity: ICE implementations must interface with external systems. Some examples of external systems include an EMR system, an eHealth eXchange (NwHIN), departmental systems (such as pharmacy), or network time protocol (NTP) servers inside or outside of the hospital or home.

Novel Applications: ICE implementations must enable the development of novel applications that run within their frameworks. The point of ICE is to enable new clinical applications to improve patient outcomes and safety.

Plug and Play: Components can be added to and removed from the system at any time. The system must dynamically determine and monitor the presence of components. In the interests of security, scalability, and performance components may be refused by the system for various reasons but this refusal must be surfaced per 2.1.3. Applications must handle the disappearance of required data and control sources or sinks and the appearance of new sources and sinks gracefully.

Industry Adoptability: The goal of ICE to achieve dramatic improvements in patient outcomes and safety can only be met if such systems are commercially available. To this end, ICE implementations (particularly open source implementations) should facilitate commercial reuse. At the same time, common networking pieces such as data representations must be shared and developed in common.

Human Factors: The user interface and other human factors issues need to be carefully designed and tested in realistic environments so that new hazards that are introduced are adequately controlled. For instance, when a device is operating as a component of a larger system, its front panel must display an indicator that it's under remote control.

2.1.2 Design Pillars About Development Process

Open Source: There must be an open source reference implementation. This should include the necessary tools to adapt and utilize the software, including commercial reuse. Prototype or reference implementations of a standard demonstrate the feasibility of proposed solutions and point out gaps in the standard. This does not preclude closed-source implementations and commercialization once the conceptual use of a middleware to build a platform for ICE apps has been proven. An open source reference implementation permits other implementers to perform testing of individual components without requiring developers to implement the whole standard – for instance, an aspiring application developer can test against a reference Supervisor. OpenICE is released under a BSD 2-clause license allowing commercial reuse.

Existing Standards: Interoperability must be built on standards, utilizing existing software standards to the greatest possible extent. Where existing standards must be corrected, completed, or extended, the rationale must be documented.

There are many facets of an ICE system that are identical to systems in other domains. For instance the reliable maintenance of distributed state information in a timely manner has been addressed by standards and implementations in other domains. When the uniqueness of the medical domain is asserted it must be proven before new standards should be created that parallel work in other domains.

Community Involvement: Developers of ICE implementations must maintain awareness of developments in other large-scale initiatives and relevant standards bodies. The linkages between external developments and implementation design decisions must be explicitly documented. Findings should be shared back with Standard Development Organizations where possible.

Clinical Scenarios: Requirements for OpenICE should be derived from publicly available clinical scenarios so that traceability of technical requirements can be maintained. Technical requirements must be linked to clinical requirements which are derived from clinical scenarios. Technical design will also be informed by those scenarios and linkages between design decisions and high-level clinical requirements must be documented.

Regulatory Pathway: ICE implementations operate in a regulated space. The regulators vary geographically, but the need to demonstrate the safety and essential performance of ICE systems and components is universal. To achieve this, ICE implementations should be designed and implemented in such a way as to facilitate regulatory clearance.

2.1.3 Design Pillars About System Attributes

Security: Medical systems inherently touch human lives and private information. ICE implementations must be secure to the greatest extent possible. Security in this domain encompasses a tremendous range. The most relevant technical requirements focus on the needs for identification, authentication, and authorization of connected devices, clinical users, and patients. Information in transit and at rest must be secured with appropriate use of encryption.

The vexing problem of security is constantly evolving as new threats emerge. It is therefore futile to postpone other design and implementation work until security issues have been entirely solved. The great importance of security means our platform must not prevent it. For instance the platform architecture must be layered to allow the passage of opaque payloads by lower infrastructure layers. A platform that could not disseminate opaque data would clearly be more difficult to secure.

There is an apparent tradeoff between security and usability. Security features must not slow down or prevent urgent clinical use.

Scalability: ICE implementations must scale gracefully. A platform that enables a revolution in bedside devices must scale to support the next generation of devices. Even while building concrete prototypes with the current generation of medical devices, we must anticipate a newer generation of devices that we expect will furnish higher resolution data streams. Software simulation should be used for stress testing because adequate numbers of physical devices are unlikely to ever be available, and software simulation allows exploring the impact of next-generation devices that are not yet available. The platform, supporting current generation devices, should exhibit a great deal of underutilized capacity.

The platform also encompasses the federation of bedside ICE systems and must consider scalable approaches to that federation.

High Availability: Medical infrastructure must guarantee high availability. ICE supports the integration of multiple sources of patient data. Components that fail should be seamlessly replaced by redundant data sources or other components if they are available. Put another way, risk control measures need to take into account component malfunctions. ICE should support achieving single fault tolerance for applications.

The ICE platform must ultimately become a trusted participant in the overall clinical environment. Doing so will require the anticipation of scenarios for seamless failover, graceful degradation, etc. Implementations must prove trustworthy to find acceptance among clinicians.

Performance: Performance is another key to acceptance by the clinical community. Sluggish performance may be inconsequential in the laboratory setting but a poorly performing system in the clinic consumes a critical resource; the clinician's time. Poor performance can also encourage clinicians to marginalize the system in order to isolate the threat to their workflow. ICE implementations must support dynamic detection and reporting of performance degradation.

Generic Interface: Each component will share its data representation in common. Software shared in common among components will mediate all communication. In the ICE diagram, Equipment Interfaces, Applications, Data Logger, and External Interfaces should all share the same interface and data representation.

Forensic Data Logging: ICE implementations must create a credible log of all activity so that adverse events can be investigated in order to surface and trace root causes of faults in the distributed system. Every aspect of implementations must avoid any data pathways that may "sidestep" this logging (while balancing this with our need for scalability and security). Information known to bypass the data logger must be documented with a rationale.

2.2 Clinical Requirements

Our requirements process starts with clinical scenarios that are suggested by our clinical user community. These lead to clinical use cases and requirements, which in turn suggest technical requirements. One of our core design pillars is to include only technical requirements that are related to necessary clinical end-user requirements. We aim to support clinical application in the near-term, so non-functional requirements such as reliability and safety are key.

Figure 2.1, illustrates the general approach. Clinical focus groups [28] suggest clinical scenarios, which are captured either in person or through our prototype clinical scenario repository [1]. These scenarios then suggest clinical requirements, such as the samples shown in Figure 2.2. These clinical requirements imply technical requirements which are implemented to build a concrete system such as OpenICE [69]. We use the technical requirements to verify the implementation, the clinical requirements and design pillars to validate the implementation, document gaps, and iterate. This waterfall development style description is overly linear, and it's important to realize that design and implementation are likely to iterate rapidly.

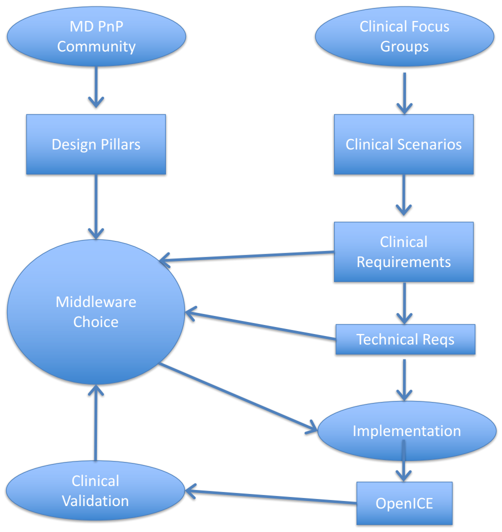


Figure 2.1: Requirements and Middleware Selection Process

Clinical scenarios may document a situation where patient outcomes or safety could be improved by the use of interoperable devices. It is vital that the set of scenarios also include situations where a technical integration failure or lack of interoperability leads to patient harm, as well as situations where interoperability leads to new hazardous situations. Scenarios can be reflect an actual or imagined sequence of events that happened, or they can be constructed from an imaginable sequence of events derived from what policies and guidelines exist to prevent. Any such scenario will suggest approaches to a solution designed to address problems illuminated by the scenario. Any one solution will by nature impose requirements on the actions taken to implement them and the tools with which the actions are taken. In this work, we concentrate on the use of clinical requirements and their influence on middleware selection, rather than the process of moving from clinical scenarios to clinical requirements or from clinical to technical requirements.

The clinical requirements primarily represent the interactions of the system, including constituent devices, with users including clinicians and the patient. Our clinical requirements have come from elicitation sessions, clinicians, hospital policies, existing documentation, ASTM F2761 Annex B, clinical care guidelines, nursing documentation, clinical specialists, incident reports, and other groups.

X-Ray / Ventilator and PCA Requirements. Consider two of the scenario sets which we have studied and implemented, X-ray / Ventilator interactions and PCA safety:

For the former, the initiating problem was shooting an X-ray of an organ during mechanical ventilation of the lungs, which introduced motion artifact into the images. Among the solutions to this single sentence scenario was to temporarily cease ventilation to take the image. An obvious clinical safety requirement for such a solution is to restart ventilation as soon as possible after acquiring the image, literally within seconds. This is generally a safe process, but if clinicians are interrupted by a situation competing for their attention, there is the possibility of forgetting to restart ventilation. This has happened, and as such represents a scenario posed by an unintended consequence of addressing a prior clinical requirement (image without motion artifact). The requirement that ventilation resume as soon as possible after the image is required can be supplemented with a requirement that the system not to anything else until ventilation resumes. Informed by a history of attention-related accidents, a requirement can be envisioned that the system do so without human intervention.

For the second, the initiating clinical problem was managing the pain experienced post-operative patients. Clinical studies had demonstrated many benefits associable with giving pain medication in smaller, more frequent doses as needed by the patient. For various reasons, providing the patient with the means to do so was more effective, suggesting the requirement that a pump capable of delivering the medication to the patient was of value. A risk to such a solution was patient overdose, and a solution was devised to limit the amount of medication a patient could self-administer. Implementation of this solution placed requirements on devices, limiting the amount of drug that can be delivered in any one hour. However, it also placed requirements on the practices of caregivers, and that once again provides the context for attention-related accidents, e.g., PCA-by-proxy [31] or misprogramming [29] [73] [54]. Fairly complex rule-based clinical practices have developed to address these situations, see for instance [21]. Again this suggests a requirement that diminishes the risk posed by human interaction with a complex system.

These scenarios illustrate the importance of clinical involvement in the requirements process. Starting from the conceptual level for both of these scenarios, other clinical questions arise driven by a sequence of events presented by a type of system that has never existed before. How do I know if and when the system is meeting its requirements, and how do I know when it is not, especially those that affect patient safety? What do I do when that happens? Does this introduce any fundamental changes to any of my practices? What about the system gives me the confidence to do so? And while it is reasonable for an engineer to propose a solution and establish requirements that addresses these and other concerns, ultimately only the end user can determine if the discerned requirements can fully address the problem in its full context. For that reason, a key aspect of our process is clinician validation of each requirement and any changes to it throughout development.

Figure 2.2 contains a selection of clinical requirements that have direct implications for middleware selection. For instance, consider SCR6 “The ICE System shall notify users when it loses connectivity with any of its components.” These clinical requirements are written from the perspective of the clinical user, who may have little or no knowledge of how the system works; they are a form of black box requirements. This requirement could be implemented in a wide variety of ways. There are no requirements stated for timing, for how the notification should happen, or for which component should do the notification. These specializations of the requirement follow from specific use cases and specific implementations. The specialization of SCR6 will be very different for an ICE implementation intended to run only an application that sends data to an offline documentation archive versus an implementation intended to support running an application controlling a closed-loop infusion of a fast-acting drug. SCR1 and SCR2 may also raise the eyebrows of those experienced in real-time systems. The closest possible match may not be a very good match at all, which is why review is required, and any deviation may throw off carefully engineered timings. It is important to remember that these requirements capture clinical needs as voiced by clinicians. They are not technical engineering requirements, and they are subject to interpretation and change in building implementations. Validation that a given implementation satisfies the clinical requirement is inherently subjective. It is our intention in compiling these that they be unambiguous and reflect clinical consensus. The clinical requirements shown in the examples are generic in the sense that they are meant to apply to all ICE systems.

1. The ICE system shall identify and display a list of external interfaces connected to it - PACS, EHR/EMR, bed-management system, third-party integrators, RTLS (Real Time Locating System) for equipment and staff etc.
2. The ICE system shall be aware of the required frequency / accuracy / reliability of the incoming data for each parameter based on clinical significance, and shall choose the closest available frequency / accuracy / reliability on the device and provide this information to the clinician for review.
3. If the device connected to the ICE system is not capable of providing the required frequency / accuracy / reliability of the incoming data for each parameter based on clinical significance, the ICE system shall choose the closest available frequency / accuracy / reliability on the device and provide this information to the clinician for review.
4. The ICE system shall PUSH alerts/alarms to the Alert Management System.
5. In the event of incorrect patient information/data going into the system, the ICE system shall provide the ability to tag the incorrect data until the issue gets resolved.
6. The ICE System shall not allow the clinician to interact with the ICE System until s/he is authenticated and authorized, unless it is an emergency situation or the new clinician identifies himself or herself.
7. The ICE System shall notify users when it loses connectivity with any of its components.
8. When the ICE system is down, local data shall be available and displayed locally on each individual device without any interruptions.
9. The ICE system shall display a compatibility error message when Device software version is incompatible.
10. The ICE system shall relate and verify the patient ID and patient location (through ADT or manually or other sources) and tag the medical device data from that location using this data.
11. When Patient A has been discharged/ moved to a different department/floor in the hospital (known through other apps/sources using ADT), the ICE system shall display a notification such as “Patient A has moved” or “Patient ID mismatch”.
12. The ICE system shall provide capability to verify if the Patient information (Patient ID/MRN, Bed Location, Patient Last Name, DoB, etc.) is the same from ALL connected devices. If there is a mismatch, the ICE system shall provide a mechanism to correct it.
13. The ICE system shall be able to display the patient’s medical record number (MRN) as it is stored in the ICE app, any connected medical devices, the patient’s wristband, and any available medication label such as a barcode.

Figure 2.2: Sample Clinical Requirements

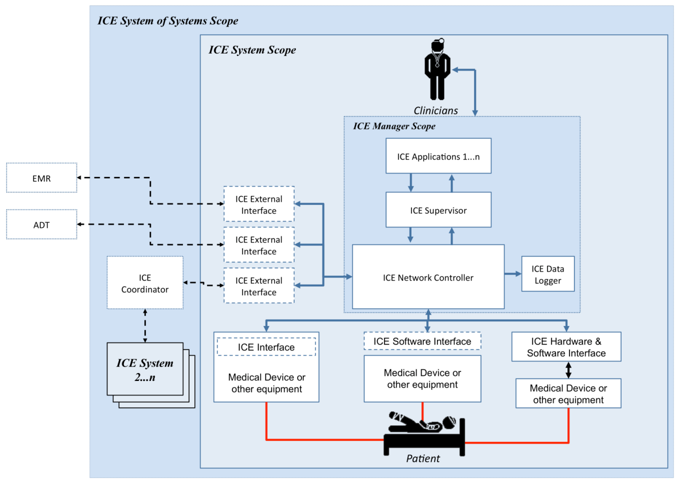


Figure 2.3: OpenICE Functional Architecture

2.3 ASTM ICE Standard

The ICE standard [16] defines an architecture for building a safe patient-centric Integrated Clinical Environment. OpenICE is the MD PnP lab's open-source implementation of the ICE standard.

2.3.1 ICE Architecture

ICE defines roles for device adapters, a network controller that mediates traffic, a supervisor capable of hosting applications, a data logger for troubleshooting and forensic analysis, and external interfaces to hospital resources such as an EHR, ADT, or pharmacy system. This architecture is illustrated in Figure 2.3.

The ICE architecture as described in ASTM F2761 was designed as a patient-centric architecture. We envisioned a typical deployment as a bedside network connecting medical devices in, for instance, an intensive care unit or operating room. This bedside network would then connect to hospital IT resources like an electronic medical record (EMR) or admit / discharge / transfer (ADT) system through the ICE External Interface. Synchronization between multiple ICE systems is achieved through an ICE Coordination for use cases such as transferring patients from an operating room to an ICU room.

Within an ICE system there is an ICE Manager and a set of Equipment Interfaces. The Equipment Interfaces connect medical devices and other equipment to the ICE system. Figure 2.3 depicts three types of ICE Equipment Interfaces: those built into medical devices where the ICE network is their native interface, those that require only a software interface, for devices that use standard network connections like Ethernet but communicate using a proprietary non-ICE protocol, and Equipment Interfaces for devices that require both hardware and software interfaces, such as devices with only a serial port connection. This last category is the most common for legacy devices. Devices connected via the ICE Equipment Interface may be medical devices or non-medical equipment, such as environmental sensors.

The ICE Manager consists of four components: the Network Controller, Data Logger, Supervisor, and set of Applications. The ICE Network Controller is responsible for connecting components together, handling discovery and message history for late-joining participants, and maintaining status information. In most ICE implementations to date, including OpenICE, some or all of these duties are assumed by a middleware. The Data Logger records low-level information about communications through the Network Controller. It is intended for debugging of ICE components (including applications running on the ICE supervisor) and forensic analysis of network errors. The ICE Supervisor provides some services required by all applications, such as patient identity management and provides a common user interface. ICE Applications implement behaviors that make use of the connected devices.

The applications are the purpose of ICE - the rest of the components exist to enable developers to create applications without having to be concerned with device communication or reimplement common services like patient identity management or logging. Applications may read and display data from devices, like a dashboard or implement physiologic closed-loop control like titrating a drug infusion from a pump based on blood pressure readings from a patient monitor.

2.4 OpenICE Implementation

Over the last 10 years we have built in our lab numerous prototype medical distributed systems [27] utilizing a variety of connectivity solutions. We started by using approaches built on web services such as SOAP and industrial systems like MODBUS to synchronize an X-Ray exposure with an anesthesia machine ventilator [5] [6]. This was followed by an infusion pump safety interlock built on a deterministic, hard real-time network implemented on custom FPGA hardware [4]. We have done extensive work on patient-controlled analgesia pumps including formal analysis of pumps [8], formal analysis of systems [7] [45] and infusion pump hazards and requirements [14] [8]. Clinical requirements and fundamental design principles have also come from our work on smart alarm systems [46], medical security analysis [12] and closed-loop control [2] [80]. In addition to publications, these systems were presented at medical conferences including HIMSS, the American Society of Anesthesiologists annual conference, the Society for Technology in Anesthesiology annual meeting, and other venues. Feedback gathered from clinicians at these venues has gone into each iteration and been included in the clinical requirements and design pillars. This body of work forms the basis for claiming the validity of the design pillars in Section 2.1.

Medical devices in hospitals and other clinical settings are not yet networked with each other. This leads to compartmentalization and siloing of information and the inability to access and use time-aligned contextually rich data for to prototype, develop, and deploy novel, life-saving algorithms. Consequently, the development of innovative solutions to improve patient safety and the quality of healthcare delivery are stifled, and the incidence of preventable adverse clinical events remains unacceptable high.

OpenICE. In response to these needs, we have developed OpenICE [69] [67], an open source implementation of the Integrated Clinical Environment (ICE) standard (ASTM 2761-09(2013)), and made it freely available on GitHub. The platform consists of software device adapters for medical devices, standards-based publish/subscribe middleware, and demonstration applications. Supported medical devices include anesthesia machines, ventilators, and patient monitors from vendors including Philips, Drager, and GE. Applications can be built on this platform to implement smart alarms, physiologic closed-loop control algorithms, data visualization, and clinical research data collection.

OpenICE is an open-source software project from the Medical Device Plug-and-Play Interoperability Program (MD PnP) at Massachusetts General Hospital. It leverages much of the program's work over the last decade to support four distinct sets of users: use case demonstrations, clinical adoption, regulatory science, and commercial adoption.

OpenICE is a collection of software that implements the ICE standard. Written primarily in Java, OpenICE is capable of running on many different kinds of hardware. In the MD PnP lab, equipment interfaces usually run on small, single-board computers that are physically attached to the back of the medical devices. These interfaces can run equally well on the same laptop running the supervisor and applications or on a server in another room. The only hardware and operating system requirements are support for Java and a physical interface, in most cases a serial port, that matches the device being interfaced.

OpenICE started as a demonstration platform that was first presented in November of 2012. It became an open source project in 2013 and was first publicly presented at a conference at the Society for Technology in Anesthesia meeting in Orlando, Florida in January of 2014. OpenICE benefitted from the involvement of a large team of collaborators through our NIH funded Quantum project and from input from developers of MDCF, Continua, and the OR net project. It has also led to the development of OpenICE-lite[37], a lightweight implementation of the architecture, as well as a related dongle-based approach[15].

DDS Middleware. Safe interoperability requires that participants on the network all play by the same rules. We use the Data Distribution Service (DDS) publish/subscribe middleware, an OMG standard [30]. DDS was chosen as the middleware for this prototype because it supports the expression of a wide range of quality of service parameters, allowing us to support a variety of clinical scenarios suggested by our user community. DDS is a publish/subscribe middleware, where applications and devices can announce that they can provide or are interested in receiving particular pieces of data. Network participants publish updates to data as it becomes available and the middleware matches publishers to subscribers. Data from apps can be indistinguishable from data from physical medical devices, enabling the development and sharing of sophisticated data processing apps that may generate data for use by other system components. OpenICE uses a community licensed version of the RTI implementation of DDS.

Matching publishers to subscribers requires that all of the participants use a common set of terms. For this work, a subset of the ISO/IEEE 11073-10101 nomenclature was used. This allows for components (applications or devices) to be semantically interoperable. Using this approach, it is our intent that a device manufacturer will be able to produce a device with an electronic interface that will work with any ICE application and any ICE application will work with any device that provides the data elements that the application requires.

The data distribution service is not a message-oriented middleware. Instead, it serves as a delegate for the task of managing the distributed state of a system composed of loosely coupled, heterogeneous components around a shared data model representing the broader system state. DDS serves to synchronize system state between the nodes of a distributed system. Working at this level of abstraction allows us to concentrate on defining the data and quality of service (QoS) requirements on the data rather than the mechanisms for transferring the data and ensuring that QoS parameters are met. Applications are provided with Readers and Writers that maintain, on behalf of the application, a coherent view of the current state of homogeneous collections of state information. This allows the associated system data model, represented in IDL, to be inherently normalized.

DDS allows data sources, whether medical equipment interfaces or other data providers, to clearly define their mechanism for sharing updated state information with other participants. The source program's sole responsibility is to write the updated sample of a particular facet of system state to a Writer. The Writer delegate then takes on responsibility for communicating that new sample to other participants under a system of constraints described in a robust set of quality of service parameters.

A Reader's responsibilities include maintaining a coherent view of the state communicated by Writers under the terms of its own Quality of Service settings as well as reconciling Reader QoS with Writer QoS to ensure compatibility. Applications can create a Reader delegate with a specified topic and type then access the current state of that facet of the system by reading samples from its own Reader delegate whenever it wants to examine state. Readers also provides a variety of highly granular semaphores for communicating events in its own duty cycle with other threads outside of the DDS pool. For instance condition variables can be configured to awaken an external (to DDS) thread when new unread samples are available or when a writer has failed to pair with the reader because of incompatible Quality of Service parameters.

Demo Applications. Initially, we wrote demo applications that included dedicated device interfaces running on a linux pc with multiple serial ports or USB to serial adapters. We found that this was difficult to scale, code reuse was challenging, and we experienced many problems with unexpected timing jitter when using multiple USB to serial adapters. Still, we were able to build successful implementations of X-Ray and ventilator synchronization [7] [6] [5] and PCA [8] [65] use cases. The next generation of demos was built as a reaction to the timing challenges and incorporated a dynamic, real-time ethernet interface using custom hardware including FPGAs to ensure real-time synchronization of state information. We used this as the basis for another implementation of the PCA system [4] . The custom boards were expensive and susceptible to static damage, and many of our use cases did not require hard real-time. Our third generation system, the basis for our current OpenICE platform, uses readily available, low cost, single-board computers such as Raspberry Pi or BeagleBone. To synchronize data across nodes of the system, we use Data Distribution Service (DDS), an OMG standard [30].

Higher level entities in OpenICE are built using the DDS primitives described above. When a user downloads OpenICE and runs it, they see a screen showing currently connected devices and available applications. We chose to present the user with devices and applications because we have found through collaborating on use case implementation with a variety of clinical users that these are the components that most users care about. All of the other components in the ICE standard are present. Because they exist to enable applications and device connectivity and are meant to be used by application developers rather than end users, these components are not visible in the user interface.

Figure 2.4 shows how the ICE architecture maps onto an OpenICE installation in the MD PnP lab. The Supervisor screen shows a list of available applications and devices, a variety of medical devices are connected using BeagleBone single board computers (shown in Figure 2.5) as ICE Equipment Interfaces, and the patient is represented by a cart loaded with an electronic patient simulator, mechanical lung simulator, and, because this is a spontaneously breathing patient with a nasal cannula for monitoring end-tidal CO_2, a mannikin head.

In this implementation, device adapter hardware and applications are cleanly sepa-

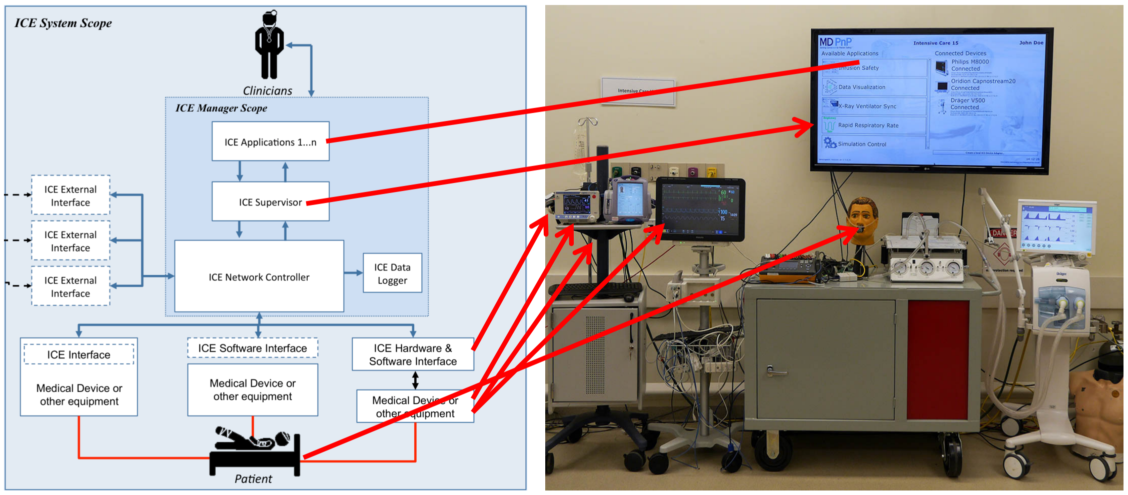


Figure 2.4: Mapping ICE Architecture to OpenICE Demo Implementation

rated. This installation is typical in that it include severals BeagleBones running device interfaces and a supervisor computer running applications that use device data and issue commands to the devices. The role of the Network Coordinator is subsumed by DDS and the OpenICE functionality that enables discovery and other parts of device lifecycle management.

ICE Equipment Interfaces in this installation take two forms. Most of the equipment interfaces involve OpenICE device adapter code running on a BeagleBone. We use BeagleBones because they are a convenient size that can easily be attached to the back of many medical devices, the ARM architecture is well supported, and can be easily expanded with a serial port. Most medical devices use a serial port for data communications; even patient monitors that have ethernet ports often require that ethernet is used to communicate with a central station leaving the serial port as the only means of connecting them to an OpenICE system. OpenICE also supports sophisticated device simulation with many built in device simulators that can be used to emulate patient monitors, ventilators, infu-

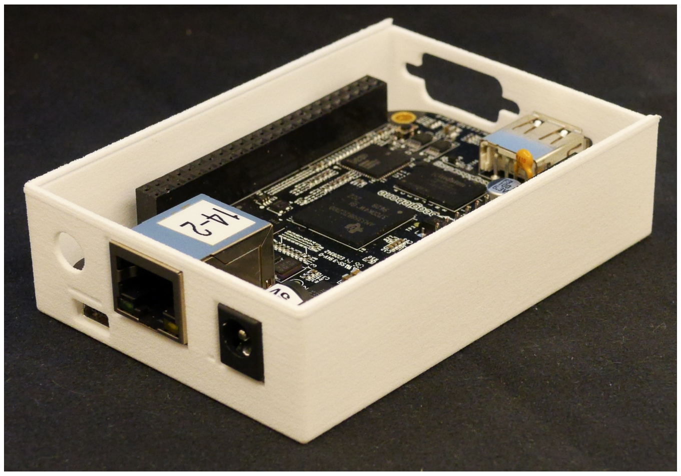


Figure 2.5: A BeagleBone single board computer in a custom 3D printed case

sion pumps, and other devices along with simulated patient variability for users who wish to test scenarios with many devices and for application development when real medical devices are unavailable.

Other applications bundled with OpenICE include a PCA safety application based on our work on infusion safety and several data visualization applications that illustrate the advantages of decoupling data analysis and display from the proprietary protocols and terminology of current devices. Once data is represented in a standardized way, applications can be written to use it independent of what particular device it comes from. Of course, if an application developer feels they need to know the specific origins of a data element that information is always available in the metadata.

Data Logger. The system shown here does not include a forensic data logger that logs all network traffic. We feel that in most cases third party tools such as tcpdump or wireshark adequately fill this role. It does, however, include several tools for recording selected data sets. This is the most often request feature from our clinical users, who

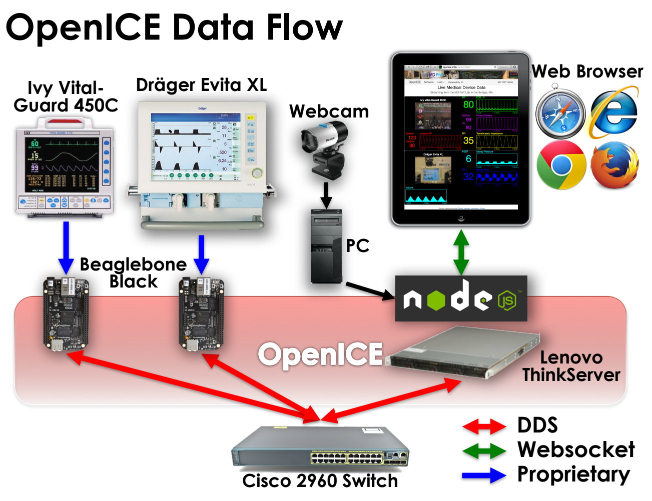


Figure 2.6: OpenICE Data Flow

struggle to find tools for recording normalized, time-synchronized data simultaneously from multiple devices. The Data Recorder tool can send data, represented using the IEEE 11073 terminology, to SQL and MongoDB databases as well as recording locally as a simple set of text files. Data elements in the OpenICE system are time stamped by device interfaces when the interface translates the data elements from the device's proprietary terminology to our standards-based terminology. These timestamps are based on a common timebase that is synchronized across devices using NTP. Device adapters also pass along the device's timestamps on data elements, though in practice we find that timestamps from devices are not always reliable as the device clocks are not set accurately in clinical use. For ICE External Interfaces, OpenICE also includes an HL7 export tool that can stream data using the HL7 standard, and our demos have included streaming data using HL7 FHIR.

We have performed several experiments with sharing OpenICE data through the web. Shown in Figure 2.6, these require running an application on the OpenICE network that subscribes to the relevant information and passes it along to a node application running on a server, which in turn uses websockets to send it to the client. This allows the clients to be any web browser that supports javascript and websockets. These experiments have been useful for illustrating the kinds of applications we want interoperability to enable. They serve as inspiration for developing the interoperability protocols that will allow for privacy preserving and secure exchange of real-time healthcare data.

The current OpenICE implementation is not secure. There is no encryption of data in transit or at rest, and no attempt is made to authenticate or authorize applications, devices, or users. We have left OpenICE insecure partly to ease installation and use of it as a research platform, and partly because best practices for security in this space are still being defined. Platform level security requirements focus on user and device authentication and authorization and encryption of data in transmission. Platform level security is necessary but not sufficient. Application level security will need to be defined and built on a per-application basis because in the OpenICE architecture only the application developers know the intended use and use environment of the assembled system [12].

2.5 Other Relevant Interoperability Standards

Plug and play is a familiar concept from such common standards as USB and web services. PnP systems share may concepts, though details vary across implementations. Universal Plug and Play combines many of the best parts of these systems, and is described here to illustrate the state of the art in the consumer device realm. Two PnP systems specific to medical devices are also described: the Integrated Clinical Environment (ICE), Medical

Device Coordination Framework (MDCF). Related standards work from Continua, HL7, and IEEE 11073 is introduced in the context of these PnP systems.

2.5.1 IEEE 11073

IEEE 11073 is a family of medical device interface standards that have been under development for almost 30 years. These standards are intended to address every layer of the ISO OSI 7-layer model for communication between a pair of compliant devices. IEEE 11073 grew out of ANSI/IEEE 1073, which in turn grew out of the Medical Information Bus (MIB).

11073 was originally intended for use with high-acuity point-of-care devices like ventilators, infusion pumps, and multi-parameter monitors. This classic version of 11073 is referred to as 11073-PoC for point of care.

In this work, we use one part of the 11073 family of standards, 11073-10101, as a common set of terminology. We translate from proprietary data representations into this terminology set and use these terms for writing specifications and safety properties. More specifically, we use the 11073 nomenclature and some of the domain information model (DIM). We have tried to decouple the medical data representation from the data needed for other purposes at lower levels of the protocol stack. 11073 is intended for point to point links and the data representation includes information which is captured elsewhere in this implementation. For instance, a full 11073 representation of heart rate would include the the frequency at which heart rate is being updated. In this system, that information is part of the quality of service parameters for the channel conveying heart rate. The information is still there, but is not part of the data representation.

2.5.2 Medical Device Coordination Framework

The Medical Device Coordination Framework is designed to be an open platform for medical device integration. It is architected around a messaging-oriented-middleware, using Java Messaging Service (JMS) to implement a publish-subscribe framework. Programing in MDCF is model-based, with an abstract system design being used to automatically generate a set of channel descriptions.

The MDCF developers identified these requirements for middleware [45]:

- Flexible, dynamic information flow (frequently needing privacy)
- Heterogeneous systems, mechanisms, and needs
- Many listeners and many sources
- Time-critical, scalable performance

On top of JMS, MDCF adds a device connection manager, maintenance console, monitoring console, clinician console, and a scenario manager.

The device connection manager "verifies that the connecting device is in a database of approved devices and associated drivers (which provide API descriptions for interacting with each device".

The scenario manager "manages the life-cycle of scenario script executions including acquisition of devices needed in the script, creation of components and JMS channels to realize inter-component communication, and tear-down of components and channels after script execution."

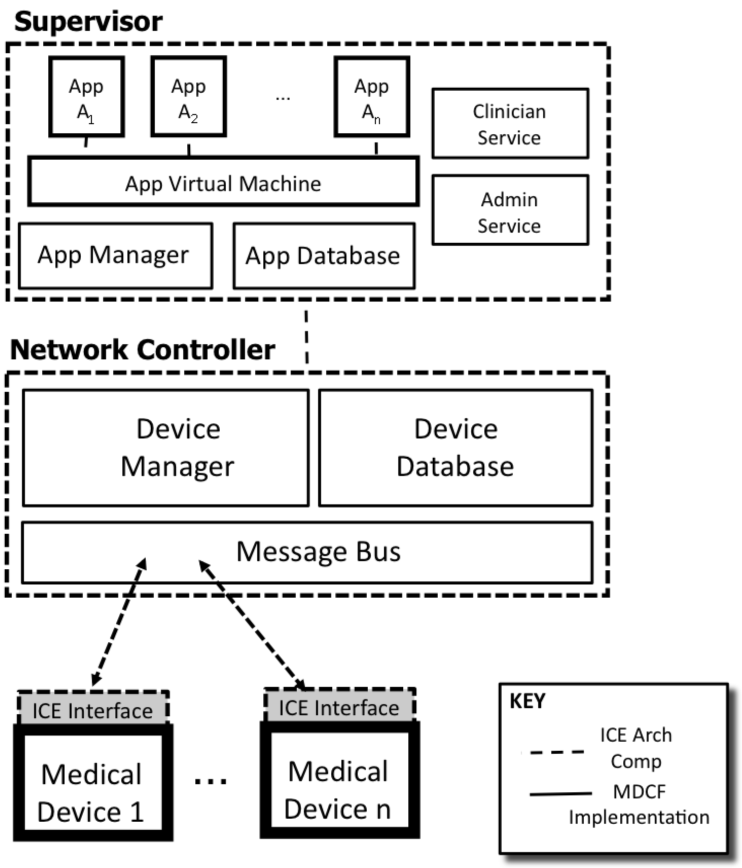


Figure 2.7: MDCF ICE

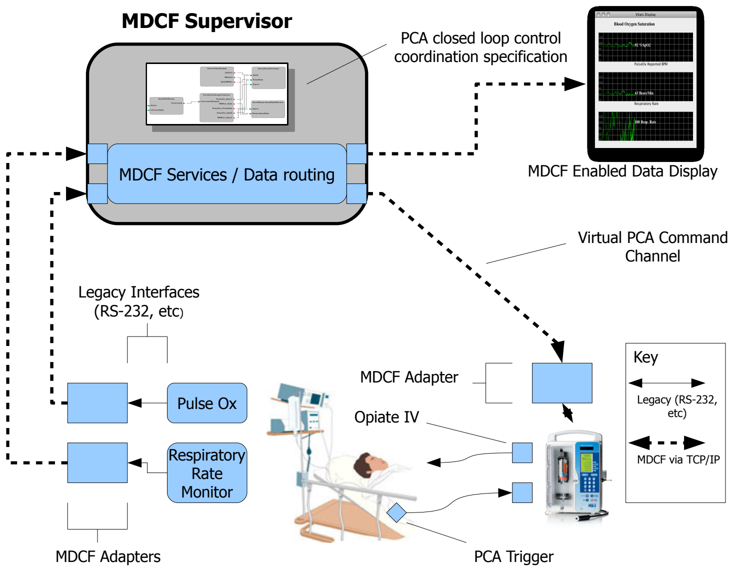

MDCF Supervisor
PCA closed loop control coordination specification
MDCF Services / Data routing
MDCF Enabled Data Display
Virtual PCA Command Channel
Legacy Interfaces (RS-232, etc)
Pulse Ox
Respiratory Rate Monitor
MDCF Adapters
MDCF Adapter
Opiate IV
PCA Trigger
Key
Legacy (RS-232, etc)
MDCF via TCP/IP

Figure 2.8: MDCF PCA

Programming is done in Cadena, using a component interface editor and a system scenario editor. Experiments were done with three types of messages: event notification, HL7, and DICOM. Experiments with varying numbers of data producers and consumers found that latencies we “within allowable bounds”. Persistent messaging was turned off due to unacceptable delays in the message database.

MDCF demonstrates the usefulness of middleware in medical device interoperability. illustrates role of tool support for model-based programming code generation conceptual framework

MDCF has also led to related work on separation architectures for combining high and low criticality devices, including internet of things devices [19], and virtual integration of devices [49].

2.5.3 ICEMAN

ICEMAN is a system for medical device interoperability developed at Draper Laboratories following their early work on what became the ICE standard. ICEMAN is best described in Hofmann’s masters thesis [34]. ICEMAN is a high-level architecture very much in line with UPnP and other PnP systems.

ICEMAN components are:

- Workflow scripts, rules, and models. Rules are safety and best practices measures constraining ICEMAN actions. Device models represent functionality of the device and physiological models capture relations between devices and the patient.

- Device interfaces specify the format and ordering of messages, not low-level implementation. RS-232, CAN, USB, Ethernet, etc. are all feasible.

- Semantic libraries: ICEMAN says that devices should support several and specify which one they use.
- Human interface
- Data logger

Discovery is done by broadcast to a fixed, globally known address. An interesting feature of ICEMAN is that legacy device protocols are described using ANTLR and TAP to describe the context-free grammar of the language. System components that wish to communicate with legacy devices synthesize a parser on-the-fly following these descriptions. The authors intended future work to support transfer functions that would relate Actions to Metrics, allowing patient models and richer descriptions of devices.

2.5.4 Universal Plug and Play

Universal Plug and Play (UPnP) is a system built of common protocols that allows networked devices to discover one another's presence, exchange descriptions of capabilities, and make use of offered services. It is most used in the home entertainment industry, with implementations included in devices such as home routers, televisions, and stereo receivers. UPnP is developed and promoted by the UPnP forum and standardized as ISO/IEC 29341.

UPnP uses existing open communication standards implemented by most devices connected to the internet. This means that little or no special driver software is necessary. User interfaces are provided through a web browser. UPnP provides an excellent example of a plug and play system and a good introduction to the set of services necessary for any PnP system. These services are:

- Addressing
- Discovery
- Description
- Control
- Event Notification
- Presentation

Other PnP systems will group and name these services differently, but they are common necessary components that will recur in the other systems described in this Chapter.

Addressing. Addressing is the means by which devices receive a network address when they are connected to the system. UPnP uses IP addressing with addresses provided using DHCP.

Discovery. Once a device has an address, it needs to know what other devices are on the network. This process of exploring the local network is called discovery. UPnP uses Simple Service Discovery Protocol (SSDP). SSDP basically uses UDP multicast to a well-known address to announce the presence of devices and the availability of services.

Description. The description consists of the steps necessary for a device to announce and describe its capabilities to other devices in the system. These capabilities include data that the device could provide to other devices or actions that the device could take at the command of other components. In UPnP, devices send a URL during discovery that points to an XML document listing device capabilities.

Control. In most PnP systems, control is just another kind of data exchange. In UPnP, control is done using SOAP, a web-services protocol for information exchange commonly used for remote procedure call style interactions.

Event Notification. In general, event notification does not require a separate communication technology. Because most communication in a UPnP network is initiated by the party wishing to receive information, a separate pathway is set up for pushing events. This follows the publish/subscribe paradigm; UPnP implements this following the General Event Notification Architecture (GENA). Devices that wish to receive updates listed as available in another device's Description can subscribe to updates from the other device using GENA.

Presentation. UPnP calls the provision of a user interface presentation. UIs are done through a web browser, which means that devices must implement a web server through which they make available an interface for external configuration and presentation of information. This is familiar to many from home network configuration, where routers publish a webpage through which the end user can configure their network.

Figure 2.9 shows an example UPnP Message Sequence.

As a combination of DHCP, SSDP, HTTP, SOAP, GENA and others, UPnP is a melange of useful internet protocols. By reusing existing communication standards, the

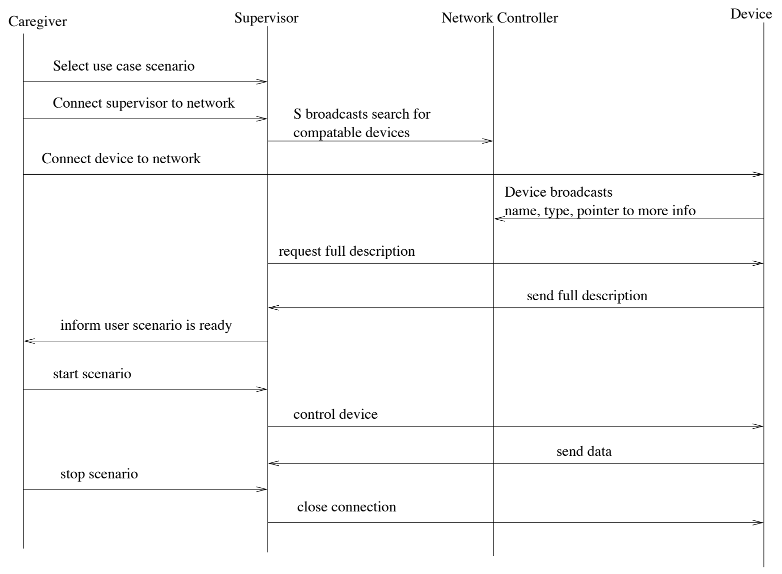


Figure 2.9: Sample UPnP Message Sequence

framers of UPnP needed only to specify how to use the existing protocols and to specify the data format for Description. Each device's description then specifies the data format for the elements that device supports.

UPnP has several shortcomings limiting its applicability to the medical domain. Most critically, there is no notion of authentication or authorization. Essentially, there is no notion of security; UPnP is intended for use within a small home network consisting of a few trusted devices. Unfortunately, this has not been how UPnP devices have been deployed. Many real-world vulnerabilities have been found because UPnP devices are commonly attached to the Internet without a protective firewall blocking UPnP services.

2.6 Discussion

In this chapter, we have surveyed related systems (Section 2.5), laid out clinical requirements (Section 2.2) and design pillars for interoperable systems (Section 2.1), and discussed the OpenICE implementation of the ASTM 2761-09 standard that will be used for some of the case study work in later chapters (Section 2.4).

This Chapter discusses some general principles and design pillars for interoperable medical device systems and examines five plug-and play systems: UPnP, ASTM 2761-09 ICE, 11073, MDCF, and ICEMAN. These share some common attributes, components and lifecycle stages but have important differences in scope and intended use. UPnP is intended for consumer applications like audio-visual equipment, not safety-critical applications. ICEMAN has a larger scope, and was a precursor to what became the ASTM ICE standard, which is discussed in detail in Section 2.3. The ASTM ICE standard, ISO 11073, and MDCF overlap in scope; they are all intended as platforms for connecting medical devices, but have important differences in implementation and middleware choices.

Limitations. This Section discusses limitations, gaps, and future work related to the standards and reference implementations of interoperability platforms related to and used in this work. The OpenICE platform is covered in detail in Section 2.4. OpenICE is used as an example system throughout, and in the case studies in Chapter 6. OpenICE is intended to address clinical needs as captured in the clinical requirements covered in Section 2.2. The architecture described in Chapter 4 and the case studies in Chapter 6 are based on the ASTM ICE standard and the OpenICE implementation. Limitations of the standard and implementation carry through to the case studies.

One important limitation of the systems we have discussed is that they have all been developed as prototype or research systems, not as part of a regulated medical device. In order to build implementations that can be used as part of medical devices, substantial effort will be needed to reimplement these platforms under a quality system. Quality systems [74] are used in medical device manufacturing to track development efforts and give traceability from hazards through to the final implementation. Device manufacturers often build prototypes as research activities or as proofs of concept and then reimplement them under a quality system if they will become part of a device. OpenICE is built in the spirit of these research prototypes with the assumption that it will be reimplemented by manufacturers before incorporation into a regulated device. FDA guidance on interoperability [77] points to a direction for designing devices intended to be used a component of an interoperable system, and the standards and systems discussed point to means of implementing such systems but much regulatory and implementation work remains to be done before commercial implementations will be available.

Another limitation of the discussed systems is that, with the exception of 11073, they do not offer a comprehensive way to manage patient identity. Device identity is generally considered to be solved using unique device identifiers, and the pairing of devices to control systems and association with patients is left as a manual, and often labor-intensive, step. Manual association is sufficient for prototypes or demonstration systems, but it is time-consuming to manually associate individual devices with patients and applications. In the future, we will need more automated ways of making these associations and verifying them once they are made.

Gap Analysis. We have discussed five systems for interoperability, but even in the medical domain there are other systems and standards we have left out of this discussion.

There are a multitude of medical device standards from ISO, AAMI, and other standards development organizations that could be relevant to aspects of this problem space. Most major medical device manufacturers have in-house programs to connect their devices, and in some cases devices from other manufacturers. Some of these are interoperable, but most have not been described in detail in publications.

Future Work. The ASTM ICE standard specifies that devices should send a device model that describes the device's capabilities. In OpenICE, we implemented this requirement using an Interface Definition Language (IDL) description of the data types and terms chosen from the ISO 11073-10101 [36] terminology set. OpenICE does not include the systems for checking device models against device requirements or the system safety properties against the whole system as described in Chapters 4 and 5.

Medical systems rarely operate in isolation, and this is also true of the bedside patient care networks considered here. These bedside networks will need to interact with other hospital information systems including pharmacy, medical records, billing, imaging, and others. At present, standardized ways of interacting with these hospital systems are still nascent. As they mature, bedside systems will need to interact with them to provide patient care.

Chapter 3

Clinical Application Modeling Language

3.1 Clinical Application Modeling Language

Medical applications and workflows inherently involve many devices and clinicians sharing information. In trying to model these workflows, we face the challenge of creating a model that is detailed enough to capture the behavior of interest and allow checking properties while also being small enough to allow for checking the properties in a reasonable amount of time. Choosing a modeling system requires balancing these constraints. For this work, we have developed the Clinical Application Modeling Language to describe systems of communicating extended finite state machines. Our toolset allows building models, translating them into the systems used by several model checking tools, and generating Java code.

We have developed the Clinical Application Modeling Language (CAML) as a way of representing Clinical Application Scripts (CAS). CAML is built on our previous work on extended finite state machines (EFSMs) including our EFSMtool toolset for manipulating, checking, and translating EFSMs into the input languages of several popular model checking tools. In this work, we use the UPPAAL model checker and describe the translation into this tool's input language in Section 3.4.

The EFSM language is designed to be a simple way of representing state machines. State machines consist of states and transitions and these are also the core of the EFSM language. The grammar for EFSMs is presented in figure 3.3. A sample EFSM appears in figure 3.7 and the same system is shown graphically in figure 3.2. An EFSM consists of a set of states connected by transitions. Transitions must have a guard condition and may also be tagged with an action which is performed when the transition is taken. Guards and actions are conditions and assignments on a set of variables. Variables may be declared as input, local, or output, and boolean and bounded integer types are supported.

The design of EFSMs and the CAML language is influenced by Communicating Sequential Processes [33] and timed autonoma [3], heritage it shares with the UPPAAL tool and the Java communication libraries used for code generation.

3.2 CAML's semantics

3.2.1 Extended Finite State Machine (EFSM)

Definition 3.2.1. An EFSM (Extended Finite State Machine) E is a tuple $\langle D, F\rangle$ where D is the global declaration and F is the finite state machine associated with E. The finite state machine F is a tuple of the form $\langle N_F, n_0, n_f, T_F, V_F, I_F\rangle$, where

```
CFR{
   States:
      TestDonationWithApprovedTest,
      HaveRecordOfPreviousTest,
      TestWithSupplementalTest,
      DoNotShipOrUseRejectDonor,
      UseDonation;

   InitialState: TestDonationWithApprovedTest;

   Final:
      UseDonation,
      DoNotShipOrUseRejectDonor;

   InputVars:
      bint[0 .. 2] ScreeningOutcome=0,
      boolean Previous=False,
      bint[0 .. 2] SuppOutcome=0,
      bint[0 .. 2] PrevSuppOutcome=0;

   OutputVars:
      boolean sample_uses_all=False,
      boolean donor_uses=false,
      bint[0 .. 5] label=0;

Transition: From TestDonationWithApprovedTest to HaveRecordOfPreviousTest
   when ScreeningOutcome==2;
Transition: From TestDonationWithApprovedTest to UseDonation
   when ScreeningOutcome==1 do sample_uses_all=True;
Transition: From HaveRecordOfPreviousTest to UseDonation
   when Previous==True and PrevSuppOutcome == 1 do sample_uses_all=True;
Transition: From HaveRecordOfPreviousTest to TestWithSupplementalTest
   when Previous==False or PrevSuppOutcome == 2;
Transition: From TestWithSupplementalTest to UseDonation
   when SuppOutcome==1 do sample_uses_all=true;
Transition: From TestWithSupplementalTest to DoNotShipOrUseRejectDonor
   when SuppOutcome==2 do label = 4, donor_uses=false;
}
```

Figure 3.1: CFR: An Example EFSM

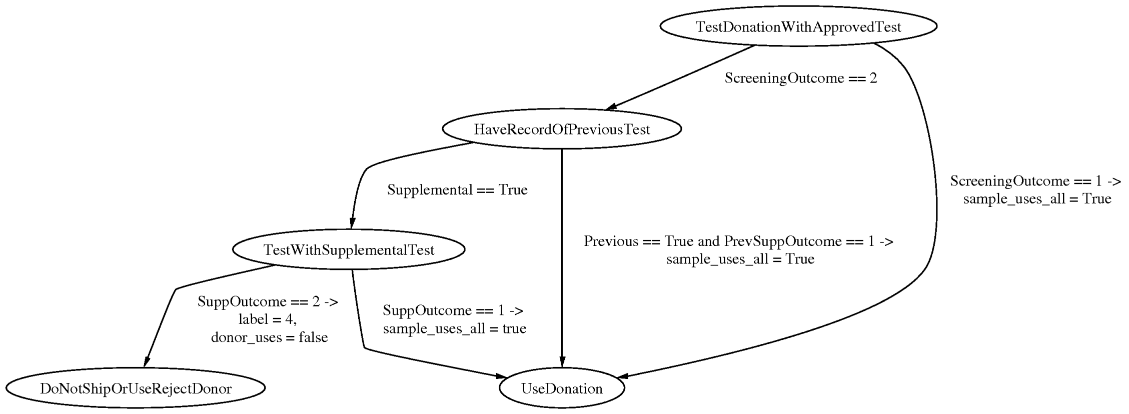

TestDonationWithApprovedTest
ScreeningOutcome == 2
HaveRecordOfPreviousTest
ScreeningOutcome == 1 -> sample_uses_all = True
Supplemental == True
TestWithSupplementalTest
Previous == True and PrevSuppOutcome == 1 -> sample_uses_all = True
SuppOutcome == 2 -> label = 4, donor_uses = false
SuppOutcome == 1 -> sample_uses_all = true
DoNotShipOrUseRejectDonor
UseDonation

Figure 3.2: Graph of CFR

System	::=	(“System” ID ⟨LBRKT⟩ DECLARATIONS CEFSM ⟨RBRKT⟩)+
Declarations	::=	(Channels)?
		(GlobalVars)?
CEFSM	::=	(States)?
		(InitialState)?
		(Final)?
		(InputVars)?
		(LocalVars)?
		(OutputVars)?
		(Transition)+
Transition	::=	⟨ FROM⟩ ID
		⟨TO⟩ ID
		⟨WHEN⟩ Condition
		(⟨DO⟩ (Assignment (‘,’ Assignment)* ‘;’)+)? “;;”
States	::=	⟨STATES⟩ ID (‘,’ ID)* “;;”
Final	::=	⟨FINAL⟩ ID (‘,’ ID)* “;;”
InitialState	::=	⟨INITSTATE⟩ ID “;;”
Channels	::=	⟨CHANNELS⟩ ID ‘,’ “SYNC” | “ASYNC” ‘,’ ‘R’ | ‘W’ | ‘B’
		(‘;’ ID ‘,’ “SYNC” | “ASYNC” ‘,’ ‘R’ | ‘W’ | ‘B’)* “;;”
InputVars	::=	⟨INPUTVARS⟩ VarDef (‘,’ VarDef)* ‘;;’
OutputVars	::=	⟨OUTPUTVARS⟩ VarDef (‘,’ VarDef)* ‘;;’
LocalVars	::=	⟨LOCALVARS⟩ VarDef (‘,’ VarDef)* ‘;;’
GlobalVars	::=	⟨GLOBALVARS⟩ VarDef ‘,’ ‘R’ | ‘B’ (‘;’ VarDef ‘,’ ‘R’ | ‘B’)* ‘;;’
VarDef	::=	(⟨BOOLEAN⟩ ID | Assignment)
		| (⟨BINT⟩ ‘[’ ⟨NUM⟩ ⟨DOTDOT⟩ ⟨NUM⟩ ‘]’ (ID | Assignment))
		| ID
Assignment	::=	(ID ⟨EQ⟩ (ID | ⟨NUM⟩) (⟨OP⟩ (ID | ⟨NUM⟩))*) |
		(ID ⟨COMMOP⟩ (ID | ⟨NUM⟩ | ⟨BOOLEAN⟩))
Condition	::=	ID ((⟨OP⟩ ID ⟨OP⟩ ⟨NUM⟩)+ | (⟨OP⟩ ID)+)*
ID	::=	(“ ‘ ” ⟨SimpleID⟩ “ ’ ”) | (“ “ ” ⟨SimpleID⟩ (⟨SimpleID⟩)+ “ ” ”)

Figure 3.3: EFSM Grammar

- N_F is a set of locations,
- $n_0 \in N_F$ is the initial location,
- $n_f \subseteq N_F$ is the set of final locations,
- T_F is a set of transitions,
- $V_F = V_L \cup V_G^F$ is a set of typed finite domain variables where V_L is the set of local variables, $V_G^F \subseteq V_G$ is the set of global variables used by F where V_G is the set of all the global variables in this system of EFSMs. Each global variable has exactly one EFSM which can write data to it, but it may have many EFSMs reading data from it.
- I_F is a set of initial constraints over the variable set V_F.

All the above mentioned sets are assumed to be finite.

The global declaration D is a tuple of the form $\langle G_D^F \rangle$. An EFSM may either read a variable or both read and write it. The set G_D^F is defined as $G_D^F = \{\langle v, X \rangle | (\forall v \in V_G^F)$ s.t. $(X = R|B)\}$. $X = R$ indicates that F is a reader for v and $X = B$ indicates that F is both a reader and a writer for v.

We now define a valuation function for the variable set V_F. Let, $dom(x)$ denote the domain of the variable x, $\forall x \in V_F$. Further let, Val be a valuation function defined over the variable set V_F such that $\forall x \in V_F, Val(x) \in dom(x)$. We let $Val(V_F)$ denote the set which has one element for each variable in V_F and that element is equal to the valuation of the corresponding variable in V_F.

Given a valuation $V_i = Val(V_F)$ and a location $n \in N_F$ we define a state s of the EFSM as a tuple $\langle n, V_i \rangle$. Therefore, the set of initial states S_0 for the EFSM F can be defined as a set of tuples of the form $\langle n_0, Val(V_F) \rangle$ such that $Val(V_F)$ satisfies the initial constraints set I_F.

Let $\mathcal{E}(V_F)$ be the set of general expressions on the variable set V_F and $\mathcal{B}(V_F)$ be the subset of boolean expressions over the variable set V_F. An expression $\mathcal{E}(V_F)$ consists of general expressions using a set of operators and the variable set V_F. The set of boolean expressions $\mathcal{B}(V_F)$ is a subset of general expressions such that evaluation of this expression will result in a boolean value. An assignment over the variable set V_F is a statement of the form $x = \mathcal{E}(V_F)$ where $x \in V_F$.

A transition $t \in T_F$ is of the form $\langle n, g, \alpha, n' \rangle$, where $n \in N_F$ is the source location, $g \in \mathcal{B}(V_F)$ is the guard condition over the variable set V_F, α is a set of assignments over variables in V_F, and $n' \in N_F$ is the target location. α has exactly one assignment of the form $x = \alpha^x$ for each variable $x \in V_F$, where $\alpha^x \in \mathcal{E}(V_F)$. For readability any variable which remains unchanged through the assignment set α (i.e. $\alpha^x = x$) is not listed in α.

$I_F \in \mathcal{B}(V_F)$ is the set of initial constraints. There is one constraint per local and output variable. Input variables are not give initial values.

An EFSM E is represented graphically as follows :

- Every location $n \in N_F$ is represented by a circle. The initial location n_0 is represented by an incoming transition with no source location.

- Every transition $t \in T_F$ such that $t = \langle n, g, \alpha, n' \rangle$ is represented by a directed edge from the source location n to the destination location n'. The guard conditions in g are written within braces alongside the directed edge and the actions in α are written without any braces alongside the directed edge. For example a guard condition $a > 5$ will be written as "$(a > 5)$" alongside the edge and an action $a = 2$ will be written as "$a = 2$" alongside the edge. g is a set of guards all of which are checked simultaneously. Hence all the conditions in g are separated with a $\wedge$. α is a list of action groups which are executed sequentially. Each action group consists of a set of actions and they are assumed to be simultaneous. Action groups are separated by ';', whereas actions within a group are separated by ','. Further there will be a $\Longrightarrow$ sign between the guard and the action of a transition.

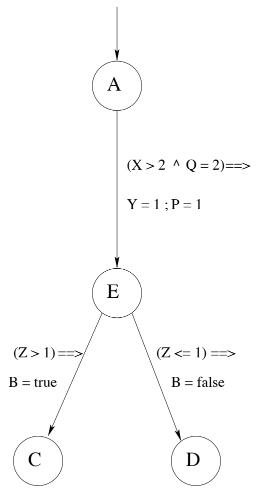


Figure 3.4: Extended Finite State Machine

3.2.2 Communicating Extended Finite State Machine (CEFSM)

Definition 3.2.2. A CEFSM (Communicating Extended Finite State Machine) is an extension of EFSM with communication channels. CEFSM E is a tuple $\langle D, F\rangle$ where D is the global declaration and F is the finite state machine associated with E. The finite state machine F is a tuple of the form $\langle N_F, n_0, n_f, T_F, C_F, V_F, I_F \rangle$, where

- N_F, n_0, n_f, T_F, V_F and I_F are defined as before and
- C_F is a finite set of input/output communication channels used in this CEFSM.

The global declaration D is a tuple of the form $\langle C_D^F, G_D^F \rangle$. The set G_D^F is as defined for EFSMs. Channels are defined with a name and a designation of synchronous or asynchronous. Let C denote the set $\{\langle \mathit{name}, \mathit{SYNC} | \mathit{ASYNC} \rangle |$ for all the channels in the system $\}$ where *name* is the global name of the communication channel and SYNC indicates the channel is synchronous and ASYNC indicates it is asynchronous.

There can be more than one use of the same communication channel in an CEFSM at different transitions. Each use of a communication channel $c \in C$ identified as $c_f \in C_F$ can be represented by a tuple of the form $\langle \mathit{name}, t, v, IO \rangle$ where

- *name* refers to the name of the channel,
- $t \in T_F$ refers to the transition linked to this use of the channel,
- v refers to the variable $\in V_F$ whose value will either be output or input through this channel, depending on whether it is a reader or writer. v can be empty in which case the channel is used for signalling and
- IO indicates whether this channel is an input or an output channel. IO will contain the value "*input*" or "*output*" accordingly.

The set C_D^F is a set of channels associated with F. C_D^F can be defined as $C_D^F = \{\langle c, X \rangle | (\forall u \in C_F) \text{ s.t. } ((c \in C) \wedge (\Pi_1(c) = \Pi_1(u)) \wedge (X = R|W|B))\}$ where $\Pi_1(z)$ is the projection of the first element of the tuple z. $X = R$ and $X = B$ have the same interpretation as the one for global variables. $X = W$ indicates that c is a writer but not a reader for this channel. A synchronous channel can be either written to or read from by a CEFSM but not both. Hence a synchronous channel can never have the property $X = B$. The use u of the channel must be consistent with the properties of the channel declared in D.

Further, each communication channel has certain global properties which can be described as follows

- **Synchronous actions:** $SYNC|ASYNC$ indicates whether the channel is a synchronous channel or an asynchronous one. This property is specified in the declaration section of every CEFSM F where this channel is used, and they must be declared the same in each CEFSM definition they appear in.
- **Consumability:** This is a property associated with the buffers of the asynchronous channel. We assume that all buffers are consumable. This means that writers can overwrite data in the buffer and readers will empty the buffer on reading data from it. Further readers will block if the buffer is empty,
- **Buffer Location:** All buffers are assumed to be located on the reader side of a communication channel,
- **Buffer Size:** All buffers are assumed to be of size 1 and
- **Multiple readers:** We further assume that each channel is a one to many communication channel in correspondence with a broadcast. Each channel has exactly

one writer but can have one or many readers.

We can now define properties that would determine whether a given CEFSM is a reader, writer or both for a global variable/communication channel. Let x be a global variable and F be a CEFSM. If any action set α of any transition $t \in T_F$ in F has an assignment of the form $x = \alpha^x$ then F is a writer for this global variable. If an expression of the form $\mathcal{E}(V)$ such that $x \in V$ is used in either the guard or action of any transition of F then F is a reader for this variable. Similarly, if there is a channel usage $c_f \in C_F$ such that it is an input channel, then F becomes a reader for this channel and if it is an output channel then F is a writer for this channel.

Graphical Representation. A CEFSM E is represented graphically as follows:

- Every location $n \in N_F$ is represented by a circle. The initial location n_0 is represented by an incoming transition with no source location.
- Every transition $t \in T_F$ such that $t = \langle n, g, \alpha, n' \rangle$ is represented by a directed edge from the source location n to the destination location n'. Other properties are similar to EFSMs.
- Every use of the communication channel $c_f \in C_F$ in the CEFSM such that $c_f = \langle name, t, v, IO \rangle$ is represented by the following notations depending on the properties of the communication channel
  - The channel is **asynchronous, consuming, reader side buffer with single writer and single or multiple readers :** If this use of the channel is as a

writer then the transition t will have the action "$cn!!v$" appended to its action list. Otherwise it will have the action "$cn??v$" appended,

– The channel is **synchronous with single writer and single or multiple readers :** If this use of the channel is as a writer then the transition t will have the action "$cn!v$" appended to its action list else it will have the action "$cn?v$" appended,

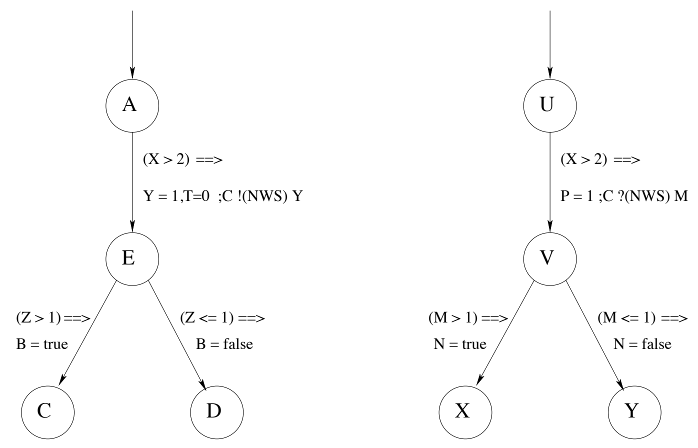


Figure 3.5: Communicating Extended Finite State Machine

3.3 EFSM and CEFSM Execution

In modeling the execution of an EFSM or a CEFSM, a state is represented by a pair $\langle n, Val(V_F)\rangle$, where n is a location, $Val(V_F)$ is the valuation of the variables in V_F. The execution of an EFSM/ CEFSM starts at a state $\langle n_0, V_0\rangle$, where n_0 is the initial location and V_0 is consistent with I_F (i.e. $I_F(V_0) = true$). $I_F(V_0)$ is an evaluation of I_F for the valuation V_0 of the variables.

A transition $\langle n, g, \alpha, n'\rangle$ can be taken from the current state $\langle n, V_i\rangle$ only if the current valuation $V_i = Val(V_F)$ satisfies the guard condition ($g(V_i) = true$). The effect of taking the transition $\langle n, g, \alpha, n' \rangle$ from a state $\langle n, V_i \rangle$ is a state $\langle n', V_j \rangle$, where V_j is the new valuation resulting from the execution of the assignment statements specified in the set α. $V_j = \alpha(V_i)$, $\alpha(V_i)$ is an execution of all the assignment statements in α for the valuation V_i of the variables. This execution must satisfy the sequentiality and simultaneity constraints of the assignment statements.

3.3.1 EFSM execution

Definition 3.3.1. An execution of an EFSM E $= \langle N_F, n_0, n_f, T_F, V_F, I_F \rangle$ is a finite or infinite sequence of the form

$$s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 \ldots$$

where

each $t_i \in T_F$ and $s_i = \langle n_i, V_i \rangle$ satisfies the following conditions :

1. Initial condition:

   $s_0 = \langle n_0, V_0 \rangle$ where $V_0 = Val(V_F)$ such that $I_F(V_0) = true$.

2. Succession Constraint:

   $\langle n_1, V_1 \rangle \vdash \langle n_2, V_2 \rangle$ iff $\exists \langle n_1, g_1, \alpha, n_2 \rangle \in T_F$ such that $g_1(V_1)$ is $true$ and $V_2 = \alpha(V_1)$.

3.3.2 CEFSM execution

We define two types of communications channels for CEFSMs:

Definition 3.3.2. Let F be an CEFSM $= \langle N_F, n_0, n_f, T_F, C_F, V_F, I_F \rangle$ and let it contain some communication channels which are asynchronous, consuming, reader side buffer with

1 writer and ≥ 1 readers.

An execution of F is a finite or infinite sequence of the form

$$s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 ...$$

where

each $t_i \in T_F$ and $s_i = \langle n_i, V_i \rangle$ satisfies the following conditions :

1. Initial condition:

   $s_0 = \langle n_0, V_0 \rangle$ where $V_0 = Val(V_F)$ such that $I_F(V_0) = true$.

2. Succession Constraint:

   The succession constraint is a specification of the form $\langle n_i, V_i \rangle \vdash \langle n_j, V_j \rangle$. This constraint is determined depending on whether the actions of that particular transition have any I/O commands to be executed (asynchronous I/O commands are specified either with a "!!" or with a "??" in the graphical representation).

   - **Actions do not have any I/O commands:** $\langle n_i, V_i \rangle \vdash \langle n_j, V_j \rangle$ iff $\exists \langle n_i, g_1, \alpha, n_j \rangle \in T_F$ such that $g_1(V_i)$ is *true* and $V_j = \alpha(V_i)$.

   - **Actions have I/O commands:** $\langle n_i, V_i \rangle \vdash \langle n_j, V_j \rangle$ iff $\exists \langle n_i, g_1, \alpha, n_j \rangle \in T_F$ such that $g_1(V_i)$ is *true* and $V_j = \alpha(V_i)$ and the following conditions must be satisfied.

     Since the transition between n_i and n_j has I/O commands in its actions, let $c_f = \langle name, t, v, IO \rangle$ be the corresponding use. Now, t will have action "$cn!!v$" or "$cn??v$" depending on whether the channel use is for input or output. Further let *cn.buffer* be the name of the channel buffer(s) for this reader/writer.

– Succession Constraint for Writer:

  The guard for the transition is enabled whenever $g_1(V_i)$ is *true*. Its traversal will result in the assignment $cn.buffer = v$ and the value of v remains unchanged.

– Succession Constraint for Readers:

  The guard for the transition is enabled when the buffer associated with this reader has data in it. Its traversal results in the assignment $v = cn.buffer$ and since the channel is consuming the buffer *cn.buffer* is emptied. Non-determinism is introduced if multiple readers are enabled simultaneously.

Definition 3.3.3. Let F be an CEFSM = $\langle N_F, n_0, n_f, T_F, C_F, V_F, I_F \rangle$ and let it contain some communication channels which are synchronous with 1 writer and ≥ 1 readers.

Let e_w be the single writer CEFSM and $e_R = \{e_{r_1}, e_{r_2}, ..., e_{r_n}\}$ be the set of CEFSM readers associated with this channel.

An execution of F is a finite or infinite sequence of the form

$$s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 ...$$

where

each $t_i \in T_F$ and $s_i = \langle n_i, V_i \rangle$ satisfies the following conditions :

1. Initial condition:

   $s_0 = \langle n_0, V_0 \rangle$ where $V_0 = Val(V_F)$ such that $I_F(V_0) = true$.

2. Succession Constraint:

   The succession constraint is a specification of the form $\langle n_i, V_i \rangle \vdash \langle n_j, V_j \rangle$. This constraint is determined depending on whether the actions of that particular transition have any I/O commands to be executed (I/O commands are specified either with a "!" or with a "?" in the graphical representation).

   - **Actions do not have any I/O commands:** $\langle n_i, V_i \rangle \vdash \langle n_j, V_j \rangle$ iff $\exists \langle n_i, g_1, \alpha, n_j \rangle \in T_F$ such that $g_1(V_i)$ is *true* and $V_j = \alpha(V_i)$.

   - **Actions have I/O commands:** $\langle n_i, V_i \rangle \vdash \langle n_j, V_j \rangle$ iff $\exists \langle n_i, g_1, \alpha, n_j \rangle \in T_F$ such that $g_1(V_i)$ is *true* and $V_j = \alpha(V_i)$ and the following conditions must be satisfied.

     Since the transition between n_i and n_j has I/O commands in its actions, let $c_f = \langle name, t, v, IO \rangle$ be the corresponding use. Now, t will have action "$cn!v$" or "$cn?v$" depending on whether the channel use is for input or output. The following conditions are required to be met depending on whether the current CEFSM is a reader or writer.

     – Succession Constraint for Writer:

       The guard for the transition is enabled when

       $\forall e_{r_i} \in e_R$, the current state of e_{r_i} is $\langle n_k, V_k \rangle$ and $\exists \langle n_k, g_k, \alpha_k, n \rangle \in T_{e_{r_i}}$ such that $g_k(V_k)$ is *true*, and α_k contains a read "$cn?v_1$" on channel cn.

     – Succession Constraint for Readers:

       The guard for the transition is enabled when

       $\forall e_{r_i} \in e_R$, the current state of e_{r_i} is $\langle n_k, V_k \rangle$ and $\exists \langle n_k, g_k, \alpha_k, n \rangle \in T_{e_{r_i}}$ such that $g_k(V_k)$ is $true$, and α_k contains a read "$cn?v'$" on channel cn and the current state of e_w is $\langle n_x, V_x\rangle$ and $\exists\langle n_x, g_x, \alpha_x, n'\rangle \in T_{e_w}$ s.t. $g_x(V_x)$ is $true$, and α_x contains a write "$cn!v_1$" on channel cn.

3.3.3 Parallel Composition of CEFSMs

The parallel composition of CEFSMs allows the concurrent simulation of multiple CEFSMs.

Definition 3.3.4. Given two CEFSMs, $E_1 = \langle D_1, F_1\rangle$ and $E_2 = \langle D_2, F_2\rangle$ where $F_1 = \langle N_{F_1}, n_{0_1}, n_{f_1}, T_{F_1}, C_{F_1}, V_{F_1}, I_{F_1}\rangle$ and $F_2 = \langle N_{F_2}, n_{0_2}, n_{f_2}, T_{F_2}, C_{F_2}, V_{F_2}, I_{F_2}\rangle$, the parallel composition $E_1||E_2$ is a CEFSM $E = \langle D, F\rangle$, such that $F = \langle N_F, n_0, n_f, T_F, C_F, V_F, I_F\rangle$, where

- $N_F = N_{F_1} \times N_{F_2}$,
- $n_0 = (n_{0_1}, n_{0_2})$,
- $n_f = n_{f_1} \times n_{f_2}$,
- T_F is given as follows : for every pair of tuples $\langle n_1, g_1, \alpha_1, n_1'\rangle \in T_{F_1}$ and $\langle n_2, g_2, \alpha_2, n_2'\rangle \in T_{F_2}$ such that these transitions do not have any use of communication channels, T_F includes the transition $\langle (n_1, n_2), g_1 \wedge g_2, \alpha_1 \cup \alpha_2, (n_1', n_2')\rangle$.
- $V_F = V_{F_1} \cup V_{F_2}$
- $I_F = I_{F_1} \cup I_{F_2}$
- The parallel composition for communication channels is given as follows :

Synchronous communication channels. For every pair of transitions such that one of them writes on a channel of the form $c = \langle cn, t, v, \text{“}output\text{”}\rangle$ where $c \in C_{F_1}$ or C_{F_2} and the other has a set of reads on the same channel $c_1 = \langle cn, t1, v1, \text{“}input\text{”}\rangle, \cdots c_m = \langle cn, t1, vm, \text{“}input\text{”}\rangle$ where $c_1, \cdots c_m \in C_{F_2}$ (resp C_{F_1}) such that the source and destination locations for t are t_1 and t_2 and for $t1$ are $t1_1$ and $t1_2$, we append the simultaneous assignments $v1 = v, \cdots vm = v$ to the action list of the transition between locations $(t_1, t1_1)$ and $(t_2, t1_2)$ in the composed CEFSM. This action list will also contain the channel write command $c = \langle cn, t', v, \text{“}output\text{”}\rangle$. The assignments have to be simultaneous to reflect the synchronous nature of the channel.

For every pair of transitions such that one of them has a set of reads on the channel of the form $c_1 = \langle cn, t1, v1, \text{“}input\text{”}\rangle, \cdots c_m = \langle cn, t1, vm, \text{“}input\text{”}\rangle$ where $c_1 \cdots c_m \in C_{F_1}$ and the other has a set of reads of the form $c'_1 = \langle cn, t2, v'1, \text{“}input\text{”}\rangle$, $\cdots c'_k = \langle cn, t2, v'k, \text{“}input\text{”}\rangle$ where $c'_1 \cdots c'_k \in C_{F_2}$ such that the source and destination locations for $t1$ are $t1_1$ and $t1_2$ and for $t2$ are $t2_1$ and $t2_2$, we append the channel read commands $c_1 = \langle cn, t, v1, \text{“}input\text{”}\rangle, \cdots c_m = \langle cn, t, vm, \text{“}input\text{”}\rangle, c'_1 = \langle cn, t, v'1, \text{“}input\text{”}\rangle, \cdots c'_k = \langle cn, t, v'k, \text{“}input\text{”}\rangle$ to the action list of the transition between locations $(t1_1, t2_1)$ and $(t1_2, t2_2)$ in the composed CEFSM. These reads must be simultaneous actions.

If only one of the CEFSMs is a writer or a reader to the channel then we do not change the channel command in any way. But if one of the CEFSMs is a reader and the other a writer or if both are readers then any composition must conform to the above listed rules. Further all transitions with synchronous channel commands in these CEFSMs cannot be composed with transitions which do not have those commands.

Asynchronous communication channels. For every pair of transitions such that one of them has a write on a channel of the form $c = \langle cn, t, v, \text{``}output\text{''}\rangle$ where $c \in C_{F_1}$ or $C_{F_2}$ and the other has a set of reads on the same channel of the form $c_1 = \langle cn, t1, v1, \text{``}input\text{''}\rangle, \cdots c_m = \langle cn, t1, vm, \text{``}input\text{''}\rangle$ where $c_1, \cdots c_m \in C_{F_2}$ (resp. C_{F_1}) such that the source and destination locations for t are t_1 and t_2 and for $t1$ are $t1_1$ and $t1_2$, we append the simultaneous assignments $v1 = v, \cdots vm = v$ to the action list of the transition between locations $(t_1, t1_1)$ and $(t_2, t1_2)$ in the composed CEFSM. Further this action list will also contain the channel write command $c = \langle cn, t', v, \text{``}output\text{''}\rangle$. The assignments have to be simultaneous because this case reflects the synchronous use of an asynchronous channel.

For every pair of transitions such that one of them has a set of reads on the channel of the form $c_1 = \langle cn, t1, v1, \text{``}input\text{''}\rangle, \cdots c_m = \langle cn, t1, vm, \text{``}input\text{''}\rangle$ where $c_1 \cdots c_m \in C_{F_1}$ and the other has a set of reads of the form $c'_1 = \langle cn, t2, v'1, \text{``}input\text{''}\rangle$, $\cdots c'_k = \langle cn, t2, v'k, \text{``}input\text{''}\rangle$ where $c'_1 \cdots c'_k \in C_{F_2}$ such that the source and destination locations for $t1$ are $t1_1$ and $t1_2$ and for $t2$ are $t2_1$ and $t2_2$, we append the channel read commands $c_1 = \langle cn, t, v1, \text{``}input\text{''}\rangle, \cdots c_m = \langle cn, t, vm, \text{``}input\text{''}\rangle, c'_1 = \langle cn, t, v'1, \text{``}input\text{''}\rangle, \cdots c'_k = \langle cn, t, v'k, \text{``}input\text{''}\rangle$ to the action list of the transition between locations $(t1_1, t2_1)$ and $(t1_2, t2_2)$ in the composed CEFSM.

For every pair of transitions where one has a use of the channel of the form $c_1 =$

$\langle cn, t1, v1, \text{“}input\text{”}\rangle$ or $c_2 = \langle cn, t2, v2, \text{“}output\text{”}\rangle$ and the other does not have any use of this channel, we append the channel command c_1 or c_2 respectively to the action list of the combined transition. This reflects the asynchronous nature of the channel.

If only one of the CEFSMs is a writer or a reader to the channel then we do not change the channel command in any way. But if one of the CEFSMs is a reader and the other a writer or if both are readers then any composition must conform to the above listed rules.

The new declaration D for the CEFSM E will be defined based on D_1 and D_2. Let $D_1 = \langle C_1, G_1\rangle$ and $D_2 = \langle C_2, G_2\rangle$. Now $D = \langle C, G\rangle$ can be defined as follows :

- Let $v = \Pi_1(g)$ where $g \in G_1 \cup G_2$ be the first projection of g. This projection will give the name of the associated global variable. Now $\forall v$ s.t. $v = \Pi_1(g)$ where $g \in G_1 \wedge v \neq \Pi_1(g')\forall g' \in G_2$ or $\forall v$ s.t. $v = \Pi_1(g)$ where $g \in G_2 \wedge v \neq \Pi_1(g')\forall g' \in G_1$, we add the tuple g to G. Further, $\forall g_1, g_2$ where $g_1 \in G_1, g_2 \in G_2$ and $\Pi_1(g_1) = \Pi_1(g_2)$ the following cases occur

  - If $\Pi_2(g_1) = B$ or $\Pi_2(g_2) = B$ then we add a tuple $\langle \Pi_1(g_1), B\rangle$ to G
  - If $\Pi_2(g_1) = R$ and $\Pi_2(g_2) = R$ then we add a tuple $\langle \Pi_1(g_1), R\rangle$ to G

- Let $cn = \Pi_1(c)$ where $c \in C_1 \cup C_2$ be the first projection of c. This projection will give the name of the channel along with its properties. Also the second projection $\Pi_2(c)$ will give information as to whether this channel is used for input or output. Two cases arise depending on whether the channel is synchronous or asynchronous

- Channel is synchronous : $\forall cn$ s.t. $cn = \Pi_1(c)$ where $c \in C_1 \wedge cn \neq \Pi_1(c')\forall c' \in C_2$ or $\forall cn$ s.t. $cn = \Pi_1(c)$ where $c \in C_2 \wedge c \neq \Pi_1(c')\forall c' \in C_1$, we add the tuple c to C. Further, $\forall c_1, c_2$ where $c_1 \in C_1, c_2 \in C_2$ if $\Pi_1(c_1) = \Pi_1(c_2)$ then the following cases occur
  * If $\Pi_2(c_1) = R$ and $\Pi_2(c_2) = R$ then we add a tuple $\langle \Pi_1(c_1), R\rangle$ to C
  * If $\Pi_2(c_1) = W$ and $\Pi_2(c_2) = R$ or $\Pi_2(c_1) = R$ and $\Pi_2(c_2) = W$ then we add a tuple $\langle \Pi_1(c_1), W\rangle$ to C
- Channel is asynchronous : Since in an asynchronous channel we just form the cross product of the states, the channel declaration section is formed using rules similar to the global variable declaration section with an additional rule for $X = W$. Now $\forall cn$ s.t. $cn = \Pi_1(c)$ where $c \in C_1 \wedge c \neq Pi_1(c')\forall c' \in C_2$ or $\forall cn$ s.t. $cn = \Pi_1(c)$ where $c \in C_2 \wedge c \neq Pi_1(c')\forall c' \in C_1$, we add the tuple c to C. Further, $\forall c_1, c_2$ where $c_1 \in C_1, c_2 \in C_2$ if $\Pi_1(c_1) = \Pi_1(c_2)$ the following cases occur
  * If $\Pi_2(c_1) = B$ or $\Pi_2(c_2) = B$ then we add a tuple $\langle \Pi_1(c_1), B\rangle$ to C
  * If $\Pi_2(c_1) = R$ and $\Pi_2(c_2) = R$ then we add a tuple $\langle \Pi_1(c_1), R\rangle$ to C
  * If $\Pi_2(c_1) = W$ and $\Pi_2(c_2) = R$ or $\Pi_2(c_1) = R$ and $\Pi_2(c_2) = W$ then we add a tuple $\langle \Pi_1(c_1), B\rangle$ to C

EFSMtool supports CAML models using channels following the above semantics. A channel $c \in C$ is of the form $\langle \{\mathit{writers}\}, \{\mathit{writerpriorities}\}$,
$\{\mathit{buffnames}\}, \mathit{name}, \mathit{sync}, \mathit{buffsize}, \mathit{consumable}, \mathit{bufflocwriter}, \{\mathit{readers}\}$,
$\{\mathit{readerpriorities}\}\rangle$ where,

- $\{writers\}$ is the set of writers for the channel. There must be at least one writer for any channel.
- $\{$*writerpriorities*$\}$ is used to assign priority to the writers. This is used only when there are multiple writers and a single buffer is on the reader side. Here the writer with the highest priority will always be able to overwrite data written to the buffer by lower priority writers whereas a lower priority writer will not be able to overwrite data written to the buffer by a higher priority writer.
- $\{$*buffnames*$\}$ is a set of names of the buffers used by the channel. These are variable names for the FSMs and each *buffname* is of the form $\langle V_1, V_2 \rangle$ where $V_1, V_2 \in V_l$ and V_1 is the output variable from the writer and V_2 is the input variable for the reader.
- *name* is the name of the channel
- *sync* is true if the channel is synchronous and false otherwise
- *buffsize* contains the value of the size of each buffer used in the channel
- *consumable* is true if the buffer is consumable and false otherwise
- *bufflocwriter* is true if the buffer is located at the writer side of the channel and false otherwise
- $\{readers\}$ is the set of readers for the channel. There must be atleast one reader for any channel .
- $\{$*readerpriorities*$\}$ is used to assign priority to the readers. This is used only when there are multiple readers and the buffer is on the writer side. Here the reader with

a higher priority will always be able to consume data written to the buffer incase more than one readers are ready to read buffer data at the same time.

EFSMtool supports these types of communications patterns:

- **Asynchronous, non-consuming, writer side buffer with single writer and single or multiple readers :** This channel can be used for clock signals or timers. The clock keeps writing or updating time into the buffer. Any process which needs the current time can read from the buffer and move ahead. The writer in this case is the clock and readers are all processes in the system which use the timer. There is no need for synchronization between these processes and the buffer is non-consuming because all reader processes must be able to read the timer value from the buffer independent of other processes.

- **Asynchronous, consuming, reader side buffer with single writer and single or multiple readers :** This channel will be used when there is a single FSM which is a procedure whose output is being used by one or more other processes. In this case, the single writer writes its output into the individual buffers of all the readers and goes ahead with its processing. The readers read the procedure output as and when they arrive at that state independent of other readers in the system. There is no need for synchronization but the buffer is required to be consuming to avoid overwriting of procedure output in the case of loops. The procedure would now be blocked if the buffer is not empty for any of the readers and it can proceed only after all the readers have consumed the buffer data.

- **Synchronous with single writer and single or multiple readers :** Syncrhonous communication channels are required when we want to force synchronization across different independent state machines.

- **Asynchronous, consuming, reader side buffer with multiple writers, writer priorities and a single reader :** This channel is required for systems where more than one potential writers can write to a single channel with a single buffer which is read by a single reader. We assign priorities to the writers so that in case of a conflict only the highest priority writer is able to write to the buffer. Further, if the buffer is full then only a writer whose priority is higher than the one whose data is there in the buffer can overwrite the buffer. The reader can then asynchronously read data from the buffer. If a lower priority writer wants to write data to the buffer and the buffer is full then it cannot overwrite the data and will be forced to ignore its data. This communication channel can thus result in loss of data and hence must be used only when loss of data is acceptable in the system.

- **Asynchronous, consuming, writer side buffer with single writer and multiple readers with reader priorities :** This channel is similar to the one above but with readers and writers swapped. Now, we have a single writer but there are more than one readers. Further readers are assigned priorities and they are used to resolve conflicts in the case of multiple readers reading the channel at the same time. The writer blocks if the buffer is full and any single reader (either the one that reaches the channel first or the one with highest priority among all waiting for input from the channel) can consume the buffer data.

Semantic preprocessing. Let E be an CEFSM = $\langle N_E, n_0, T_E, C_E, V_E, I_E \rangle$. Every use of the communication channel $c_i \in C_E$ such that $c_i = \langle cn, t, v, IO, PTY \rangle$ is represented by two dummy locations n_1 and n_2 and a set of transitions ct between them as indicated in previous section. This is a semantic preprocessing step where we convert the communication channel use associated with a particular location n into its corresponding set of dummy locations and the set of transitions between them. Further, different properties of the communication channel will entail different transitions between the dummy locations with specific guards and actions as listed below :

- The channel is **asynchronous, non-consuming, writer side buffer with single writer and single or multiple readers :** For each use of this communication channel there will be a single transition between the two dummy locations n_1 and n_2 for both the reader as well as the writer. If the channel is used for output then the action associated with the single transition ct will be "$cn!(NWS)v$". If it is used for input then the guard associated with the single transition ct will be "$cn?(NWS)v$",
- The channel is **asynchronous, consuming, reader side buffer with single writer and single or multiple readers :** For each use of this communication channel as well both the reader and the writer have a single transition between the dummy locations. If the channel is used for output then the guard associated with ct will be *"all buffers empty check"* and the action for the transition will be "$cn!(NRS)v$". If it is used for input then the guard associated with ct will be "$cn?(NRS)v$",
- The channel is **synchronous with single writer and single or multiple readers**

: For each use of this communication channel again there will be a single transition between the dummy locations for both readers and writers. If the channel is used for output then the guard associated with ct will be "$cn!v$". If it is used for input then the guard associated with ct will be "$cn?v$". This is required to ensure that the writer and all the readers of the channel synchronize before using the channel,

- The channel is **asynchronous, consuming, reader side buffer with multiple writers, writer priorities and a single reader :** For each use of this communication system, there will be two transitions ct_1 and ct_2 for the writers and a single transition ct for the reader. If the channel is used for output then

  - Transition ct_1 will have a guard *"PTY lower than the one in buffer"* and an empty action. This will be required to skip the output if the priority of the writer is lower.
  - Transition ct_2 will have a guard *"PTY higher than the one in buffer or empty buffer"* and the action associated with this transition would be "$cn!(CRMS)v$".

  If it is used for input then the guard associated with ct will be "$cn?(CRMS)v$",

- The channel is **asynchronous, consuming, writer side buffer with single writer and multiple readers with reader priorities :** For each use of this communication channel there will be a single transition for the writer and 2 transitions for all the readers. If the channel is used for output then the guard associated with ct will be *"buffer empty check"* and the action for the transition would be "$cn!(CWSM)v$" . If it is used for input then

- Transition ct_1 will have a guard *"PTY lower than other ready readers"* and the action associated will be empty
- Transition ct_2 will have a guard *"PTY highest among all ready readers"* and $cn?(CWSM)v$"

3.4 Translating from CAML to UPPAAL

The EFSM toolset contains a translator which can convert a CAML model into the input language for UPPAAL. This translation imposes some restrictions on the CAML system; in particular, probabilistic transitions are not allowed and the translator supports only synchronous channels, so any value passing must take place through shared variables.

EFSMs and communicating sets of EFSMs in a CAML system, as described in section 3.3, and when limited to eliminate probabilistic transitions and only use synchronous channels, map onto structures in the language of UPPAAL [47], [17]. Translating models is thus a fairly straightforward rewriting into the correct input format for the tool.

The full translation of an example EFSM into UPPAAL is included in Appendix B.

UPPAAL uses an XML representation of the automata which we can create directly from the EFSM. This representation begins with a boilerplate block which states the XML version used and points to the online document type definition.

Any channels used in the system are defined in the first block.

This is followed by a section containing a definition of an automata in a "template" block. An UPPAAL system may contain many automata, which are specified one after the other in this section. Each automata contains the following parts:

First come the variable definitions. Variables in the UPPAAL model are bounded integers. Boolean variables from the EFSM model are translated to integers with a range from 0 (False) to 1 (True). All variables must have initial values defined.

Location definitions come next. There is a location definition for each state of the EFSM. This assigns a unique ID to each state. State names in UPPAAL may contain only letters and numbers, so other characters are replaced. For instance, "&" becomes "And".

Transitions are given in the next block. Each transition has a source and target field (using the IDs given in the previous section). Transitions may also have guards and actions. Synchronization actions are included in this section, though none appear in this example. The translator supports only synchronization channels, not channels which pass values. This is because UPPAAL only supports synchronization channels.

The definition of this automata ends with </template>. If there was another automata in this system, another template block would follow.

Finally, the system is given a name, and the document is closed.

3.5 Java code generation from CAML

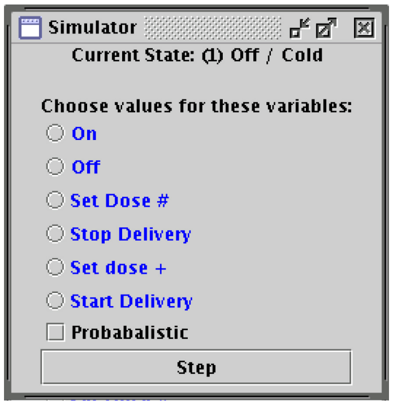
Code generation is used to create an executable computer program from a CAML model. Generated code forms a key piece of the X-Ray / Ventilator synchronization application discussed in Section 6.1 and several other system implementations. Code generation makes creating a program from the model faster and easier because it reduces the amount of code the implementor needs to write. It also helps to increase confidence that the code correctly mirrors the logic of the model.

The job of code generation from a model including communicating processes into the Java programming language makes use of the Java Communicating Sequential Processes (JCSP) library [81]. This library implements a combination of Hoare's Communicating Sequential Processes (CSP) [33] and Milner's π-calculus [60]. JCSP implements these communication types on top of the Java language's processes and interprocess communication. The EFSMtool code generator creates processes for each EFSM and uses communication patterns corresponding to CEFSM communication channels.

EFSMtool includes a code generator for single EFSMs that creates an interactive application that prompts the user for input, as well as a code generator that will create a standalone Java program for single EFSMs or a set of communicating EFSMs in a CAML model. The simulator created from an EFSM model simulates walking through the model. At each step of the state machine, the user is prompted to pick values for the input variables required to make the next step. If a step does not require any input variables, that is, if all transitions out of the current state have guards which use only local variables, then the step is taken automatically.

When the simulator is started, the screen in Figure 3.6 appears. The user can press the buttons on this screen to set the values of the system's input variables, then use the "step" button to cause the simulator to step forward to the next place where it reads an input variable. This interactive simulation continues until the system reaches a stuck state.

Alternatively, the user can press the probabilistic button at any time. This causes the simulator to pick a transition randomly, set the input variables associated with that



Simulator

Current State: (1) Off / Cold

Choose values for these variables:

- ( ) On
- ( ) Off
- ( ) Set Dose #
- ( ) Stop Delivery
- ( ) Set dose +
- ( ) Start Delivery
- [ ] Probabalistic

Step

Figure 3.6: Simulator User Interface

transition to values which make the guard true, and step once. In order to find values for the input variables which enable the transition to be taken, the simulator translates the guard into a satisfiability problem and uses the built in SAT solver to find a satisfying assignment. If no satisfying assignment can be found, the simulator shows an error message to the user and stays in the current state.

Transitions are picked randomly for the probabilistic simulation unless probabilities are provided as part of the input. The CSV parser of the EFSM toolset supports tagging transitions with probabilities. If these are provided, they will be used in the probabilistic simulator.

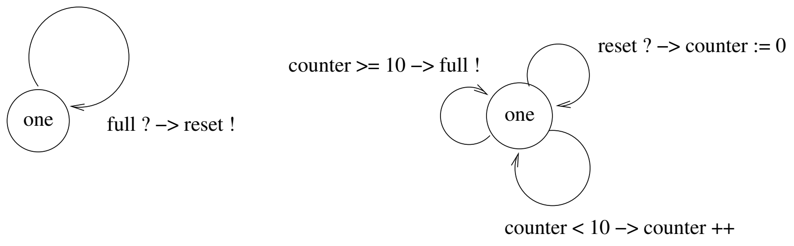


Figure 3.7: Codegeneration Example

EFSMtool's “JCSP Codegeneration” tool is a method for generating Java code from an CAML model. The model may contain one or more state machines, which can communicate using one-to-one synchronization channels.

The code generator has the additional constraint that writes to a synchronization channel may occur only in the action of a transition and reads may occur only in the guard of a transition.

This technique generates a Java thread for each state machine and adds a new main thread to initialize and start the system.

```
import jcsp.lang.*;
import jcsp.plugNplay.Printer;

class CGtop {

  public static void main (String[] args) {

    final One2OneChannel full = new One2OneChannel ();
    final One2OneChannel reset = new One2OneChannel ();
    final One2OneChannel out = new One2OneChannel ();

    new Parallel (
      new CSProcess[] {
          new CounterThread(full, reset, out),
          new ResetThread(full, reset, out),
          new Printer (out, "CGtop ==> ", "\n")
      }
    ).run ();
  }

}
```

```
import jcsp.lang.*;

public class ResetThread implements CSProcess {

    private final AltingChannel full;
    private final AltingChannel reset;
    private final ChannelOutput out;
    private final String name = "ResetThread";
    private String initialState = "one";
    private String currentState = initialState;

  public ResetThread (final AltingChannel full, final AltingChannel reset,
                final ChannelOutput out) {
      this.full = full;
      this.reset = reset;
      this.out = out;
  }

  public void run() {

    final Skip skip = new Skip ();
    final Guard[] guards = {full, reset, skip};
    final Alternative alt = new Alternative (guards);

    while (true) {
      switch (alt.priSelect ()) {
      case 0: // full
```

```
          if (currentState == "one"){
              full.read();
              reset.write(1);
              out.write(name + " full ? -> reset !");
          }
          break;
      case 1: // reset
          break;
      case 2:
          try {Thread.sleep (400);} catch (InterruptedException e) {}
          out.write ("\t" + name + " sleeping");


          break;
      }
    }
  }
}
```

```
import jcsp.lang.*;

public class CounterThread implements CSProcess {
    private final AltingChannel full;
    private final AltingChannel reset;
    private final ChannelOutput out;
    private final String name = "CounterThread";
    private String initialState = "one";
    private String currentState = "one";
    private int counter = 0;

  public CounterThread (final AltingChannel full, final AltingChannel reset,
                        final ChannelOutput out) {
      this.full = full;
      this.reset = reset;
      this.out = out;
  }

  public void run() {
    final Skip skip = new Skip ();
    final Guard[] guards = {full, reset, skip};
    final Alternative alt = new Alternative (guards);

    while (true) {
      switch (alt.priSelect ()) {
      case 0: // full ?
        break;
      case 1: // reset ?
        if (currentState == "one"){
          reset.read();
          counter = 0;
          currentState = "one";
          out.write(name + "reset ? -> counter := 0");
        }
        break;
      case 2:
        if ((currentState == "one") & (counter < 10)) {
          counter ++;
          currentState = "one";
          out.write(name + "counter < 10 -> counter++");
          // break;
        } else if ((currentState == "one") & (counter >= 10)){
          full.write(1);
          currentState = "one";
          out.write(name + "counter >= 10 -> full !");
        }


        try {Thread.sleep (400);} catch (InterruptedException e) {}
        out.write ("\t" + name + " sleeping ");
        break;
      }
    }
  }
}
```

3.6 Discussion

In this Chapter, we present the clinical application modeling language (CAML). We describe the language at a high level in Section 3.1, describe the semantics of extended finite state machines (EFSMs) and sets of communicating EFSMs in Section 3.2, and describe the execution and parallel composition of communicating EFSMs in Section 3.3. We have modeled the clinical systems as sets of communicating extended finite state machines in CAML. From the CAML model, we translate into the language of the UPPAAL tool (Section 3.4) to check properties and we generate Java code (Section 3.5) that uses the JCSP library for communications and threading. The CAML models will be used to create models of system components as described in Chapter 4, the creation of system safety properties and checking these properties against the models is covered in Chapter 5, and the generated Java code forms the basis of the case studies presented in Chapter 6.

Limitations. In this Section, we discuss limitations, gaps, and future work related to the CAML language, the UPPAAL translator, and the Java code generator. Limitations of the modeling language will affect the systems we model using the language and the properties we can check against the models. Limitations of the UPPAAL translator and Java code generator will affect our ability to guarantee system safety properties for the implementations we build for the case studies in Chapter 6.

It is important that the semantics of the UPPAAL model match those of the CAML system and, if code is generated, those of the generated code. If the semantics do not match, then each of these three systems will behave differently under the same sets of inputs. This means that properties we prove about the UPPAAL system may not hold in the code or CAML system. We want to argue that we can model a system in CAML, prove properties about it using UPPAAL and ultimately have those properties hold in the generated code. For this argument to be sound, the semantics of the three systems must match.

We have not provided rigorous proofs that the translations into UPPAAL and Java code, described in Sections 3.4 and 3.5, preserve the semantics of the CAML system. For code generation, the details of this proof will vary depending on the exact configuration of the target platform. Matching semantics and meeting timing properties with Java is particularly challenging because of the need to deal with multiple levels of scheduling (threads within a Java Virtual Machine process versus operating system process threading) and the need for garbage collection, which can be difficult to predict.

An overall limitation of this work is the inability to model continuous dynamics or check properties about continuous dynamics. In this Chapter, these dynamics were handled by abstracting to a higher level in the models using terms like the boolean 'breath_is_done' in lieu of integrating flow rates to calculate breath volume, and by replacing the high-level terms with manually written code sections during code generation.

Gap Analysis. There is a tradeoff between the expressiveness of the modeling language and the computational cost of verifying properties. CAML is kept toward the inexpressive end of the spectrum to make model checking at the point of care more tractable. It would be useful to have more options for communication patterns, parallel operations, and the other language features discussed in Section 4.7, but careful evaluation of additional features is needed to assess the trade-off of language expressiveness versus model checking cost.

CAML allows for synchronous and asynchronous communication channels. Distributed systems of medical devices such as OpenICE and MDCF are built on middlewares that support a wide variety of communication patterns and are usually deployed on TCP/IP networks that may be lossy, reorder packets, and otherwise change the communication semantics. Modeling language support for these networks and middlewares is needed to realistically model clinical deployments.

Future Work. There is much future work to be done on evaluating this tradeoff between the precision of the models and the scalability of checking the modeled systems. Some properties could be statically checked before systems are assembled in a pre-deployment environment where more time and computational resources are available. This could allow whitelisting devices for particular applications in advance of system assembly.

Another promising direction for future work is the incorporation of CAML models, and particularly CAML device models, directly into the OpenICE platform along with the translation and model checking tools.

Chapter 4

Architecture for Provably Safe Interoperability

In order to prove properties of interoperable medical systems, we need to know something about how the systems are built and where they will be used. These systems are made out of many components including medical devices and communications infrastructure like network switches and may include other non-medical devices. Medical devices that are built with the intention of being part of an interoperable system will have well defined, standardized data outputs and may accept commands over their network interfaces. This Chapter describes the additional data needed from interoperable devices to enable safety analysis of the systems that use them as components.

The architecture used here closely follows the ICE architecture described in Section 2.3.1.

Figure 4.1 shows the components of the system. Medical devices are connected together through a network controller in a hub-and-spoke topology. Data from connected devices is made available to applications, which may send control signals back to some devices. Each component of the architecture has a specific purpose and is necessary to achieve the goal of proving safety properties over the whole system. Components come from different sources and the system is assembled by users at the point of care.

Two Use Scenarios. The most dramatic example of system assembly by end users would be putting together devices that have never been used together at the patient's bedside. A more likely scenario is for a hospital biomedical engineering department to assemble and test systems using the specific brands and models of devices available at that hospital before the system would be used for patient care. The hospital would be assembling the system, including software applications, out of components that are separately approved by the FDA. The MD PnP Program has been working with FDA to develop a regulatory pathway for component-wise approval of such systems. A key part of this is the notion that devices could include connectivity as part of their intended use. A device's intended use could include sending data to other devices and accepting commands over a well-defined interface. The manufacturer would need to be able to argue that their device is safe without knowing in advance what other devices might be receiving the data or sending the commands. This work provides a framework and some examples of an approach that enables such argument.

Medical devices in the system come from medical device manufacturers. This includes traditional medical devices like infusion pumps or ventilators as well as software-only devices such as the ICE Applications. It is expected that most ICE applications will themselves be regulated medical devices. Device manufacturers provide the device model, either as part of the communication protocol for devices capable of transferring their model during discovery or as a separate file that is pre-loaded onto the ICE Supervisor for devices that are only, to use the ASTM 2761-09(2013) ICE standard terminology, model-compliant. The app writer provides the app itself, along with the patient model, caregiver model and set of safety properties.

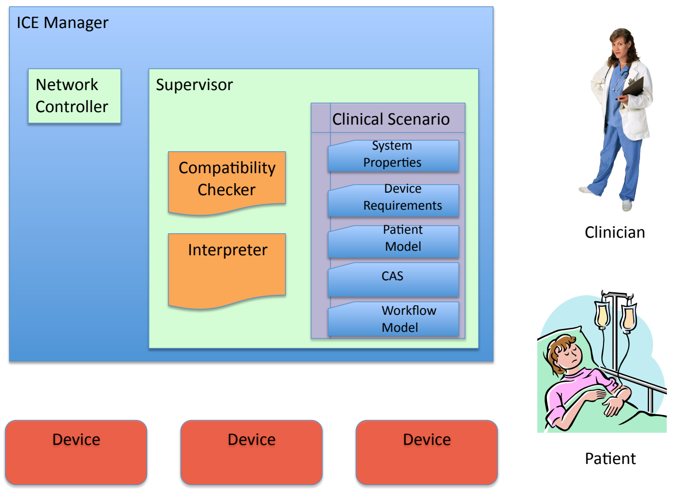


Figure 4.1: Components of the System

The architecture described here supports two distinct kinds of tests of the plug and play system. The first test is done when devices are connected or disconnected, and checks whether the set of connected devices at that time satisfies the needs of the clinical application as captured in the clinical application's device requirements. The second test is done after a sufficient set of devices is connected, and tests whether the entire assembled system satisfies safety properties provided by the clinical application developer. These tests, of device requirements and system safety properties, are the reason for creating device models, application models, and the other system components described here.

This Chapter describes the parts of the system and discusses how these components are created. Some components can be provided in multiple ways. For instance, device models can be a simple list of variables or a more complex state machine capturing device behaviors. For these components, we discuss the different forms they can take and the tradeoffs that are made by choosing one form or another. Chapter 5 shows how the pieces come together and are used to test whether the assembled system meets the application developer's requirements and safety properties. Chapter 6 provides detailed case studies.

4.1 Device Models

When a medical device is connected to a plug and play network, it must identify itself and describe its capabilities. This description is called a device model. Device models contain identifying information about the device, essential information about device functionality accessible over the network, and may also contain meta-data about the accessible functions. We describe two distinct kinds of device models here: static models made up primarily of variable lists, and state machine models that capture some behavioral aspects of the device.

Static device models list the data that the device can provide and the commands it can accept through its network interface. For example, a pulse oximeter could provide SpO_2 and Pulse Rate as output variables. Metadata, like the averaging time of a pulse oximeter, is simply listed as another variable. If the averaging time can be set through the device's interface then it is also listed as an input.

It is up to the device manufacturer to decide which functionality to expose over the network interface. Device manufacturers may choose to limit the functionality accessible via the network or impose restrictions on how it is used in order to support their own FDA filings and safety arguments. This is challenging because the device does not know the context in which it is being used; this is why safety checks are left to the clinical application in this work. If manufacturers limit their devices' functionality in order to reduce the potential for misuse, then they also reduce the possibility for beneficial use.

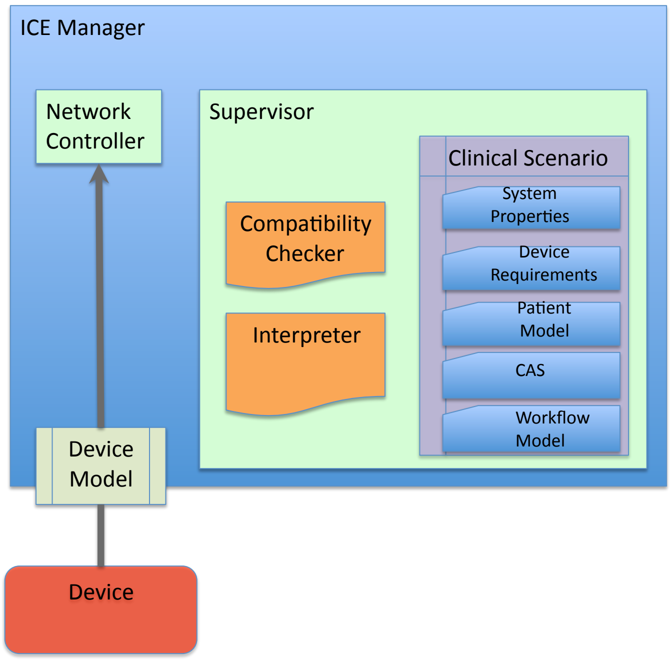


Figure 4.2: Device Models are provided by devices at connection time

The manufacturer can not anticipate all possible ways in which the data from the device may be used.

Applications that run on the Supervisor and use the connected devices to accomplish a clinical goal are called Clinical Application Scripts (CAS's). Developers of a CAS bundle it with a set of Device Requirements and a set of System Safety Properties. The bundle of three documents is called a Clinical Scenario Package.

When a device is connected to the network, it sends its device model to the Supervisor via the Network Controller. The Supervisor checks whether the set of connected devices fulfills the Device Requirements of a Clinical Scenario Package by comparing the capabilities described in the models with the needs represented in the requirements. Once a set of devices fulfill the Device Requirements, the Supervisor checks whether the System Safety Properties hold for the system resulting from composing the devices with the CAS. If all of these checks are met, then the Supervisor can start running the CAS. In some implementations, such as OpenICE, some of these checks are performed by the application itself; this facilitates (re-)checking properties while the application is running.

A device model includes a header containing information about the device, such as its serial number, information about what data the device can supply (e.g., blood pressure or heart rate), and information about the data, or meta-data (e.g., sample rate and processing time). Device models may also include a state machine that encodes some of the device's behavior.

Device models are divided into a header and a body. The header contains a high-level description of the device- its manufacturer, model number, software revision, and device type- and a description of the format of the body.

The body of the device model describes the capabilities of the device. We define two types of device models with different amounts of information about the device. These device models all use the same header format, but represent device capabilities using static variable lists or behavioral models.

This section describes the header format and both types of device models using a simplified X-Ray Machine and Ventilator as running examples.

4.1.1 Header

The header contains the following items:

- FDA Unique Device Identifier (UDI)
- Manufacturer Name
- Device Name
- Software Version
- Patient ID
- Device Model Body Type
- Device Model Body Language

```
struct DeviceIdentity {
    UniqueDeviceIdentifier unique_device_identifier; //@key
    LongString manufacturer;
    LongString model;
    LongString serial_number;
    Image icon;
    string<128> build;
  };
```

Figure 4.3: Device Model Header IDL

The FDA Unique Device Identifier (UDI) is a work in process based on FDA guidance that requires UDIs to be printed on device packaging. This guidance does not specify an electronic format for UDI, but this is under development by FDA and related standards groups. Until the electronic format is released, we simply use a numeric version of the printed barcode. The next three items are strings of free text defined by the device manufacturer. Patient ID is a complicated issue in its own right. Here, patient ID is a Medical Record Number unique to a particular patient that is used by all of the communicating devices. Propagating, validating, and managing patient identity is outside the scope of this work. The device model body type is one of two types: variable list or behavioral model.

4.1.2 Body definition for Variable List

A device model body entry contains the following fields:

1. Type
2. Encoding
3. Name
4. Direction
5. Value
6. Associated Variable (for Type:Metadata only)

Type. The Type field specifies what kind of information is contained in this record. It can be one of two values: Data or Metadata.

Data. The Data type is used for variables of the device. Device variables include information such as “heart rate”, “temperature”, “infusion rate” and so on.

Metadata. Metadata is information about data. Metadata is always associated with a single Data type element in the Device Model. This association is done using the “Associated Variable” field. Metadata includes information like sample rate, accuracy, units, and so on that is associated with a data element.

Encoding. The Encoding field calls out the specific data taxonomy or ontology that the entry Name is taken from. Examples of Encodings include “11073”. “HL7”, “SNOMED”, etc. It is expected that creators of device models will use variables chosen from common, well-known taxonomies such as those and that creators of Clinical Application Scripts for Supervisors will accept devices using the same encodings.

Name. Names are the name of the variable in the specified Encoding.

Direction. The Direction field can be "Input", "Output", or "Both".

Value. The Value is the numeric or other type of value for the variable. The type, units, and so on are defined by the Encoding.

Associated Variable. Metadata must always have an Associated Variable. This is the other variable in the Device Model that the Metadata is about. For instance, a Metadata entry for "Sample Interval" could be associated with a Data entry "Temperature".

4.1.3 Body definition for Functional Models

The format for encoding a functional model in a device model is CEFSM as described in Section 3.1.

4.1.4 Example: X-Ray and Ventilator Device Models

Variable Lists

X-Ray:

- Provides:
    1. exposure time
    2. image

3. external trigger latency

- Accepts:
  1. external trigger

Ventilator:

- Provides:
  1. instantaneous flow rate
  2. age of flow rate sample
  3. respiratory rate
  4. local clock time
  5. respiratory rate change notification
  6. inspiratory time
  7. inspiratory hold time

Functional Models. When a device is plugged in, it sends a model representing its functionality including input and output variables. The supervisor analyses this model together with the Device Requirements to decide whether a device is suitable for a particular CAS.

There are advantages to having the Supervisor calculate meta-data from a model. In particular, the device manufacturer doesn't have to anticipate what the CAS designer will need and the Supervisor can calculate exactly the data it requires. On the other hand, if the device provides the meta-data, then no calculations necessary in Supervisor and there is no duplication of effort. Relaying enough information to allow the calculation of meta-data may run the risk of exposing algorithms that the device manufacturer considers trade secrets. For this reason, many device manufacturers will prefer to transmit meta-data rather than a detailed accounting of how their systems work internally.

X-Ray Functional Model Example. Figure 4.4 illustrates a functional model for an x-ray machine.

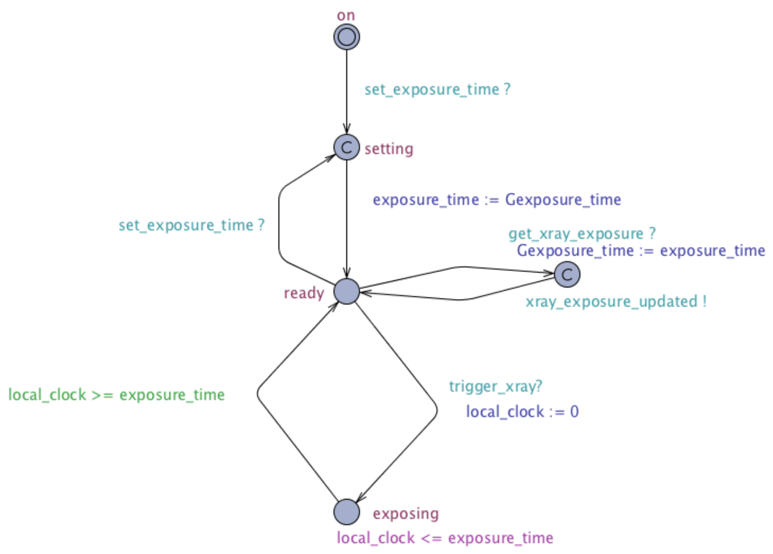


Figure 4.4: X-ray / Ventilator Example: X-Ray Device Model

Ventilator Functional Model Example. Figure 4.5 illustrates a functional model for a ventilator.

4.2 Device Requirements

Clinical applications are created to solve a clinical problem. While patient treatment can seem (and sometimes is) very dynamic, non-linear and even chaotic, doctors and nurses frequently follow well-defined workflows for specific pieces of patient care. These pieces are sometimes short procedures like placing an IV line, intubation, or checking vital signs. Some procedures may be much larger in scope, for instance weaning a patient from a ventilator or some of the more routine surgeries. Automating these workflows with a

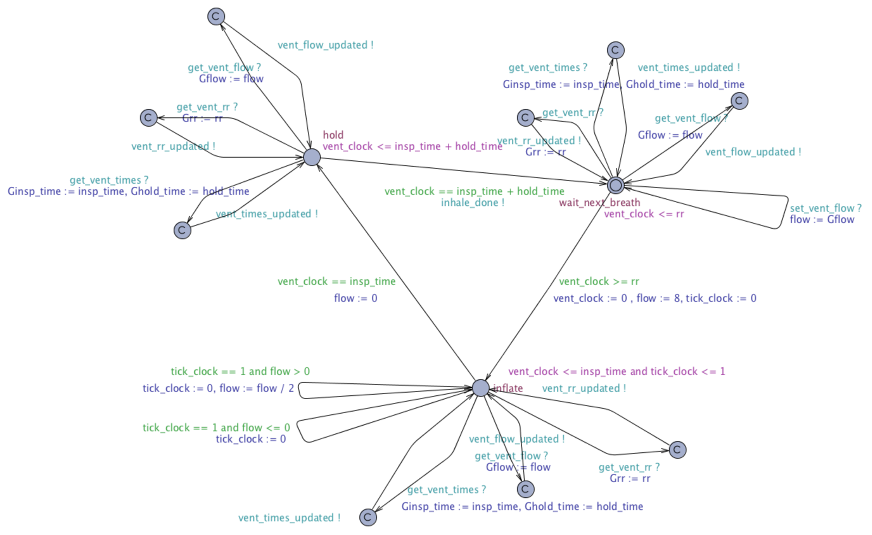


Figure 4.5: X-ray / Ventilator Example: Ventilator Device Model

clinical application can relieve caregivers from performing some repetitive tasks, such as documentation or responding to unnecessary alarms, allowing them to spend more time on aspects of patient care that can't be done by a computer algorithm. Automating surgical workflows here does not mean developing surgical robots but rather building systems and algorithms to support the clinical staff and improve patient safety during surgery. This could include smart alarms, automating checklists, software to coordinate between the OR and the ICU, etc. Automation of tasks in medicine has proceeded more slowly than automation in other domains like manufacturing because of technical and cultural barriers, regulatory concerns, and the difficulty of handling patient variability.

The technical barriers to building applications around the practice of medicine are largely around getting data out of medical devices and control signals back into those devices. It would be easier to design clinical applications if medical devices operated as basic sensors and actuators with well-defined characteristics like latency, averaging time, and

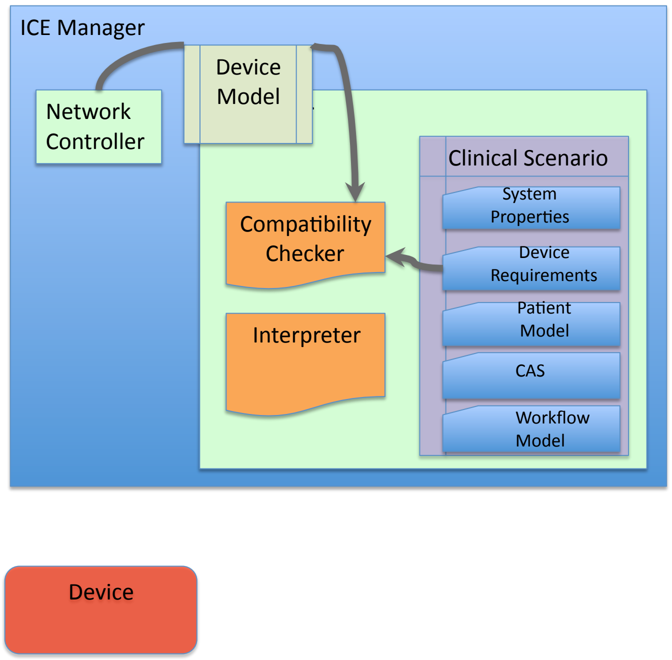


Figure 4.6: Checking Device Requirements against Device Model

so on. This is how sensors and actuators for industrial control systems are often designed. Currently available medical devices are very limited in the data they make available, and generally do not allow any external control. This is because device manufacturers do not see a business advantage to opening their devices' interfaces, but do perceive an increase in liability. They commonly claim that they would be liable if their device did something to injure a patient based on a command received from another medical device in a way that they would not be liable if the same command was manually entered by a caregiver. While there are clearly new patient hazards introduced by assembling stand-along devices into interconnected systems, some device manufacturers seem to be using this as an excuse to build proprietary interfaces that will only work between their devices and that require healthcare providers to buy all of their equipment from a single vendor if they want the equipment to exchange data.

The creators of a clinical application have requirements about the medical device capabilities needed to support the application. These requirements are formalized as a set of Device Requirements, which become part of the Clinical Application Package. Requirements elicitation is a complex topic closely tied to the hazard analysis and safety analysis of the devices and overall system [48], [78], [63]. Device Requirements may be written in two ways: as a set of constraints on variables, or as a set of CTL formulas over variables.

4.2.1 Variable Constraint Device Requirements

Variable Constraints, as the name suggests, are simple bounds on the values of variables contained in the device model. A single constraint takes the form of a triple ⟨ VARNAME OPERATOR VALUE ⟩, where VARNAME is the name of the variable, OPERATOR is one of $=$, $\neq$, $>$, $<$, $\geq$, or $\leq$, and VALUE is a fixed value of the type represented by VARNAME in the nomenclature. So, a valid constraint on the value SPO2_AVG_TIME, defined as an integer number of seconds, could be SPO2_AVG_TIME ≤ 5. A special kind of variable constraint with null OPERATOR and VALUE (i.e., just the VARNAME) is used to specify that a variable must be available but has no other constraints. This is useful for patient physiological values (such as heart rate, SpO_2, or Blood Pressure) that may be available but not yet populated by a device that is attached to the system before being connected to the patient. When we say that SPO2_AVG_TIME is defined as an integer number of seconds, this is simply the English language definition of the term. While progress is being made in the development of terminologies and ontologies for medical device data, there is not yet a standardized set of terms sufficient for writing device requirements. Variable constraints can be written using any set of terms understood by both the constraint writers and developers of the other models in the system.

Variables from a device may be either static or dynamic. Static variables are those that do not change while the device is connected to the system. These include things like the unique device identifier, serial number, and firmware version. Medical devices have frequent updates to their firmware, but must be placed in a special mode to install updates. We do not expect that they will be able to communicate with external devices while in this mode, so firmware versions are unlikely to change during use. Static variables also encode parameters that are fixed for a particular model of device, like maximum or minimum settings. Dynamic variables encode values that are expected to change – the time since the last update to a reading, or the value of a patient's physiological parameter. Device constraints on static variables can serve to disqualify or admit a device for an application. When dynamic variables are checked against device constraints, it serves only as a 'spot check' that may disqualify a device, but can not ensure that the variable checked will not violate the constraint in the future. If it is necessary to check, for instance, that the variable 'averaging time' will always stay between 2 and 16, it is not enough to see that the current value of the variable is 4. Better constraints can be written if the encoding of the variable includes bounds, as many 11073 data encodings do. Then it is possible to check that the lower bound of 'averaging time' is ≥ 2 and the upper bound is ≤ 16. Because we do not have bounds on many variables and they can and do change during use, in OpenICE we check some key constraints on every update to the value.

Many constraints will be enforced by applications as part of their normal running.

For instance, the Xray / Ventilator Synchronization app [7], [5], [6] requires that the patient's lungs remain stationary for longer than the X-ray exposure time. The amount of time the lungs are stationary is dependent on several ventilator settings, and the X-ray exposure time is likely to be programmed after the system is assembled, possibly with the assistance of the synchronization app. Properties like these, that are dependent on settings or patient data that are dynamic, should be checked by the app at run-time. Some properties, particularly meta-data like averaging time, are likely to change while the device is in use. Applications will need to monitor relevant metadata while they are using the associated data and reject new data samples when the meta-data goes out of bounds. Some of these properties are also amenable to run-time monitoring by a separate process, but that is beyond the scope of this work.

4.2.2 CTL Device Requirements

Most variables in an ICE system are not static, and there are many interesting properties that can not be checked by looking at a snapshot of their values.

Medical device behavior is usually described in terms of modes. Training materials, documentation, and the users of devices describe them as having modes such as 'programming', 'running', 'stopped', 'paused', 'alarming', and so on. This way of talking about device behavior lends itself quite well to state machine modeling. These models can be black boxes that capture the device's behavior from the point of view of a user or another device on the network, or they can be more detailed models that describe its internal operations. Combining the models of various devices allows us to build composite models that will mirror some aspects of the behavior of the real system. These composite models allow us to check properties about the system's behavior that go beyond what we can check with the simpler constraint checks.

For instance, consider an infusion pump that allows remote control of its rate, but only after a fallback infusion rate is set. If the pump loses its network connection while it is being remotely controlled, it will revert to the fallback rate. Using constraint checking, we can set constraints that the pump must support external rate control and that the application must support setting a fallback rate. This kind of basic compatibility check is valuable. Using model checking, we can additionally ensure that the app must always set the pump's fallback rate before the app can remotely control the pump.

Device requirements may also be written in a temporal logic to facilitate testing more complicated properties with a model checker. In this work, we use the UPPAAL model checker, and write device requirements in the subset of CTL it supports.

4.2.3 Example: Device Requirements for X-Ray and Ventilator

Variable List Device Requirements.

X-Ray:

- Must Provide:
  1. exposure time
  2. image
- Must Accept:
  1. external trigger
- May Accept:
  1. exposure time

Ventilator:

- Must Provide:
    1. instantaneous flow rate
    2. respiratory rate
    3. local clock time
    4. respiratory rate change notification
    5. inspiratory time
    6. inspiratory hold time

4.3 System Safety Properties

System Safety Properties (SSPs) capture essential aspects of the system that the system designer requires must hold for the system to be safe. SSPs are distinct from device requirements; rather than placing constraints on individual devices, SSPs specify behaviors of the complete assembled system. These behaviors necessarily include behaviors of individual devices, so SSPs include safety properties of individual devices. If an application requires that a component devices incorporates or prevents certain behavior, the application developer can include one or more SSPs capturing the intended behavior. SSPs thus help to address novel hazards created by assembling the system and to mitigate device level hazards. Device manufacturers and app developers do not know the specifics of the assembled system in advance. It is critical that they specify all necessary aspects in the device requirements and SSPs.

The goal of SSPs is to help to mitigate known hazards. It is possible that they will also help to mitigate unanticipated hazards, but this is a lucky side effect not the main intent. To achieve this goal, application developers need to provide SSPs such that all known hazards are mitigated if SSPs are satisfied. Application developers are thus ultimately responsible for ensuring the safety of the system.

Safety Properties are expressed as temporal logic formulas. Because we use the UPPAAL model checker in this work, system safety properties here are written in the subset of CTL supported by that tool.

4.3.1 Device and System Level Safety Properties

The main source of system safety properties is the system hazard analysis. This is distinct from the device hazard analysis in that the system hazard analysis contains only the new hazards created by assembling the devices into a system and the device level hazards that are mitigated at the system level. Consider a system where an infusion pump communicates with a safety interlock application that may stop the infusion based on vital signs from a patient monitor. This system introduces hazards that are not present when a clinician uses a pump and monitor to treat a patient without a safety application. The application may stop the pump inappropriately, patient information may be intercepted or forged on the network by malicious actors, or the system may fail to stop the pump when it should. These are new hazards that are introduced by connecting the devices to a network and running a safety interlock application that interacts with them. If these hazards are not mitigated properly, there is increased risk to the patient. However, the system can also do a better job of mitigating device hazards than is possible when the devices are not connected. Overinfusion is a common and serious hazard that is possible whenever an infusion happens. Infusion pumps do not have any means of monitoring the patient and typically do not directly measure the flow rate of medication. They can take measures to ensure that the programmed dose is faithfully delivered but they have no feedback to indicate whether that dose is appropriate for the patient. Thus many causes of overinfusion cannot be mitigated at the pump level; the pump simply does not have the necessary information. The patient monitor has the patient's vital signs, so when the system makes the pump and patient monitor information available to the interlock application then the hazard of overinfusion can be better addressed.

4.3.2 Relation of System Safety Properties to Hazard Analysis

A hazard analysis enumerates ways that the patient may be harmed. For each of these ways, it lists means of mitigating the harm and it may include a measure of the likelihood and severity of the harm. System safety properties tend to be associated with specific vectors of harm and their mitigations. Overinfusion is a hazard, but it is difficult at best to write a single SSP for it because it is too high level, that is, there are too many ways that an overinfusion can happen. Rather, SSPs are tied to particular means by which an overinfusion can happen. These could include a pump failing to stop when it is commanded to either through its user interface or remotely, a pump being loaded with the wrong concentration of drug or the wrong drug, freeflow of medication through a pump, misprogramming of a pump, programming based on an incorrect measurement of the patient's weight, and so on. In many cases the SSP will involve data from multiple devices. SSPs can and should be written for particular ways that an overinfusion could happen. Some of these could be written at a global level, for instance that the pump should never infuse more than x milligrams of a medication in an hour. Parameterized safety properties like this also have regulatory advantages; the platform can be approved based on its ability to monitor and enforce the parameter while leaving the responsibility of choosing a value for a the parameter to the clinician or health delivery organization where the device is used.

Some safety properties are too complex to be modeled in this framework, especially properties involving continuous dynamics. These can be written into the application, and a simpler SSP can be written around the behavior of the application. For example, we may wish to write a safety property that says that if a patient is receiving an infusion of medication and the vital signs of the patient significantly diverge from the predicted values from a patient model that incorporates a model of the patient's reaction to the drug (personalized pharmacokinetics for that drug) then an alarm must be triggered and the infusion pump must stop. This essentially says that if the patient reacts to treatment in a way that we don't expect, then alarm and stop treatment. Because it involves a sophisticated patient model that incorporates pharmacokinetic / pharmacodynamic (PK/PD) modeling, it would not be amenable to modeling in finite state machines or encoding as CTL properties, though it could be modeled as a hybrid system or approached through a combination of modeling methods as in [9]. These complex behaviors can be written into the safety application, which can generate an externally visible event when they occur, and the SSPs and application model can be written to use this event. This simplifies the model, makes the properties tractable to analysis, and still allows the application to incorporate sophisticated behaviors. This approach was used in [7] where code that was automatically generated from the application model was merged with handwritten stubs implementing the complex behavior.

4.4 Patient Model

A patient model is a model of the parts of a patient's response to treatment that are relevant to a particular application. Patient models are closely tied to a particular use case and a specific patient population. Generic patient models that attempt to model all of a patient's physiology and response to a general set of stimuli exist but are not well suited to model checking because of their size, complexity, inherent mathematics that are not well suited for formal analysis and the difficulty of matching the model physiological semantics to the other components of the system. Our intention is not to create general models of human physiology, but rather tailored models of the limited aspects of a patient's response to treatment that are relevant for a specific application. Patient models in this system are created by application developers and meant to be used in conjunction with a particular app. Human physiology is complex, poorly understood, and highly variable from one person to the next and the intended use of apps covers the full spectrum of medicine. Medical treatment is generally done 'to effect'- treatment is rendered, and stepped up until the desired outcome is reached, rather than being entirely determined in advance. Patient variability is part of this, and gaps in medical knowledge is another part. Patient models are specialized models of the patient as a black box based on inputs to the patient, the outputs of devices and other workflow processes and outputs from patient, which provides input to devices and other processes. App safety is closely tied to the patient model. An application intended for use on adults will likely be unsafe for use on small children. Generally, an assembled system that is safe for one patient may be unsafe for another patient who is sicker, or more sensitive to treatment because of physiological differences.

Patient models can be useful if they simply indicate the potential directions the patient may go rather than specifics of how they get there. The point is not to mathematically simulate the detailed behavior of the patient in reaction to treatment, but simply to broadly indicate the spectrum of outcomes. Patient models should include only enough detail to support analysis of the System Safety Properties of interest to the developer.

Patient models can represent a specific individual, a population of patients, or any possible patient. These types of patient models go from specific to general. Within the state space of possible patient reactions, the specific individual model will be the most constrained, the patient population will be broader, and the model aiming to represent every possible reaction will be the broadest.

Creating a mathematical model that faithfully represents the reactions of a specific patient or even a patient population is difficult and may not be necessary to assess the safety of a system. The modeling systems we use often do not support all of the mathematical constructs used in the analysis of clinical data sets. Sets of differential equations are commonly used and these are incompatible with many model checking tools. The level of detail and specificity provided by these models is not needed for checking most safety properties. Instead of trying to capture all of the mathematical detail, models should be written at the proper level of abstraction to support checking the desired properties. Most safety properties are written about the boundaries of the system; the edges of the state space where the patient's condition is seen as undergoing a transition. A model at a higher level of abstraction that captures only the pieces of patient status essential for decision making is more efficient for model checking and easier to produce and validate while still allowing checking of the critical transitions.

In developing the Computer-Assisted Resuscitation Algorithm (CARA) [2], a system for managing fluid resuscitation, we wanted a patient model that would capture how a patient's blood pressure would vary with blood loss and the rapid infusion of fluid. We started out thinking that we could create a simple model of the cardiovascular system where blood pressure would drop as volume decreased. The human body is not that simple. There is a complex web of compensatory systems that react to blood loss. These reactions involve many body systems as heart rate and breathing rate change, peripheral vasculature closes down, and the patient goes into shock. Mathematically modeling these reactions quickly becomes very complex and requires choosing values for variables like the rate of change of particular vital signs makes the model specific to a particular patient. We found in the CARA project that it was much more useful to model the patient at a high level of abstraction where the model remained applicable to a broader set of patients and more tractable to analysis. This general principle has held for many varied applications.

Patient models are specific to an app, patient population, and use environment – that's why they're bundled with the app. An app may have more than one patient model, in which case it's up to the clinician to pick an appropriate one when the app is started. Some automatic configuration based on patient attributes like age, height, weight, etc. may be possible. The patient model's terminology and semantics are expected to be aligned with the other pieces from the application developer.

4.4.1 Interaction Between Patient Model and Safety Properties

There is a close relationship between patient models and system safety properties. System safety properties are written about aspects of the system that are safety related. Ultimately, this means they are related to patient safety, and patient safety is tied to specific patient populations. Patient models and safety properties need to be written at a similar level of abstraction and using the same terminology.

We present some safety properties from the case studies and discuss how they interact with the patient models. A basic assumption of both the x-ray / ventilator synchronization and PCA safety interlock case studies is that patients need to breathe. This is a broadly applicable requirement, but would not be true, for instance, of a patient model for a cardiac bypass use case where the patient would not be breathing and their heart would not be beating during portions of the surgery. We could write an even more abstract model, something like 'patients need oxygen' that could cover more use cases, but the act of breathing is important for both PCA and X-Ray synchronization and writing the safety property directly about breathing avoids the need to introduce model components tying breathing to oxygenation.

There is considerable variability in the details of patients' physiology. One approach to managing this variability is to group patients into populations then create models for each of the relevant populations. For PCA, these models could capture patient populations covering different age or weight ranges, or tied to specific comorbidities like COPD that would affect the treatment plan.

The patient model is not intended to be comprehensive. It is also not intended to reflect the actual response to treatment of any particular person – no individual patient is expected to respond exactly like the patient model. Patient models radically simplify physiology and the response to treatment. They may reflect an 'average' or 'typical' patient response, but in many cases it is more useful to model a patient whose response to treatment makes it difficult to keep them safe.

For instance, in our PCA safety interlock app, the patient model is written to respond much more quickly and much more negatively to a drug dose than any real patient. Where a real patient might need to be overdosed by 10 units of medication and then take 30 minutes to absorb enough drug to have a harmful result, the patient model needs only 1 unit of drug to cause an overdose and reacts in less than a minute. For the X-Ray synchronization app, a particularly challenging model is a patient who takes a very long time to exhale (leaving little time for the x-ray exposure) or whose SpO_2 drops very quickly when ventilation is paused. With these patient models, we make the argument that if the system can respond quickly enough to prevent adverse events for the unrealistically bad patient model then it will be able to prevent these events for any real patient.

These patient models are specifically designed to challenge particular safety properties. They are designed in an antagonistic way to see if we can create a patient model that will break the property, and so we term them adversarial patient models.

4.5 Clinical Application Script

The point of building an interoperable medical device platform is to enable novel clinical applications. These could include better alarms, integrated displays of data from multiple devices, clinical decision support, physiologic closed-loop control, and other applications we haven't yet invented.

Clinical Application Scripts define the behavior of the applications. An ideal clinical application script (CAS) would include sufficient detail to allow the core application code to be automatically generated from the script. An example of this is shown in [7] and described in detail in Chapter 6.

Clinical Application Scripts need to be written in a form that is sufficiently formal to allow analysis of the device requirements and safety properties, sufficiently rich to allow for many types of complex behavior, and yet not too difficult to understand and create because the CAS for an application will be provided by the application developer. Balancing the expressive power, difficulty of model checking, and ease of use of a language is a difficult task. In this work, we use communicating extended finite state machines (CEFSMs). Tool support for this language includes code generation and translators into the input languages of several model checking tools, as described in Sections 3.4 and 3.5.

A CAS is not essentially different from a behavioral device model. This is because from the point of view of the interoperability platform there is no fundamental difference between a physical medical device and a software application. Physical medical devices act directly on patients, and this is of course an important distinction from software, but both implement ICE interfaces and incorporate behaviors that are important to understanding the overall behavior of the system as a whole. Behavioral device models and Clinical Application Scripts are given different names in this system because the application developer who creates the CAS is assumed to also supply the device requirements and system safety properties necessary to ensure that the system behaves as they expect.

A CAS expressed as a CEFSM is a set of state machines. Each state machine consists of a set of states connected by transitions. Transitions have guards that can include expressions over variables and communication actions and actions that are performed when the transition is taken. Transitions are enabled when the guard conditions evaluate to true and the communication action can happen. The full semantics of CEFSMs are described in Chapter 3.

4.6 Caregiver Workflow Model

Plug-and-play is not end end unto itself. Rather, it is an enabling technology that makes it easier to build versatile systems. In the medical domain, the context of use of a system and the specific intended use to which is is put are important from both safety and regulatory perspectives. We bring in the use context and some aspects of the intended use by modeling caregiver workflows. There is a rich history of workflow modeling and a wide variety of modeling languages are available.

Why such a proliferation of workflow modeling languages? A "perfect" model that doesn't abstract any details of the real system and could be used for any purpose is not possible, and probably not desirable. Some amount of abstraction is necessary, and in picking a workflow modeling language and a toolset, we pick the aspects of the system that are most important and model those. Dropping the other aspects allows us to have models that are a manageable level of complexity and a useable size.

Medical Protocols. Some medical workflows are inherently procedural or regulated and defined as linear sets of operations. These tend to be standardized responses to emergency conditions or common practices where a consistent methodology is important. Emergency response procedures include ACLS protocols and documents like the Stanford Emergency Manual. Common practices include procedures like starting an IV, programming an infusion pump, or weaning a patient from a ventilator. Protocols are sometimes instantiated as checklists, which can be paper-based or electronic in various forms.

As Computer Scientists and Engineers looking at written protocols like ACLS or the Stanford Emergency Manual, it's easy to get the misconception that patient care is systematic, even algorithmic. While some aspects of care can be quite procedural, it is important to realize that in practice workflows are almost never as clean, straightforward, and linear as they look in these diagrams and protocols.

Medical workflows include multiple activities occurring in parallel, often with strict timing requirements, 'just in time' resource allocation, communication and coordination between activities, usually without benefit of a central scheduler. Exceptions are the norm. These workflows are usually fairly small, perhaps a dozen steps. The exception handling around them, because it includes everything that might go awry during the procedure, is usually substantially larger than the core algorithm.

This combination of attributes makes medical workflow extremely challenging to model. Modeling always requires abstraction. The art of modeling lies in making the necessary tradeoffs and decisions about what to include in the model in order to allow reasoning about the desired properties. Models that include extraneous information increase the complexity of checking the safety properties without adding any value. The trick here is that the modeler does not know everything about the system. Workflow models are provided with the Clinical Application Script by the application developer and are meant to be used with that model, but the application developer does not know what other devices will be part of the system. An application developer who creates, for instance, an app to support ventilator programming can use the device requirements to require that a ventilator is in use and that it supports all of the interactions in the caregiver workflow.

In this work, we are faced with creating workflow models without knowing the desired properties to be tested. We know that these workflow models will be combined with device, application, and patient models to form a system and that the end user will want to prove safety properties about this combined system. The challenge in creating these models is to include enough detail that the proofs of safety properties are possible and 'meaningful enough'.

In modeling medical workflows, it's important to remember that clinicians are usually caring for more than one patient at a time. Even routine, linear, well-documented, simple procedures like starting an IV can be and often are interrupted by alarms, nurse calls, or even emergencies in other rooms.

Workflow Model Types Fixed workflows are those with a single path through the process. Although almost no real clinical practices are so straightforward, it may be useful to model them this way. Fixed workflows may be a set number of steps or a loop but do not contain branches.

"Daisy" workflow models are so named because their state machine models have a single central state surrounded by many petal-like self-loops. These workflows represent a process where any of a number of things can happen in any order, at any time. It can be useful to model a clinician or device in this way when we don't know the details of their behavior, under the assumption that properties that hold in a system where a participant may do anything also hold when the participant is more constrained.

Behavioral workflow models attempt to capture a set of possible behaviors that adequately represent the behavior of the participant. This does not mean that they model all possible behaviors; in most cases this would be impossible. Instead, the model represents likely patterns of behavior as sequences of actions. Unlike Fixed and Daisy models, behavioral models include branches. Behavioral models are expressed as CEFSMs using CAML. Behavioral models allow for communication and coordination of action when multiple caregivers are involved in a patient's environment.

Workflow models are created by the application developer to represent the environment where they expect their application will be used. As the workflow model becomes more detailed and prescriptive, it captures more assumptions about how the caregivers will behave. The system model is checked as a whole. If safety properties are checked using a Clinician model that stipulates step 2 is always done after step 1, the results may not be sound if the clinician ever does things in the opposite order. Because clinical practice is so fluid and variable, daisy models are preferred unless there are very strong constraints on ordering.

4.7 Workflow Modeling Example: Coronary Artery Bypass Graft Post-surgical Care

Coronary artery bypass graft (CABG) surgery is a procedure where a patient's blocked coronary arteries are routed around, or bypassed, using arterial grafts. Around 427,000 of these procedures are done annually in the United States [24]. CABG surgery can have a number of severe complications including cardiac dysrhythmia, pulmonary problems, and infection [62]. The nursing staff is responsible for stabilizing the patient immediately after surgery and then safeguarding them against these complications until they are ready to be discharged from the hospital[61].

Phases of Care for the CABG Patient. A CABG patient goes through four phases in their hospital stay. These are the *pre-operative*, *operation*, *SICU*, and *discharge* phases.

The pre-operative phase includes hospital admission through the move to the operating room, including getting the patient ready for surgery.

The operation phase includes the actual surgery, as well as preparing the staff and room before surgery.

A SICU, or Surgical Intensive Care Unit, is a specialized ICU for care of post-operative patients. The SICU phase includes preparing a room in the SICU for the patient, moving the patient from the OR to the SICU room, stabilization of the patient, and monitored care for the patient after they are stabilized including a regular ongoing assessment.

The final phase is transfer of the patient from the SICU to a step-down unit or specialized cardiac surgery floor. This usually occurs within 24 to 48 hours after surgery and is followed by discharge from the hospital.

This process has been described as very linear, and in normal operation it is. In some cases, however, a patient may go back and forth between phases. For example, a patient may be prepped for surgery and then have to wait for an OR or surgeon to become available, possibly being delayed until the next day, or a patient may develop excessive bleeding from the surgical site after the operation and have to go back to the OR to have sutures replaced.

The focus here is on the stabilization phase. This phase is especially interesting in the context of medical workflows because it is highly parallel, there are many operators, many tasks, and few rigid task assignments, and treatment is very adaptive to the patient's condition. Though the treatment is adapted to a particular patient's circumstances, the overall course of treatment is fairly standardized- most CABG patients go through the same process. This facilitates studying and mapping the treatment plan.

Stabilization of CABG Patients. The SICU phase proceeds through four steps:

1. Initial Setup
2. Ongoing Assessment
3. Ventilator Weaning
4. Discharge to ICU or home

We'll focus on the first step, since this has the most people and parallel tasks. The initial setup happens as the patient is brought from the OR to the ICU by two people – the anesthesiologist or PA and another nurse from the OR. The anesthesiologist ventilates the patient with a bag-valve-mask (BVM).

The ICU nurses get a report from the anesthesia provider roughly 30 minutes before the end of the surgery with details of the procedure and patient status so they can finish preparing the room. This includes gathering and checking all the equipment, setting up the monitors and pumps, and preparing paperwork. Getting the patient set up in the room takes 10 or 15 minutes and includes the following tasks:

T_1	Move Patient into ICU bed
T_2	Program and Connect the Ventilator
T_3	Attach Patient Monitor
T_4	Connect ABP transducer
T_5	Initial Patient Assessment
T_6	Draw Blood for analysis
T_7	Connect and Program External Pacemaker
T_8	Connect and Program CO Monitor
T_9	Adjust Programming and Start Pumps
T_{10}	Apply Warming Blanket
T_{11}	Connect suction
T_{12}	Secure and Label all Wires and Tubes
T_{13}	Obtain 12 lead EKG
T_{14}	Check that all tasks are properly completed

Table 4.1: Tasks in CABG Workflow

When the tasks in Figure 4.1 are complete, the primary nurse takes over and checks that all tasks are done and the patient is stable. They then start regular ongoing assessments and deal with any complications that arise.

Actors and Devices in the Stabilization of CABG Patients. A large number of people are involved in the initial stabilization of the post-operative CABG patient. The patient in their bed is wheeled over to the SICU by two members of the operating team. These are usually the anesthesiologist or physician's assistant (PA) and a nurse from the OR. The patient is still sedated from the surgery, so the anesthesiologist or PA is breathing for them with a bag valve mask (BVM).

When the patient arrives in the SICU room, they are met by the nurse who will assume care for them (the primary nurse), two or three other nurses from the unit as available, and a respiratory therapist. The respiratory therapist will work with the anesthesiologist or PA to get the patient started on the ventilator, then the members of the surgical team leave. The unit nurses assist with getting the patient stabilized, then the primary nurse takes over care. Once the patient is stabilized, the EKG technician arrives and obtains a 12 lead EKG.

A_{PN}	Primary Nurse
$A_{N1} \ldots A_{N3}$	2 or 3 other Nurses from Unit
A_{AN}	Anesthesiologist or PA from OR
A_{ON}	Nurse from OR
A_{RT}	Respiratory Therapist
A_{ET}	EKG Tech

Table 4.2: Actors in CABG Workflow

The patient is connected to several devices in the SICU. The most visible of these is the patient monitor, which measures the patient's heart rate and rhythm, invasive and non-invasive blood pressure, oxygen saturation, temperature, and respiratory rate. They may also be connected to a cardiac output monitor and an external pacemaker. The patient's temperature is lowered during surgery, so a forced air warming blanket is used to raise their temperature back to normal. Patients usually receive IV fluids and several drugs, which are administered with infusion pumps. Patients are typically connected to three infusion pumps, each capable of controlling four channels of fluids.

D_{PM}	Philips MP70 integrated patient monitor
D_{CO}	Cardiac Output Monitor
D_{EP}	External Pacemaker
D_{BED}	ICU Bed
D_{WB}	Warming blanket
$D_{IP1} \ldots D_{IP3}$	3 Alaris Medley infusion pumps
D_{EKG}	12 lead EKG
D_{VENT}	Ventilator
D_{SUC}	Suction

Table 4.3: Devices in CABG Workflow

Workflow for CABG patient stabilization. Each nurse is capable of doing multiple tasks from the list. They choose one based on what they see as the most immediate need and where they can physically fit in around the patient. Some actors have a specialty (e.g., the Respiratory Therapist and EKG tech) and focus on that single task. Some tasks need to be done before others, and the actors know this ordering from previous experience. For example, the warming blanket can't be put on before the external pacemaker is attached, or it will just need to be removed again. The high level scheduler captures this implicit ordering.

Draw blood for ABG, complete blood cell count (CBC) and other blood work as ordered.

Resources, Resource Contention, and Scheduling. Devices, personnel, access to the patient, and the physical space around the patient are all considered resources. Contention for these resources is a major factor in scheduling and cause of delays or deadlocks. For example, an x-ray can only be taken when everyone except the x-ray technician has left the patient's room. While this is happening, no one else can access the patient or any devices in the room.

ID	Patient Part
P_{RIGHT}	The patient's right side
P_{HEAD}	The patient's head
P_{ALL}	The entire patient
P_{ART}	The arterial line
P_{CHEST}	The patient's chest

Table 4.4: Patient Parts

Workflow Modeling. The CABG workflow described above looks complicated, but is actually vastly oversimplified when compared to what happens in the SICU. This is primarily because it leaves out the numerous exceptions, interruptions, and higher priority

Task	Required Resources	Can be done by	Required Precursors
T_1	P_{ALL}	Two or more of A_{PN} A_{N1}, A_{N2}, A_{N3}	
T_2	D_{VENT}, P_{HEAD}	A_{RT}	T_1
T_3	D_{PM}, P_{RIGHT}	A_{PN} A_{N1}, A_{N2}, A_{N3}	T_1
T_4	D_{PM}, P_{RIGHT}	A_{PN} A_{N1}, A_{N2}, A_{N3}	T_1
T_5	P_{ALL}	A_{PN} A_{N1}, A_{N2}, A_{N3}	$T_1 \ldots T_4$
T_6	P_{ART}	A_{PN} A_{N1}, A_{N2}, A_{N3}	$T_1 \ldots T_5$
T_7	D_{PM}, P_{CHEST}	A_{PN} A_{N1}, A_{N2}, A_{N3}	$T_1 \ldots T_5$
T_8	D_{CO}, P_{CHEST}	A_{PN} A_{N1}, A_{N2}, A_{N3}	$T_1 \ldots T_5$
T_9	$D_{IP1} \ldots D_{IP3}$	A_{PN} A_{N1}, A_{N2}, A_{N3}	$T_1 \ldots T_5$
T_{10}	D_{WB}, P_{ALL}	A_{PN} A_{N1}, A_{N2}, A_{N3}	$T_1 \ldots T_9$, T_{11}, T_{12}
T_{11}	D_{SUC}, P_{RIGHT}	A_{PN} A_{N1}, A_{N2}, A_{N3}	$T_1 \ldots T_5$
T_{12}	P_{ALL}	A_{PN} A_{N1}, A_{N2}, A_{N3}	$T_1 \ldots T_8$, T_{11}
T_{13}	D_{EKG}, P_{CHEST}	A_{ET}	$T_1 \ldots T_{12}$
T_{14}	P_{ALL}	A_{PN}	$T_1 \ldots T_{13}$

Table 4.5: Tasks, Resources, and People

events that call nurses away from the room. In listing the personnel, we noted that it includes several additional nurses ‘as available’. This availability is contingent on what else is happening on the floor – any of the actors may be called away at any time. When nurses are juggling the resource allocation and prioritization of activities for this patient, they’re also thinking of who else on the floor they’re responsible for and what else needs to be done. A nurse might, for instance, help for a minute or two when the most help is needed even through they know one of their patients is waiting for something, then leave to help that patient as soon as the incoming patient is settled.

In some ways, post-CABG care is the best case for workflow modeling. Many CABG surgeries are done per year and the course of the surgery, post-surgical care, and recovery trajectory are all well documented and well understood. A patient going for a CABG has a good idea of what to expect and a SICU nurse receiving a patient after the surgery has likely seen similar patients many times.

4.8 Discussion

This chapter describes an architecture for interoperable medical devices that allows proving safety properties about the composed system. Device capabilities are represented in device models (Section 4.1). These will be checked against device requirements (Section 4.2) using the process described in Section 5.3, just as system safety properties (Section 4.3) will be checked against the composed system using the procedure in Section 5.4. Patient models are discussed (Section 4.4) along with the model for the clinical application (Section 4.5) and the caregiver workflow (Section 4.6). Finally, some difficulties in workflow modeling are discussed using the example of caring for a patient after coronary artery bypass graft surgery (Section 4.7).

Limitations. This Section discusses limitations related to the architecture used for modeling interoperable systems. System components are modeled using the CAML language described in Chapter 3, and they are assembled following the ASTM ICE architecture described in Section 2.3. Architectural limitations and gaps discussed here will affect checking device models and system safety properties in Chapter 5 and the case studies in Chapter 6.

Section 4.7 describes some of the challenges of modeling workflows. It includes several attributes that are difficult to model in CAML. In particular, CAML does not have built-in components that support resource allocation, dynamic prioritization, or scheduling. In the case studies, these issues are mitigated by modeling a single caregiver who does all of the caregiver actions or by creating a ‘daisy’ caregiver model that can perform actions in any sequence. Resource allocation and scheduling will be important for modeling multiple simultaneous applications, because applications will share devices and other system components, be operated by the same clinicians, and act on the same patient.

CAML is used here to model a single clinical application and its use environment. In many cases, it will be desirable to use multiple application simultaneously to care for a patient; for instance one application to manage ventilator setting and another to monitor or control a drug infusion. Simply composing the two applications is unlikely to be successful. Applications can interact with each other through the patient, for example when one application takes an action to lower the patient's blood pressure while another is acting to raise it. Applications may compete for shared resources such as network capacity, caregiver attention, or physical locations on the patient. Identifying when resource contention exists, defining rules for composing application, patient, and caregiver models, and checking the compatibility of system safety properties are all open questions.

Patient models are challenging to create and validate. We have not attempted to create general models of human physiology, but instead to model the limited aspects of a patient's response to treatment that are relevant for a specific application. The patient models shown here are sufficient for checking the safety properties of interest, but still very abstract. We discuss creating patient models tied to individual properties ('adversarial models'), as well as models that capture a typical response to treatment. It is difficult to know what to include in a patient model, and better theoretical frameworks for creating sufficient but not overly detailed patient models are needed.

Gap Analysis. We touch on unique device identifiers (UDI) in Section 4.2.1. Unique device identifiers are necessary for correctly associating devices to patients, tracking devices, and managing multiple copies of a particular device that might be used simultaneously on a patient. UDIs have some complications, especially for composite (or in ISO 11073 terms 'hybrid') devices that are made up of a collection of individual devices. Patient monitors are usually hybrid devices that are made up of a display computer that implements signal processing and event detection algorithms, a patient interface box that connects to the EKG leads, blood pressure cuff, and other devices that touch the patient, and finally the devices that touch the patient including complex sensors and actuators like pulse oximeters. Ideally, unique device identifiers would be structured as a tree so that when a patient monitor connects to a network it would transmit not only the UDI for the monitor display computer but also those for all of the other devices involved. Prototyping, standardizing, implementing, and deploying such unique device identifiers remains as future work.

System safety properties are derived from the application hazard analysis. Performing a hazard analysis is still a very manual process, where an expert examines the system and uses their experience and knowledge of historical failures of similar systems to list the ways in which the system might fail and what mitigations might be used to reduce the likelihood of harm from failures. This can be done systematically, but a hazard analysis is never complete; ongoing maintenance is necessary as new failure modes are discovered. As hazards and mitigations are documented, we try to create system safety properties that will assure that the mitigations are effective. Some device hazards are simply out of scope from the perspective of the application developer; most physical devices document hazards around size, weight, and shape to reduce the risk of the users injuring themselves.

Checking some safety properties requires special features in the device models, for instance a behavioral model may need a special variable to indicate that a particular state has been reached. We have had the benefit in the case studies of creating device models along with the other system component models and with advance knowledge of the system safety properties we want to check. In an interoperable system with device models created by the device manufacturer and the other components, including safety properties, created by the application developer, matching the level of abstraction of the models, terminology, and including all of the necessary device behaviors will be a challenge.

Future Work. The system described here does not include network modeling, though a network model could in principle be added to represent a particular middleware or network architecture. Network components capable of distributing a device model describing network properties or Supervisors with integrated and well-characterized networks would allow application developers to write device requirements around network performance that could be tested like other device requirements. It also has limitations around patient modeling, particularly in the inability to model continuous dynamics. Future work is also needed in modeling multiple applications and their interactions, and in combining multiple CAML systems that might be used simultaneously on a single patient.

Creating a complete system with a set of models, device requirements, and system safety properties requires application developers familiar with formal modeling, the clinical use environment, and the particular clinical application. Tool support for model creation could help to make this more accessible, particularly for allowing clinicians without modeling expertise to give feedback. Feedback from clinicians is essential for building safe and effective treatment systems, and existing modeling tools are often opaque to users who are not deeply familiar with modeling systems.

We expect that patient models will be increasingly derived from large data sets, and tools and methodology for creating and maintaining patient models from growing and evolving data sets will be needed. Static patient models that are created to simulate patient behavior that is documented in large data sets can be treated like the manually written patient models in Section 4.4. Patient models that change their behavior over time as they receive new input could also be modeling in this way, at least at a high level of abstraction and perhaps omitting the details of the learning algorithm. Such learning or evolving patient models benefit from the modeling and property checking approach presented here and in Chapter 5 in that safety properties will need to be checked against these patient models at the time of use. Because the model's behavior changes over time, safety properties checked against an old version of the model may not hold with the newer one. Model checking with learning models and the regulatory science around safety-critical systems such as medical devices using such models are rich areas for future work.

A lot of responsibility is put on the application developer, who is expected to provide application, clinician, and patient models as well as device requirements and system safety properties. This follows current regulatory structures, which require the device manufacturer to define the intended use of the device and to defend the safety of the device against that intended use by systematically examining and mitigating known hazards.

Chapter 5

Proving Safety Properties of Interoperable Systems

In order for systems built of interoperable components, as described in Chapter 4, to be usable in a clinical setting, we need to be able to test safety properties. We need to be able to prove that these safety properties hold or do not hold for various combinations of applications, device models, workflow models, and patient models. A particular combination of devices may never be assembled until it is about to be used on a patient, so we may need to be able to prove these properties right at the point of care, though in most healthcare settings the systems will be assembled and tested well in advance of their use on patients. Still, device properties change, calibrations expire, and other aspects of the environment change. Testing at the time of system assembly and throughout use, for instance by re-sending a device model whenever a device attribute changes, is necessary to assure that the assumptions of the system designer actually hold at the time of use in the actual use environment.

Our goal is to prove that an application is adequately safe for its intended use in its intended use environment. The use environment for an application includes the patient model, caregiver workflow, other medical devices used in the system, and the application itself. If the application is used simultaneously with other applications, then the use environment includes these other applications, though we assume that applications will not usually include other applications in their intended use environment. We rely on the application developers to write a set of safety properties and device requirements that encode the assumptions and requirements that they are placing on the rest of the system. Intended use and risk are inherently coupled. The intended use of the application is captured by the safety properties defined by the application developer. Safety is defined as freedom from unnecessary risk. Risk is tied directly to the application hazard analysis, which is the source of the safety properties we are testing. The application developers, as the participants in the system who are defining the specific intended use of the entire system - including the application - must categorize the risks tied to the intended use of their application, decide which can be mitigated within the application, implement such mitigations, and define the assumptions they place on the rest of the system. These assumptions are captured as safety properties and device requirements. The safety properties are invariants that must hold in order for the application to be adequately safe.

System Implementation and Risk Management. There are usually many ways to implement a system, and it is up to the application developers to determine how to best mitigate the risks associated with their application and how to express the application's safety properties and device requirements. Some risks can be mitigated either with a safety property or with a set of device requirements. For instance, consider a control algorithm that takes heart rate as an input. Heart rate measurements represent a moment in time and old measurements may persist. If the measurement device is disconnected, the latest measurement may be very old. There is a risk that the heart rate used for calculation may be too old, so the application developers need to place constraints on the system to make sure that it only uses acceptably recent measurements. This could be written as a safety property $l_1 + l_2 + l_3 \leq 0.5$ where $l_{1...3}$ are the latencies of the heart rate monitor, network transmission of the measurement, and a signal processing application used for pre-processing. Alternatively, a similar end result could be achieved with the device requirements $l_1 \leq 0.2$, $l_2 \leq 0.1$, and $l_3 \leq 0.2$. Though they have a similar effect of keeping the total latency less than a half second, the system property and the set of device requirements given here are not identical and some systems that satisfy one will not satisfy the other. It is up to the application developer to design appropriate sets of system properties and device requirements to adequately mitigate the specific hazards associated with their application. Many aspects of this approach are subjective – when is a risk 'necessary'? Clinicians may choose to use applications that are objectively very risky when a patient has an otherwise untreatable condition that is immediately life threatening. The safety and correctness of the system ultimately rest on the ability of system designers to appropriately formulate these properties.

Constraining the latencies of the system components is one way to ensure that measurements received by the application are not too old. Another approach, if synchronized clocks are available, is to have each measurement device add a a timestamp when the measurement is taken so that the application can calculate the age of each measurement and ignore those measurements that are too old. This approach ensures that the application will not use data that is too old, but if the system is slow the application may never receive any usable data. In OpenICE, we combined these approaches. Data is timestamped by (most) medical devices, again when it is processed by the ICE Equipment Interface, and again when it is received at the Supervisor. This allows the system to detect inaccurate device clocks when the device timestamp varies significantly from the Equipment Interface timestamp and document network latency for every piece of data by comparing the Equipment Interface and Supervisor timestamps. In OpenICE, we tag data as old and put a visual marker over the device icon when the network latency is more than a few milliseconds.

Mitigating System-Level Hazards. Application developers can mitigate some system-level hazards within their application, and device manufacturers can similarly reduce some system-level hazards within their devices. An application that detects that required vital signs data is not present, or is too old to be used, or lacks some necessary meta-data can automatically switch to a failover algorithm that does not need that input, or switch to an alternate source for the data if one is available, or ask a caregiver to replace or reconnect a device. For instance, an infusion pump that allows applications to externally control the dose and rate of an infusion may require that the application first set a failover rate to be used in case of accidental disconnection. This will cover not just accidental disconnection, but disconnection for any reason including malicious action.

The key argument here is that an application developer can encode a set of safety properties for their application such that if these safety properties hold then all known hazards whose mitigation relies upon components external to the app can be mitigated.

Hazard Analysis. Our core claim is that if the application's system safety properties hold for the composed system, then the system is adequately safe for the application's intended use in that use environment. Because the system safety properties come from the application hazard analysis, it is important that the hazard analysis is sufficiently complete. Measuring the completeness of the hazard analysis, like completeness of any set of requirements, is difficult. You know a hazard analysis is incomplete when it does not contain a known hazard. However, there are always unknown hazards. Over time, previously unknown hazards become known when they actually occur in use of the device. At that point, the hazard should ideally be documented, added to the hazard analysis, and mitigated in a new iteration of the application. This is one reason why reporting mechanisms for accidents and near misses are important. The ICE data logger is a key part of the system and an important tool for improving patient safety.

The hazard analyses for the case studies here are listed in Appendix A. These hazards were compiled by a process of reviewing related literature, examining similar devices, and most importantly, through long, detailed discussions with experienced clinical engineers, device users including doctors and nurses, and engineers experienced in the development of the involved medical devices.

Hazard Analysis of the application program includes application-specific hazards, the new hazards introduced by the assembly of the system, and device-level hazards that are mitigated at the application level. Hazards that are internal to the individual devices are assumed to be handled by those devices. For example, failures of ventilator valves are assumed to be handled by the ventilator, but the hazard of overinfusion by an infusion pump can not be handled entirely at the device level because the device has no way to monitor the patient. Overinfusion can be better mitigated at the system level, where an application can be aware of the infusion pump settings, the patient's vital signs, and the treatment plan simultaneously. Generalizing this example, some device-level hazards may be mitigated by features of the app. This is likely to be part of the effectiveness of the application.

Similarly, some system-level hazards may be mitigated by the devices. For example, consider a x-ray / ventilator synchronization system where the ventilator implements a pause feature. Rather than taking the x-ray by synchronizing with the motion of a running ventilator, the system sends a pause command to the ventilator and then triggers the x-ray while the ventilator is paused. This system introduces the system level hazard that the app might send repeated pause commands to the ventilator, which could cause the ventilator to never resume ventilation. This hazard could be mitigated by a feature of the ventilator that causes the ventilator to lock out repeated pause commands for a period of time. So the ventilator will execute the first pause command but then refuse any additional pause commands for a preset time interval, say 2 minutes. In this way, the ventilator manufacturer could allow limited external control of their device, while still mitigating much of the additional risk introduced by it. Hazards here function similarly to exceptions in a traditional programming language. Hazards can be mitigated locally or passed to a higher level where they are mitigated by another part of the system.

Deriving System Safety Properties. Our approach is to start with the hazard analysis for the application and derive a set of safety properties from it. These safety properties can be checked against the parallel composition of the application, clinical workflow, patient, and device models. This process is shown in Figure 5.1, where the models feed into the checker. The checker will find that either all of the safety properties hold, or that some properties do not hold. For properties that do not hold, the checker will provide a counterexample. The checker provides this result to the Supervisor, which can tell the user the outcome and, if some safety properties do not hold, give them the option of overriding the failed safety check. Medical devices are used in a wide variety of unpredictable environments, and it generally safer for the patient if the system assumes that the clinician knows best and allow them to override the automatic checks after informing them of any failures. If a user does override the safety check, this should be logged, and ideally the logs should be reviewed periodically to detect any unnecessary overrides and to discourage any users from overriding safety checks anytime other than in an emergency. Some clinical systems that do automatic checks, like drug libraries for infusion pumps, have some settings that can be overridden and some that can not. It may be desirable to set up multiple categories of safety properties such that some could be overridden by anyone, some can only be overridden by a specific subset of users, and some safety properties are considered so essential that the application can never run in an environment where they do not hold. It is important to note that more detailed analysis is also more specific and won't match as many real environments.

Figure 5 shows the overall approach to checking device requirements and safety properties. The caregiver starts the process by launching the app they want to use. The app

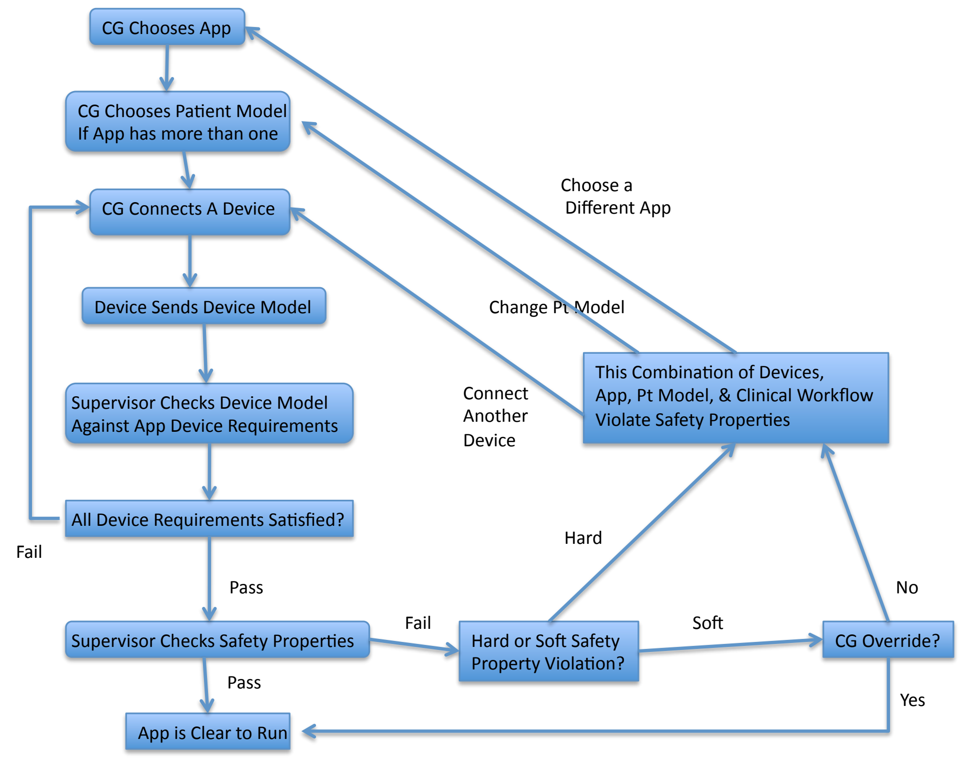

CG Chooses App
CG Chooses Patient Model If App has more than one
CG Connects A Device
Device Sends Device Model
Supervisor Checks Device Model Against App Device Requirements
All Device Requirements Satisfied?
Fail
Pass
Supervisor Checks Safety Properties
Pass
App is Clear to Run
Fail
Hard or Soft Safety Property Violation?
Hard
Soft
CG Override?
No
Yes
This Combination of Devices, App, Pt Model, & Clinical Workflow Violate Safety Properties
Choose a Different App
Change Pt Model
Connect Another Device

Figure 5.1: Workflow for Device Model and Safety Property Checks

may include multiple patient models. If this is the case, then the caregiver picks the one that best matches the patient. A PCA safety app may include models for opioid-naive patients, opioid-tolerant patients, pediatric patients, hospice patients and so on. As devices are connected to the system, they provide their device models. The supervisor checks the set of available device models against the app's device requirements and returns a pass/-fail result. If all of the device requirements are satisfied by the current set of connected devices, then the device requirements pass until the set of connected devices changes and they must be re-evaluated.

Hazards are divided into two categories depending on whether the App designer wishes to allow the clinician at the point of care to override warnings based on the hazard. These categories are ‘hard’ hazards, which may not be overridden, and ‘soft’ hazards which may be overridden. They are analogous to hard and soft limits on infusion pump programming or warnings versus alarms in many domains. Safety properties based on soft hazards are desirable, but the system will be needed in some contexts where they may not hold. The caregiver is given the option to override soft safety properties if they are not proven to hold. Hard safety properties must be proven or the app will not be permitted to run.

Once the device requirements are satisfied, the supervisor checks the app’s safety properties. If all of the safety properties pass, then the supervisor clears the app to run. If some safety properties are not satisfied, then the next steps depend on whether a hard safety property has been violated or whether only soft safety properties are not met. Hard safety properties, by definition, must hold for the application to run. Thus, if one does not hold the caregiver must change some aspect of the system to proceed. They can choose a different app, connect another device (or, conceivably disconnect a currently connected device), or select a different patient model. In the case where only soft safety properties are violated, the supervisor gives the caregiver the option to allow the app to run even though some of the safety properties do not hold. The caregiver may choose to run the app knowing that the system is not optimal, or to make a change, for instance connecting a different device or choosing another app.

5.1 Creating Safety Properties

Creating a list of system safety properties starts with a hazard analysis. Hazard analyses are done as an early part of the creation of most medical devices and are an expected part of a device's regulatory package. They list known hazards associated with the use of the device, where a hazard is something that may lead to harm to the patient or another person. It is vital that hazard analyses are updated as new hazards are discovered.

Many hazards are generic, in the sense that they will apply to many different medical devices. At a high enough level of abstraction, most hazards are generic, and there are thus common categories that hazard analyses share. Devices that have user interfaces, power supplies, leads that connect to patients, physical cases, and other common components will share hazards associated with those components. Classes of medical devices such as infusion pumps, ventilators, or patient montors share additional hazards specific to a particular device type. Within those broad classes, subsets of devices, for instance syringe infusion pumps, large volume infusion pumps, or elastomeric pumps will share more specific hazards and the same applies for particular applications. A syringe infusion pump or a large volume infusion pump could in principle be used for patient-controlled analgesia (PCA). A network-connected syringe infusion pump intended for PCA use would inherit hazards from at least five distinct categories: those that are general to medical devices, to infusion pumps, to syringe pumps, to network communication, and to the PCA use case. We have published a hazard analysis for such a pump as part of the Generic Infusion Pump project, developed in conjunction with the FDA.

Medical device manufacturers do not generally share hazard analyses or specific hazards. It has been very difficult for academic researchers or medical device manufacturers starting a new product design to find out about hazards and their mitigation. The common argument is that this information is proprietary to the companies that have documented it and that sharing such documents could expose a company to recalls or litigation if it was believed that they had not sufficiently addressed some hazard. This has had two unfortunate results. First, the time and expense of learning about hazards and their proper mitigation is a real barrier to the development of novel medical devices, particularly by startup companies. Second, it is likely that this attitude of secrecy has held back the industry as a whole, been a contributing factor to many device recalls, and a cause of unnecessary patient harm.

Generic Hazard Analysis. The lack of available example hazard analyses lead us to create the GIP hazard analysis as an open document that could be freely shared and collaboratively edited. The GIP hazard analysis has been used by many groups including several startup pump manufacturers and the AdvaMed industry consortium as a reference document for the creation of their infusion pump assurance case.

In the past, hazard analyses focused on harm that could follow from the physical design of the device (such as sharp corners), from electrical malfunctions (especially shock), or from component failure (such as free-flow of drug resulting from a broken infusion pump part). As software has become a more important part of medical devices, and especially with medical devices that consist only of software, hazards relating to software errors or unintended interactions between system components are becoming a larger part of the hazard analysis. Appendix A contains two hazard analyses that are largely about software and system level hazards. These analyses – one for a Generic Infusion Pump and the other for an X-Ray / Ventilator Synchronization system – focus on software and system-level hazards.

Ideally, the system should maintain traceability from the hazard analysis through safety properties and device requirements in order to present the user with justifications for not allowing an application to start or taking other actions. Error messages from such a system could look like "The averaging time from XYZCorp Pulse Oximeter is too high to allow SafetyApp to run. The safety property 'Total processing time ≥ 3 seconds' can not be verified. Please reduce the averaging time manually or connect a different pulse oximeter". This message points to a device requirement violation - the pulse oximeter averaging time is too high, and a safety property violation resulting from this - the end-to-end processing time is too long. The message also suggests several actions the user could take to correct the problem, namely changing the averaging time on the pulse oximeter manually or simply connecting a different pulse oximeter (with, hopefully, a shorter averaging time).

The next Section will address the process of going from a hazard analysis to a set of safety properties and device requirements.

5.1.1 Hazard Analysis and the Generation of Safety Properties

Creation of a hazard analysis is a common step in the development of any safety critical system. There are many specialized formats and elicitation processes [71] in various domains, but at its most basic the hazard analysis simply lists the things that can go wrong, their perceived likelihood, and what can be done about them. The process described here is not the only way to perform a hazard analysis, but it is customized and oriented toward producing useful safety properties and device requirements for later analysis.

When a device manufacturer does a hazard analysis for a stand-alone medical device, it is important to carefully limit the scope to exactly the intended use of the device, or even to just a portion of the device. For instance, multi-parameter patient monitors usually include an EKG and an analysis algorithm to detect arrhythmias in the waveform produced by the EKG. The simplest arrhythmia is asystole, a cessation of cardiac electrical activity. Asystole will be familiar from the many TV shows where patients “flatline” and the monitor emits a long beeping sound. Multi-parameter patient monitors, as the name suggests, are capable of simultaneously measuring and displaying several different things about a patient. Cardiac electrical activity is measured with an EKG at the same time that blood pressure is monitored with an invasive line and pressure transducer (IBP) and blood flow and oxygenation can be measured at an extremity such as a finger tip with a pulse oximeter that reports heart rate, SpO_2, and a photoplethysmograph waveform.

The various monitoring modalities of multi-parameter monitors are designed to be capable of working alone. This ensures that if the patient is only being monitored with a pulse oximeter, or only monitored with EKG, that the alarms will function correctly. There is a significant hazard in not sounding an alarm for a patient in asystole. If an alarm is heard and acted upon quickly, caregivers may be able to help the patient. The longer a patient is asystolic, the lower their chances of survival become; longer than a few minutes is almost uniformly fatal.

EKGs work by measuring the electrical potential difference between several electrodes on the patient’s chest. The electrodes are small, sticky pads with a button snap on the back where a lead wire from the monitor connects. EKGs use between three and ten electrodes; more electrodes allow monitoring and displaying cardiac activity in greater detail. The connection between the patient and the monitor can be broken if the lead wire detaches from the electrode or if the electrode peels off from the patient. This happens when the patient moves around and is especially common at night as the patients are trying to sleep. If enough leads are disconnected, and in some systems this may take only a single disconnection, then the monitor is no longer able to monitor the patient or produce an EKG. Because monitoring is no longer possible, and because the potential risk of unexpectedly stopping monitoring is so high (the patient's heart might have just stopped), the monitor will immediately alarm. This makes sense in a stand-alone system being used to monitor a high-risk patient, but leads to a great number of false positive alarms, which are themselves a risk to patient safety.

This thought process leads to a hazard analysis where false negatives are considered critical failures that must be avoided at all costs, while false positives are considered a minimal risk and largely externalized. Systems designed on this basis can be expected to produce large numbers of false positive alarms, and this is indeed what we see in hospitals today.

Hazards that can not be controlled in an single device system, and may not even be considered to be part of the system at all, may be mitigated in a multiple device system. This is why connected systems and interoperability as an enabler of connectivity have such promise to improve patient safety and outcomes.

There are two novel aspects to creating hazard analyses for devices that are intended to be used as components of interoperable systems. First, designers must consider the inputs that their device can accept, the hazards introduced by accepting those inputs, and how those hazards may be mitigated. Second, designers must consider the failure modes of the complete system. This is more properly the purview of the application designer, but the component device designer should include hazards they know about.

5.1.2 Creating Safety Properties from Hazards

Creating safety properties from hazards is a bit of an art form, but there are aspects that can be approached systematically. Safety properties in this context are intended to be used in model checking. The models they will be checked against come from a variety of sources. The most common expected used case is where a designer creates a new medical device. This designer will make a device model, a patient model representing the intended patient population for the device, a workflow model capturing the intended use and foreseeable misuse of the device and sets of device requirements and safety properties. These requirements and properties will then be checked against the composition of the designer's models with the models of the other devices in the system and the network model.

Since the models come from multiple sources, it is perhaps questionable whether they would use the same terminology or the same semantics for terms. This has been a long-standing problem in medical informatics. In this work, we make the simplifying assumption that a common terminology set with well-understood semantics has been used by all of the model creators. This is a necessary assumption to enable combining models from multiple sources, and a reasonable one. Without such a shared terminology, interoperability is not possible and thus the safety properties of interoperable systems would not be testable. There has been promising work on the creation of shared terminologies, notably the Rosetta harmonization project and the adoption of HL7 FHIR. In this work, as in the development of OpenICE [10], we use as much as possible of the terminology from the ISO 11073-10101 standard [36].

There are published best practices for performing hazard analysis; the best starting points are FDA guidance documents [76] [75] and the ISO 14971 standard [35]. Most manufacturers consider their hazard analysis proprietary, but some open examples are available [68] [14].

5.1.3 PCA pump example

An infusion pump is a stand-alone device that will be a component of many interoperable systems. The following example works through portions of two related hazard analyses - the first for an infusion pump and the second for an infusion safety application that makes use of the pump and other devices.

The core function of an infusion pump is to reliably deliver fluids at a programmed rate. Most infusion-related adverse events originate with an incorrect program. Designing infusion pump user interfaces to reduce programming errors is an important step toward reducing these adverse events, but pumps fundamentally do not have the contextual information to know when their programming is incorrect. Infusion pumps are designed as actuators that move fluid. They have sensors and alarms, but only related to the core functionality of moving fluid. They will alarm if the tubing is blocked and pressures rise too high, but not if the patient is getting an overdose or even, in most cases, if they are just pumping fluid onto the floor. The ICE architecture addresses this problem by allowing applications to communicate with pumps and with other components that can provide context. These include the computerized physician order entry system (CPOE), the pharmacy system, and electronic medical record (EMR) as well as other medical devices in the patient's room such as the patient monitor or ventilator. Information from these disparate sources can be combined and analyzed by an application in order to address root causes of over infusion such as patient identity, drug, and dose errors and to catch overinfusions that happen despite these additional safeguards.

In hazard analysis terms, the connected system can reduce the incidence or likelihood of some classes of hazards, thus reducing their overall risk, and allows for mitigation of other hazards. Connecting devices into a network introduces new hazards, such as a whole class of network security hazards that do not exist for standalone devices. As with most engineering, there are tradeoffs between different choices. The hazard analysis provides a framework for identifying the risks associated with the choices and their potential mitigations.

Many medical devices seem overly complicated or hard to use to novice users. The safety features that may seem to interfere with use have evolved over time in response to years of adverse event reports and clinician feedback. As one small example, syringe pumps including PCA pumps have mechanisms to firmly hold the top of the syringe plunger. Manufacturers of syringe pumps have devised various clever mechanisms to ease loading the pumps, but this mechanism inevitably makes it harder to load the syringe into the pump and adds additional parts and failure modes. So why include this extra part that adds complexity and makes the pump harder to load? The accuracy of a syringe pump is limited by the leadscrew and halfnut that push on the plunger and by the stiction of the syringe. It takes more force to get the plunger started moving than to keep it moving. The force needed to get the plunger moving is enough to cause it to keep moving a little once it starts, so syringes with less stiction are capable of delivering smaller doses and more accurate flow rates, especially at low rates. This has driven the development of lower stiction syringes. When a pump is loaded with a low-stiction syringe and placed higher than the patient, the weight of the fluid column in the infusion line can be enough to pull the syringe plunger. Effectively, the medication siphons out of the syringe and into the patient at an uncontrolled rate; this is called free-flow. Free-flow of medication is a hazard that syringe pumps mitigate by adding a mechanism to hold the end of the syringe plunger. Adding this mechanism makes loading the pump more difficult and introduces new failure modes and hazards; pump designers must balance these risks against each other.

The first step in performing a hazard analysis is to list the hazards. Hazards are events that may lead to an unsafe situation. The events often have multiple potential causes and these causes can be enumerated in the analysis. Compiling the hazards starts with any available sources of known hazards and never really ends. The hazard list needs to be updated as part of the ongoing lifecycle of the device as novel hazards are identified in use.

Sources of hazard documentation include the FDA's MAUDE database, academic publications, interviews with clinicians (particularly including those involved in adverse event analysis), FDA recall notices, manufacturer's records and experience with previous versions, and the designer's imagination.

A common approach is to look at individual subsystems and consider how failures of components impact use of the system. This works for both hardware and software. Often, components can fail off (not signaling when they should), on (constantly signaling a condition), or intermittently. Consider timing – a sensor that registers early (because it's in the wrong place) or a valve that takes longer than expected to close could affect the system, as could software that delivers the correct result too late or a sensor value that arrives later than expected due to network congestion.

Use Errors versus User Errors. Many adverse events are ultimately blamed on caregivers. Caregivers do indeed make mistakes. In designing equipment intended to be operated by people, it is essential to consider how the devices might be used incorrectly, what might happen if a distracted and untrained user attempts to use it, and how it might be creatively put to work in ways the device designers might not have intended. Much of this falls under what the FDA terms "reasonably foreseeable misuse'. It is generally more useful to think about use errors than user errors. That is, to think about how the user interface and implied workflow of the device design might encourage or discourage particular kinds of errors, for instance in programming a pump. Caregivers and patients are part of the care system, and explicitly included as such in this analysis. Systems need to be built in ways that encourage people to use them correctly. Getting this right requires testing with actual users during development and, most importantly, robust mechanisms to accept feedback from the field after a device is deployed. Data logging that includes key presses from devices is an essential mechanism to improve the usability of devices.

The hazard analysis is the source of system safety properties, and also suggests many device requirements. Creating these properties and requirements from a hazard analysis is not an algorithmic task; it requires understanding of the system, the intent of the system designers and assumptions of the component device designers, and the environment in which the system will be used. Formulating properties that will enforce the usually unwritten safety goals of the application designers is challenging. One intent of this work is to begin breaking down some of these unwritten and informal goals and processes with the aim of allowing more formal analysis and enforcement of the safety properties.

C4.5.2	“GIP user types wrong number while programming pump”
C4.5.3	“GIP user enters value using wrong units. E.g., milliliters instead of microliters.”
C4.5.4	“GIP user selects wrong drug from drug list.”
C4.5.7	“Use on inappropriate patient”
C4.5.9	“Infusion Order is incorrect”
C8.2.7	“Limited ability to link specific alerts to individual drugs”
C8.2.8	“Upper and Lower hard and soft limits are not sufficient to address all administration errors”

Figure 5.2: Hazards and Causes

These causes shown in Figure 5.2 are all drawn from the hazard analysis included in full as Appendix A. They are all related to pump programming, and they drive the same device requirement – the need for the pump program.

Device requirements and safety properties are related. If a value is needed to evaluate a safety property, then there will be a device requirement that the variable be available.

Pumps look deceptively simple. There are a surprising number of different settings and variables in even a basic pump. Table 5.1.3 lists the pump variables that could be required by a PCA application.

Once hazards are identified, mitigation strategies must be created for each hazard. The mitigation strategies inspire safety properties, and these safety properties dictate what information will be needed. Information can be provided from devices or from the application, which may require caregivers to provide input not otherwise available.

PCA Pump Data and Hazard Mitigation. In the following section, we will look at a key safety property for a PCA infusion safety interlock and how checking this property is affected by the data made available by the infusion pump. The property states that when vital signs are abnormal then the pump will eventually stop. Eventually in this case shouldn't be too long or the property will be trivially true because the pump will run out of fluid.

Generic Infusion Pump Data	
Group ID #	Data Element Name
Patient Demographics	
2	Name
2	Medical Record Number (MRN)
2	Age
2	Height
2	Weight
Pump Information	
1	Unique Device Identifier (UDI)
	Manufacturer
	Model
	Firmware Version
Pump Program	
3	Number of Channels
(for each channel)	
1	Drug Name
3	Drug Concentration
3	Volume to be Infused (VTBI)
3	Infusion Rate
3	Bolus VTBI
3	Bolus Rate
3	Bolus Lockout Interval
Pump State	
4	Time remaining for each infusion
4	Alarm status
4	Warning status
4	Bolus locked out? If so, time until next bolus enabled
4	Pump Data (Event) Log
Inputs	
1	Pump Stop
5	Set Patient Demographics
5	Set Program

Figure 5.3: Generic Infusion Pump Data Elements

The group ID numbers are convenient groupings for discussion, and not meant to imply levels of functionality or distinct sets. Alarms, for instance, are only mentioned in the fourth group but would be useful for most apps and essential for many.

Group 1: Minimum for PCA interlock. The minimum interface to enable a PCA safety interlock application are the Pump UDI, Drug Name, and Pump Stop command. With these, an app could stop opioid drug infusions when the patient's vital signs (as reported by other devices) indicated a problem. The app would not know whether the pump was running or the dose of the drug and would not be able to confirm that the pump had stopped, but the basic functionality of stopping the pump would be possible. An interlock app can not stop a pump unless it supports a pump stop command. This command should completely stop the infusion (no KVO) and put the pump into an alarm state where the pump clearly indicates that it has been stopped by a remote command and name the source of the command. For multichannel pumps, other channels of the pump may continue running.

The pump UDI is needed to distinguish between devices. There may be several pumps of the same type connected to a patient. The drug name allows the application to stop the correct pump or channel. Patients may receive other infusions at the same time as PCA, and it may be unsafe, even life threatening, to stop these infusions. It would be possible to build a PCA safety interlock with a pump that does not supply UDI or the drug name, for instance by having the caregiver manually pair the application with the specific pump, but this introduces significant hazards. The added effort of providing UDI and drug name to a pump that has an electronic interface is minimal and the patient safety benefits are quite substantial.

A fourth data element, the current flow rate for the channel, would provide positive feedback that the pump had acted upon the stop command. The stop command will trigger an alarm on the pump, and the app may also sound an alarm or send an alarm signal to the nurses' central station. If the application learned that the pump had not stopped as commanded, all it could do would be to sound an alarm – which it will have already done anyway. It would be useful for the application to be able to warn the caregiver that the pump had not stopped, but this is not essential to enable the basic performance of the PCA safety interlock app.

Group 2: Patient Identity. Most existing pumps do not track patient demographic data. Some have limited capability to enter a patient name or MRN through the user interface, but do not typically output this electronically. If this information was available, apps could confirm that they were communicating with a pump on the patient they expect. Newer pumps with barcode readers may soon have the capability to read patient ID barcodes, but it remains to be seen how common this will be. Reading patient barcodes is not always easy, especially if the patient is asleep or otherwise unable to cooperate. The most useful electronic interface for infusion pump patient information would likely be the ability to push patient ID to the pump, including a photograph of the patient that could be displayed on the pump's UI.

Group 3: Pump Program. The pump program includes the data elements that specify the medication and how the pump should deliver it. Multi-channel pumps are very common and may be user-configurable. For instance Alaris Medley pumps are modular, and clinicians can assemble modules for large-volume infusion, syringe infusion, or PCA at the point of care. The only visible differences between the syringe and PCA modules are the addition of a patient button and a locking cover on the PCA module. The Alaris controller supports up to four modules. The pump's network interface is housed in the controller and communicates data from all of the connected modules. The clinician must ensure not only that the pump is programmed correctly, but also that the correct channel is programmed. When the patient has a dozen or more channels of infusion feeding into several ports, ensuring that the correct drug container is on the correct physical channel, programmed correctly, and attached to the right place on the patient is challenging.

Drug Name, Drug Concentration, Volume to be Infused (VTBI), and Infusion Rate are all self-explanatory, but there are some subtleties. Drug names should be chosen from a standardized set and displayed using a format like TALLman, which reduces reading and entry errors. TALLman uses mixed capitalization to emphasize differences in drug names; for instance DOBUTamine instead of DOPamine. Drug concentration errors have been caused by the inability of pumps to accept entries in the appropriate units. For example, some chemotherapy drugs may have concentrations measured in nanograms/liter. If the pump only accepts programs in milli- or micrograms, caregivers will work around this by entering it deliberately using the wrong units and then trying to remember the conversion factor.

Drugs may be delivered continuously or pushed at a high rate for a short time in what is called a bolus. PCA is based on the idea of infusing a bolus when the patient presses a button. The relevant pump settings are the Bolus VTBI, the volume to be infused; the Bolus Rate, the rate to infuse the drug; and how long to wait after finishing a bolus before the next one can start, the Bolus Lockout Interval.

Group 4: Pump Status. Pump status includes measurements of the current state of the pump including where it is in each program. Key data elements here include the time remaining for each infusion, alarm status, warning or alert status, bolus state (delivering, locked out, time remaining until bolus is unlocked), and a pump data log including internal status events like button presses.

Group 5: Programmability. The final group, programmability, adds the ability to remotely control the pump. There are two categories of data elements we want to be able to set remotely: patient information and the pump program. Patient demographics include the patient's identity, and vital statistics such as height, weight, and drug allergies that are essential for programming the pump. The pump program includes the elements in Group 3; the difference here is that these elements are externally set rather than just being reported.

5.2 Consistency and Completeness Checks for Device Models

Consistency for static models means that the models agree on assignments. More formally, we define consistency to mean that the models have compatible assignments: e.g., $x == 3$ and $x < 5$. There must be at least one possible set of values for all variables that satisfies all constraints.

We say that a set of static device models is sufficiently complete for a given set of device requirements if it contains assignments for all variables used in the requirements.

5.3 Checking an Application's Device Requirements

An application's device requirements will be checked multiple times during the lifecycle of the application. Checks are triggered when an application starts, when a device is added or removed, when a (relevant) device attribute changes, and even, for some applications, every time a data value is updated.

Device requirements are checked when the application starts. This initial check determines whether the set of connected devices at run time is sufficient for the application. If the requirements are not met, this may indicate that necessary devices are not present or that the patient is in a state that the application can not handle. The check at start time allows the application to inform the caregiver that there is a problem and will prevent the application from starting. If the problem can be resolved, for instance by connecting another device, then the application can be restarted and may pass the check the second time.

Device requirements that hold under a particular set of devices may no longer hold if a device is added or removed. Thus, a check is triggered when devices join or leave the network, possibly with a short delay to prevent problems with a device that fails in a way that causes it to join and leave many times per second.

A device's settings can significantly change the data outputs even if the patient's status is unchanged. When settings like EKG filter selection are important to the operation of the application, application developers can specify particular settings that are explicitly allowed or blocked. If a device starts with an allowed setting but a caregiver changes the setting to one that is blocked while the application is running, this check will inform the application that the setting is no longer acceptable. The application may continue running, perhaps with degraded capabilities, alert the caregiver to change the setting back to one that is acceptable, or halt with an alarm.

Similarly, applications may incorporate algorithms that are only valid for particular ranges of vital signs. An arrhythmia detection algorithm might only work if the heart rate is between 30 and 300. If the monitoring device reports a value outside of this range, whether it's a valid reading or an error, then the device requirement check will fail and the application will receive a notification that the vital sign is outside of the acceptable range.

Checking Device Requirements. Device requirements can be checked at several points in a workflow: at application start time, whenever a device is added or removed, when a device attribute changes – including setting changes or addition or removal of device modules, or every time any data value is updated. These options are listed roughly in order of increasing computational cost. Checking device requirements, especially behavioral requirements, every time a data value is updated may not be possible. Data values can be updated many times per second and checking the requirements may take several seconds. In OpenICE, we leave it up to the application to determine which device requirements must be checked when. As an example, the infusion safety interlock requires heart rate, oxygen saturation, and respiratory rate and requires fresh data elements at least every five seconds. The application subscribes to these data elements for the relevant patient and maintains a timer that will trigger an alert if no updates are seen for five seconds. In this example, the device requirements are checked continually as the application runs. This accommodates devices that do not specify their data update frequency and also works correctly with unreliable networks – the safety interlock works when the network is working well enough and fails safely when the network performance is not adequate.

There are two types of device requirements and two types of device models. Device requirements can be a set of arithmetic constraints or a set of CTL constraints. Arithmetic constraints operate on variables and numeric constants, evaluate to a boolean, and are assumed to be side-effect free. The four combinations of device requirement and device model types are shown in Figure 5.3.

	Static DM	Behavioral DM
Constraint DR	arithmetic	make CTL, model check
CTL DR	make automata, model check	model check

Figure 5.4: Combinations of Device Model and Device Requirements Types

When checking a set of constraint device requirements against a set of static device models (with no behavioral device models), it is sufficient to replace the variables in the requirements with the corresponding static assignments from the device models and then evaluate the resulting expressions. If the device model asserts that $x = 2$ then the requirement $x \leq 3$ is replaced with $2 \leq 3$ and evaluates to *true*.

To check CTL device requirements against the same set of static device models requires first generating a set of automata from the device models and then checking the device requirements against the generated models.

Checking Arithmetic Constraint device requirements against behavioral device models requires making CTL formulas from the device requirements. These CTL formulas assert that the expression in the arithmetic constraint always holds in the model. These formulas are then checked using model checking.

Finally, CTL device requirements are checked against behavioral models using model checking, specifically the UPPAAL model checker.

5.4 Proving System Safety Properties

Much of this work has been setting up the process of checking system safety properties. Our end goal is to enable the production of safe and secure interoperable systems of medical devices. The system safety properties encode the constraints that the application developers believe must hold in order for the application to be safe and secure. Previous sections describe the architecture and components necessary to facilitate writing and checking these properties. With these pieces in place, we can now discuss how the properties will be checked. The system architecture and development process are designed to make it as easy as possible to check the system safety properties. If the checking process seems simple and straightforward, then the enabling steps that have gotten us to this point have been successful.

Checking system safety properties requires four models. These are the device models, which are provided by the device manufacturers, and the clinician, patient, and application models, which all come from the application developer. We also need the system safety properties, which also come from the application developer. This is a lot of work for the application developer. Applications in this framework are very likely to be regulated medical devices, and as such the application developers must shoulder the burden of demonstrating that they are safe and secure. This framework provides means for structuring and formalizing their argument, and a methodology for run-time checking of critical safety properties in an interoperable environment. Safety properties for applications may hold or fail depending on the configuration of the connected devices - configurations change as the devices are used in normal practice. Checking these properties as the system is assembled and used allows us to prove that the critical properties hold in the actual use environment. This allows us to make safety arguments about the devices that rely on conditional configuration details that are not known in advance of use, allowing for the mitigation of hazards that are otherwise uncontrolled. This process is shown graphically in Figure 5.5.

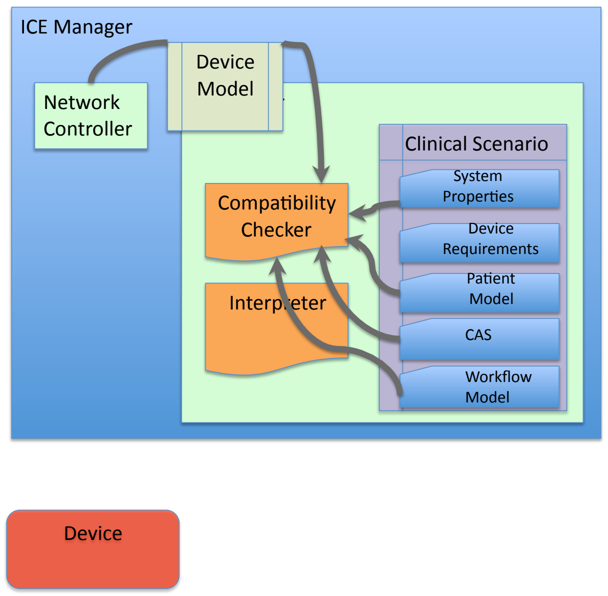


Figure 5.5: Checking System Safety Properties

The application developer is responsible for most of the key pieces, including the System Safety Properties, the Clinician Workflow Model, the Patient Model, and the App Model. Device Models are provided by the devices. Some of the models may be provided in multiple forms. We transform them into EFSMs if they are provided in another form, and later convert the EFSMs into the input format of the model checker. Clinician Workflow Models and App Models are all provided as EFSMs. Device Models may be either behavioral, which are provided as EFSMs, or constraint, which are used to generate a simple EFSM containing the constraints. System Safety Properties are provided as a set of CTL formulas. The patient model used here is a state machine model. Models with continuous dynamics may be used in conjunction with a discrete model as shown in [65], which also includes modeling the effect of network components.

When generating EFSMs from constraints, there's a coverage problem. A constraint can be of a form like 'x is greater than three'. When an EFSM is generated, a value must be assigned to x. We can't pick all possible values unless x has a very limited range, so we have to pick one or run a few trials with a few choices. We might, for instance, choose four, as the first integer that meets the constraint, but we can't ensure that there is not a value of x where the constraint does not hold.

We could generate an EFSM where x is assigned a value from its range nondeterministically, then use model checking to see whether the system properties hold under any possible assignment of x. This would work, but would dramatically increase the state space of the model, increasing the time necessary to check the properties. This is also not a good match for how constraints are intended to be used, and how they are used in current medical devices, where they are not randomly assigned. Constraints, rather, are single numbers reflecting a particular setting and change only rarely in use. They must be communicated by the device because they are changeable, but the value at any particular time can be communicated by the device. If a device has more complex behavior around a constraint, the device should send a behavioral model.

5.5 Discussion

This chapter focuses on proving properties about a system written using the CAML language described in Chapter 3 and the architecture described in Chapter 4. We start by describing system safety properties and the process of creating safety properties from a hazard analysis of the application in Section 5.1. This discussion includes examples drawn from the patient-controlled analgesia use case described in more detail in Section 6.2. Section 5.2 talks about checking the consistency and completeness of device models

Device requirements capture assumptions and requirements that the application developer has about the medical devices that the application will interact with. Section 5.3 includes discussion of how and when device requirements are checked, pointing out that device settings and capabilities can change while they are in use. Device requirements can be given as either sets of arithmetic constraints or as CTL formulas. Device models can be static or behavioral, and we handle all of these combinations, as shown in Figure 5.3.

Finally, we prove system safety properties over the assembled system by creating an UPPAAL model, as discussed in Section 5.4.

Currently, device manufacturers create and maintain hazard analyses for their devices and do not share them publicly. There is potential for improvements to device and patient safety through the creation and maintenance of open, shared hazard analyses along the same lines as our Generic Infusion Pump hazard analysis [14]. Safe interoperable systems will require applications and other devices to share information about their hazard analysis and hazard mitigations. This thesis suggests one specific technique of checking device requirements and system safety properties for applications, devices, workflow, and patient models that can be captured in CAML. Building application and system hazard analyses and safety properties should be generalizable beyond this to other systems that use medical devices as interoperable components.

Limitations. This Chapter is about checking device requirements and system safety properties against the components of the architecture described in Chapter 4. The kinds of properties we can prove about the system and the ways we can prove them inherit some of the limitations of the language and the architecture. These limitations also affect the process of checking the system safety properties. Some of the implications of these limitations will affect the case studies in Chapter 6. Limitations around timing, network modeling, and matching semantics between the model and implementation are discussed in more detail in that Chapter.

In this work, we only consider single applications. It is likely that clinicians would want to use multiple applications simultaneously on a patient. As discussed in Chapter 4, this requires work to define the composition of multiple patient models and resolution of contention for shared resources before we can prove properties against a system created by composing multiple CAML systems.

We check properties against the UPPAAL model and want them to hold in the generated Java code. As discussed in Chapter 3, this requires that the UPPAAL model shares the same semantics as the Java code running in its execution environment, typically a Java virtual machine running under an operating system. Preserving properties in different execution environments will require careful construction of translations, configuration of target environments, and validation, likely through both formal proofs of correctness and extensive testing of implementations, before being used for patient care.

Gap Analysis. In addition to the problem of composing CAML models of multiple applications, there are also the problems of merging device requirements and system safety properties. Device requirements might be identical (both systems require heart rate measurements once per second), compatible (one system requires measurements at least once per second, another at least once every two seconds), or incompatible (one system requires a pulse oximeter averaging time of 3 seconds, another requires an averaging time more than 5 seconds). Safety properties are particularly tricky because applications may interact in unintended ways through the patient. For instance, an application managing blood pressure may affect heart rate, which might be an input to another control application. If the patient model for the blood pressure application does not include heart rate, the interaction would be difficult to detect automatically. More work is required to provide a theoretical basis for composing device requirements and system safety properties.

The hazard analyses and safety properties from the CAML systems being composed may also be incompatible in various ways. For example, consider a patient who is receiving pain medication while also on a ventilator. It would be reasonable to have one application manage the infusion of pain medication and another application that optimizes ventilator settings for the patient. The pain medication application would be concerned with preventing overdoses leading to respiratory depression. One common way to do this is by monitoring their SpO_2 using a pulse oximeter and stopping the pump when the patient's SpO_2 drops below a threshold. A safety property for this application would be "if SpO_2 is less than 85, the pump must be stopped". The application managing ventilator settings would be responsible for adjusting the rate and volume of breaths and many other ventilator settings, likely within some clinician-specified upper and lower bounds. One of the settings, FiO_2, specifies the fraction of inspired air that must be oxygen. Ventilators can adjust the percentage of oxygen delivered to the patient between 21% (what is found in room air) and 100%. The ventilator application will adjust these settings (within bounds) to keep the patient oxygenated without injuring them. A safety property for the ventilator application could be "increase FiO_2 (up to configured maximum) to keep SpO_2 above 90". If the pain management and ventilator applications are used together, we have the possibility of the ventilator application masking the onset of an overdose by increasing the FiO_2 so that the pain management application doesn't stop the pump. The patient could receive too much medication, start to deteriorate, and have their condition hidden by the automatic adjustments by the ventilator application. If the patient received a severe enough overdose, they would decompensate to the point where even the maximally elevated FiO_2 could not keep their SpO_2 above the pain management application's threshold and it would alarm, but this alarm would come much later than if the two applications had not been working at cross purposes.

We need a theoretical framework for composing and checking the compatibility of safety requirements. In the short term, applications could be evaluated for compatibility manually, but this kind of pairwise evaluation does not scale and leaves open the possibility of unexpected emergent behavior as the third, fourth, or further applications are added.

Future Work. In Section 5.4, we suggest that devices could resend their variable, and perhaps behavioral, device models when settings or other metadata change. More work could be done to refine this notion to send only changes that are necessary for verifying the system safety properties of interest. This would reduce network traffic and load on the supervisor, but would require the devices to know something about the system safety properties and be capable of determining which data elements needed to be sent. Supervisors could send devices a list of variables of interest, but mechanisms for creating these lists from the safety properties and device requirements and updating them through the application lifecycle would need to be developed.

We make an assumption that a single terminology is used for all of the models and properties and that terms are used in the same way by application developers, creators of hazard analyses and system properties, and all of the other contributors to the system. Terminology and ontology development remains an active area of ongoing work.

Integrating the CAML models and verification process described here into OpenICE would require extensions to both. OpenICE uses a publish / subscribe middleware to communicate between processes. This middleware can be configured to have semantics like the communication channels in CAML, but it would also be useful to add communications channels to CAML that match the semantics of the channels used in OpenICE.

Chapter 6

Case Studies

We have implemented several workflows to show the value of medical interoperability, to gather and record safety-related hazards, and to demonstrate the need for and benefit of checking device requirements and system safety properties in interoperable medical systems. In this Chapter, we present two of these case studies. The first, synchronizing an x-ray with an anesthesia machine ventilator, includes a tight real-time control loop including device control. The second case study of patient-controlled analgesia (PCA) brings together the pieces described in previous chapters to show a safe way of integrating a patient monitor with an infusion pump to create smarter alarm systems and a safety interlock.

Each of these case studies includes a description of the medical workflow because it is important to understand the workflow in order to model it, generate safety properties, and understand the capabilities of the devices that are used. Without this detailed understanding, it is not possible to prove the safety of these systems. Fortunately, not everyone involved in building the system has to understand every part. The application developers need to know the workflow they intend to operate within, the patient model, and what they intend their application to do. The device manufacturers need to understand the capabilities of their devices, but do not need to know the clinical workflow or patient model details, though of course it is helpful for device manufacturers to understand how their devices will be used so they can construct a better hazard analysis and, ultimately, devices that better fit their users' needs.

The case studies presented here are complementary. Both systems were built to illustrate the possibilities of interoperable medical devices and to exercise the architecture presented here, but they emphasize different parts of the problem. The Xray/Ventilator synchronization case study is an open-loop control problem with safety properties focused on triggering the Xray at the correct time. Code generation from the CAML model is used to create the executable program. Model checking was used to understand the system dynamics and safety properties, but the code for the implementation was manually written rather than being generated. There was still great value in modeling the system before implementing it, and an implementation with better-defined communications semantics would allow for more confidence in generated code.

System Component	**X-Ray / Vent Sync**	**PCA**
Device Models		✓
Device Requirements		✓
SSPs	✓	✓
Caregiver Model	✓	
Patient Model	✓	✓
CAS	✓	✓
UPPAAL Translation	✓	✓
Java Code Generation	✓	

Table 6.1: System Component Use in the Case Studies

Table 6 shows a breakdown of architectural component usage in the two case studies. This gives a good overview of what was used in each of the studies and how the studies complement each other. The PCA interlock system builds on the techniques developed for the Xray/Ventilator study. The PCA system is designed to be interoperable, and is an example of a physiologic closed-loop system with safety properties dependent on the timing characteristics of each component.

The X-Ray / Ventilator case study consists of a CAS, caregiver model, and patient model and uses the UPPAAL translator and the UPPAAL tool to check some SSPs. Java code generation is used to create an executable that can be used with real hardware devices. The system includes fixed models of the key devices and a discussion of some variable-list type device requirements, but it was designed and built as an interconnected but not interoperable system.

The PCA case study does include behavioral device models and device requirements, but does not use a caregiver model or Java code generation. As a closed-loop safety interlock, operator intervention is not intended and so the caregiver model is not useful. The implementation of the system was hand-written as part of the OpenICE implementation, and so code generation was not used.

6.1 Synchronizing an X-Ray with an Anesthesia Machine Ventilator

This use case was inspired by a real event documented by the Anesthesia Patient Safety Foundation[55]. It illustrates a potential problem with the way x-ray images are usually taken during surgery.

> A 32-year-old woman had a laparoscopic cholecystectomy performed under general anesthesia. At the surgeon's request, a plane film x-ray was shot during a cholangiogram. The anesthesiologist stopped the ventilator for the film. The x-ray technician was unable to remove the film because of its position beneath the table. The anesthesiologist attempted to help her, but found it difficult because the gears on the table had jammed. Finally, the x-ray was removed, and the surgical procedure recommenced. At some point, the anesthesiologist glanced at the EKG and noticed severe bradycardia. He realized he had never restarted the ventilator. This patient ultimately expired.

It is common practice to stop the anesthesia machine ventilator for a short time when an x-ray is required during surgery. This ensures that the patient's chest is not moving when the exposure is made and does not harm the patient provided that the ventilator is restarted promptly. Difficulties arise only if the ventilator is not restarted for some reason. This kind of problem can be mitigated by using interoperable devices and a synchronization application. If the anesthesia machine ventilator can synchronize with the x-ray, then it is no longer necessary to stop the ventilator to make the exposure.

Medical devices generally have proprietary interfaces which are only documented in technical manuals or other material not openly available. We were fortunate to have the cooperation of Dräger, the manufacturer of the ventilator we used. The interface of the ventilator was designed to be used for diagnosis of machine faults and to send data to the electronic medical record, not as a source of real-time status information. Thus, it runs at a relatively slow rate, and the low maximum sample rate (5 - 10 samples per second) was the limiting factor in designing our control algorithm.

Our risk assessment process started with a hazards analysis. We documented potential hazards and their mitigations, and used them in writing device requirements and safety properties. The risk analysis process and how we used hazards to derive safety properties with which to verify the system is described in Section 6.1.2.

Software development for this case study started with informal system requirements which were used to build a state machine model of the desired system behavior. We checked this model for safety properties using model checking software and then generated code from the model to produce the supervisor.

6.1.1 Xray / Ventilator Synchronization System

This architecture of this system follows closely that of the ICE standard [16]. The major components are a set of medical devices, a network controller, a supervisor, the patient, and a caregiver. Medical devices connect to each other and the supervisor through the network controller. The devices' connections to the network controller may go through physical adapters and data format converters if their connectors and formats are not directly compatible. The network controller may also connect to an external network such as a hospital information system. The supervisor runs the control software for the system.

Our initial system implementation, which follows the conceptual architecture, is shown in Figure 6.1. The devices we used were a Dräger anesthesia machine ventilator and a simulated x-ray machine. The role of network controller and equipment interface to the simulator is filled by the LiveData RTI software program. LiveData Inc. is a company which produces software to integrate medical devices for common display data. For this implementation, we worked with LiveData to connect the ventilator and simulated x-ray.

The supervisor program implementing the synchronization algorithm runs on the same computer as the LiveData RTI software. Finally, the patient was represented with a physical lung simulator consisting of a bellows and spring. While a simple lung simulator does not capture all the nuances of a real patient, it is sufficient for this application. Lung movement is the factor we can control in taking a clear x-ray, and a supervisor which can synchronize with a simulated lung can be expected to do the same with a real patient.

This initial implementation was an interconnected medical device system rather than a fully interoperable system. An interconnected system is one in which devices are functionally connected through an interface. It differs from an interoperable system in that the devices are hard-coded. The system is built around specific devices and will not operate with other, similar devices. This initial project was useful for identifying functional and non-functional requirements for the standards in progress and illustrating the benefits of the interoperability work.

We later reimplemented this algorithm on the OpenICE platform as an interoperable system. This system still used a simulated X-Ray machine, which is safer in a lab environment than a real one, and was able to synchronize with any source of respiratory flow information for the patient. We tested this implementation with a variety of ventilators and anesthesia machines. This application is included as one of the standard demonstration apps in the OpenICE repository. Work in this Section was developed over a long period of time with several sets of collaborators as documented in [7], [5], [6].

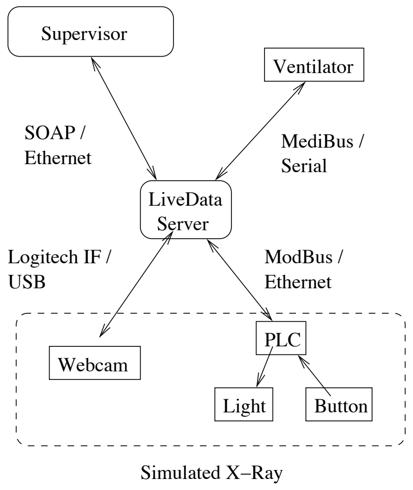


Figure 6.1: Overview of the System

Synchronization Algorithms. The supervisor uses information from the ventilator to decide when to trigger the x-ray. The synchronization algorithm defines exactly how this decision is made. Figure 6.2 shows the respiratory cycle graphed as pressure over time. The pressure increases until the end of inspiration (at time *Tinsp* after start of breath), at which point it drops off quickly through expiration. There is usually a pause between the end of exhalation and the start of the next breath. For this case study, we want to support taking an x-ray when the 'lung' was not moving significantly. This occurs when the patient is relatively still at the peak of inspiration or between the end of expiration and the start of the next breath. An exposure is possible if the time the patient is still exceeds the time needed for the exposure plus the latency between triggering the x-ray and the actual exposure.

Synchronization Method 1: Dead Reckoning. The first method used to determine when to trigger the x-ray is simple dead reckoning using the time of last breath, time of inspiration, and frequency. The variables used for this method are shown in Figure 6.3.

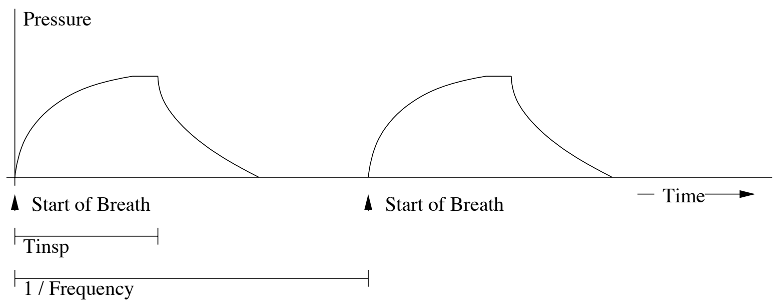


Figure 6.2: Respiratory Cycle

All times are in seconds.

name	description
T_{now}	current time
T_{lb}	time of last breath
T_{nb}	time of next breath
T_{δ}	a small offset time to accommodate jitter
$T_{trigger}$	time to send trigger signal to the X-ray
T_{exp}	time of X-ray exposure
$freq$	frequency, breaths / minute
$flow$	instantaneous flow rate

Figure 6.3: Variables for dead reckoning

If we know the time of the start of the last breath and the frequency of breathing, then it is trivial to calculate the time of the start of the next breath.

$$T_{nb} = T_{lb} + 60/freq \tag{6.1.1}$$

There is likely to be time to trigger the x-ray just before start of the next breath, as long as the patient has finished exhaling before the start of the next inhalation.

$$T_{trig} = T_{nb} - T_{exp} - T_{delta} \tag{6.1.2}$$

We can check whether the patient has actually finished exhaling by sampling the instantaneous flow rate just before the start of the next breath. If it is close to zero, then the patient is not inhaling or exhaling and is still enough to allow taking the x-ray.

1. Get values of the variables T_{now}, T_{lb}, $freq$
2. Calculate T_{trig}
3. Sleep for $T_{trig} - T_{now}$ seconds
4. Wake up and sample $flow$
5. If $flow = 0$, trigger X-ray

   else, start over

This method of synchronization makes many assumptions. The most critical assumption is that the respiratory frequency is not going to change between the last breath and the next one. If it does, or if the system setup changes in other ways, this method of synchronization will not work. The check of instantaneous flow rate should prevent the system from triggering the x-ray when the patient is moving, but the system may not be able to take an image in situations where a different synchronization method would allow an exposure.

Synchronization Method 2: Dynamic. Another way to calculate the trigger time is to sample the real-time flow rate rapidly enough to build a picture of the flow graph. We experimented with two techniques for doing this. The variables used in the following descriptions are listed in Table 6.4.

We originally envisioned sampling at a high enough rate to be able to integrate the total flow volume by multiplying the sampled flow rate by the time interval of the samples.

name	description
$flow$	instantaneous flow rate
T_{flow}	time of last $flow$ sample
$S_{current}$	value of current flow sample
$T_{current}$	time of current flow sample
S_{last}	value of last flow sample
T_{last}	time of last flow sample
$slope$	calculated slope value
$Threshold$	slope threshold

Figure 6.4: Variables for dynamic synchronization

This would allow the supervisor to trigger the x-ray at the right time no matter what changes were made to the ventilator's programming or how the patient reacted. It would also allow synchronization with spontaneously breathing patients, and enable detection of coughing, which would allow the algorithm to wait to synchronize until the breathing pattern stabilized. However, the ventilator was not able to provide samples at a high enough rate to enable this method to be used. The SOAP server and interface introduced additional latency and jitter into the samples, which further reduced their usefulness for this purpose. Using OpenICE reduced the latency and jitter of the data transmission but could not improve the sample rate of the ventilator. We hope to revisit this application with higher resolution devices when they become available.

Our second idea was to use the slope of the flow signal to find when inspiration is about to end. This meant taking two or more samples, calculating the rate of change of the flow rate between them, and triggering when this rate of change was low enough. The problem we ran into here is that the flow graph tails off very rapidly, making it unlikely that we would get even a pair of samples in the short time when the breath is about to end. The low sample rate made this problem worse.

```
1. prime $S_{last}$, $T_{last}$, $S_{current}$, $T_{current}$ with two consecutive samples
2. $S_{last} = S_{current}$
3. $T_{last} = T_{current}$
4. $S_{current} = flow$
5. $T_{current} = T_{flow}$
6. $slope = S_{current} - S_{last}/T_{current} - T_{last}$
7. if $slope < Threshold$ and $flow$ is near 0, trigger x-ray
   else loop back to 2.
```

In the end, we found that dynamic synchronization is possible only at relatively low respiratory rates – under about 8 to 10 breaths per minute. The dead reckoning method functions at much higher rates, up to approximately 25 to 30 bpm depending on the other ventilator settings. The supervisor program for our demo checks the respiratory rate and chooses whether to use the dynamic or dead reckoning method accordingly.

Alarms. The system should not trigger the x-ray if the ventilator has active alarms. The ventilator will take care of displaying the alarm condition to the caregiver and sounding alarms, so the supervisor just has to detect that the ventilator has active alarms and not trigger the x-ray on that respiratory cycle. It does this by getting a summary of all active alarms and warnings from the ventilator. If the list of active alarms is not empty, then the supervisor will not trigger the x-ray.

This technique is easy to implement and covers the most common situation where the alarm sounds sometime before the supervisor decides to trigger the x-ray. This is sufficient for the demo, but an implementation with a real x-ray machine and a real patient would have to take into account factors such as the alarm being raised after the supervisor checks the alarm status but before the exposure is made.

In the case where this happens, many conditions which would cause a ventilator alarm will not affect the synchronization algorithm. These include alarms like low gas levels, overpressure, some sensor failures, etc. Any alarm that does not indicate an unexpected change in ventilator settings will not stop the supervisor from being able to synchronize. Alarms for major mechanical malfunctions are very rare, but would indicate conditions where we would not want an exposure to be made – though any failure which stopped the ventilator from operating would mean that the patient's chest was not moving.

The biggest problem with triggering an x-ray exposure during an alarm is not that the image would be blurred, but that the safety of any caregivers responding to the alarm could be compromised. Caregivers are also protected by the use of a 'dead man switch' that the x-ray technician holds during the exposure. If the switch is released, the x-ray will not be taken. The time interval where there was an active alarm and the exposure was being made would be a fraction of a second, but this should be taken into account in the risk management process. Any system using a real x-ray machine would also need to take into account alarms from the x-ray, and any system using medical devices which are capable of pushing alarms rather than having them polled (as we did with this ventilator) would also need to consider possible race conditions between the alarm handling and synchronization parts of the supervisor.

6.1.2 Modeling, Verification, and Code Generation

The software for the synchronization app is the key element of the system. The app's role in this demo is to gather data from the ventilator, decide when to trigger the x-ray, and send the signal to the x-ray machine at the correct time. The app interacts with the caregiver to get input such as whether to make the exposure during inspiration or expiration and to provide the caregiver with status information and, ultimately, with the x-ray image.

The functioning of the synchronization app is critical to the safety of the system, so we devoted a significant amount of time and effort to ensuring its correctness.

The app software development process started with gathering informal requirements. These requirements were collected during discussions with caregivers and biomedical engineers and included functional requirements such as “when the exposure is made, the red light on the x-ray box should light up” and safety requirements like “the caregiver’s x-ray trigger button must be held down for the x-ray exposure to be made”. These requirements were refined and expanded upon throughout the development process. For instance, when we started development we did not know that we would need a dead-reckoning synchronization algorithm in addition to the dynamic method and thus did not initially include any requirements about when the supervisor should use one or the other of these techniques.

We built and verified a state machine model of the app that meet essential safety properties. We used the model to generate Java code which then ran the demo. This development process is described in more detail in the following sections.

Verification. We began by modeling the app as an extended finite state machine (EFSM). Once the system was modeled as a state machine, we used a tool to translate it into the input format for the model checker UPPAAL. The model checker was used to simulate the system, to test the system for general properties like deadlock, and to test more specific properties. These activities suggested changes to the EFSM specification, and the process went though several iterations. Eventually, we produced an EFSM specification which satisfied all the safety requirements.

The safety requirements for the system were gathered by talking with clinicians and working though an informal hazard analysis process.

The primary hazard introduced by this system is triggering the x-ray at the wrong time. This could potentially endanger the x-ray technician or other clinicians. Triggering the x-ray when the patient is moving will result in a blurred x-ray and the need to take another exposure, meaning additional radiation exposure for the patient.

Another hazard is that an image might not be taken even though it is possible. This is less serious, since the system will inform the clinician that the exposure was not possible and try again on the next breath. The exposure is delayed slightly, but this is a small cost compared to that of a failed exposure.

The EFSM model of the system was checked for structural properties like deadlock (that the system can't get 'stuck') and for specific safety properties. These focused on when the x-ray is triggered, since this is the single safety-critical action the system takes. We checked that the trigger signal was sent only at the correct time and that the system would not trigger unless the flow rate reported by the ventilator was near zero.

$$\text{AG xray} = \text{exposing implies } T_{now} = T_{nb} - Texp - T_{\delta} \tag{6.1.3}$$

Formula 6.1.3 is used for checking the system when it is being used to make an exposure at the peak of expiration (the lung is empty) in dead reckoning mode. This specification says that whenever the x-ray machine is in a state where it is exposing (AG xray = exposing) the current time must be the time of the next breath minus the exposure time minus a small offset ($T_{now} = T_{nb} - Texp - T_{\delta}$). This means that if there is any possible way that the EFSM could have the x-ray in the state 'exposing' when it is not that time, the model checker will show it as a counterexample. Similar formulas are used for checking exposure times for inspiration.

$$\text{AG xray=exposing implies } flow <= flow\_threshold \tag{6.1.4}$$

Formula 6.1.4 states that when the x-ray is exposing, the instantaneous flow rate must be less than the flow threshold. This threshold is defined to be low enough that the lung will not be moving enough to blur the image, but also high enough to allow an exposure when there are very small movements.

Code Generation. The final EFSM specification was used to automatically generate Java code which was used in the demo implementation. The demo includes a handwritten GUI frontend which is the user interface and the supervisor application, which is largely generated code. The generated code interacts with some handwritten functions which perform low-level actions. For instance, the model simply uses values like $flow$, while the generated code replaces references to such variables with calls to handwritten library functions which actually provide the values.

Implementations. The application starts with a screen describing the clinical use case. This is followed by giving the user a choice of taking an image at the peak of inspiration (when the lungs are full) or the peak of expiration (when the lungs are empty). The user is asked to confirm their choice and taken to a screen describing the image-taking process. The user is asked to play the role of an x-ray technician and to pick up a physical button which they will hold while the exposure is made. In a non-synchronized x-ray, this button would trigger the x-ray directly. In our system, the button is held down to give the system permission to make the exposure. The clinician holds the button for several seconds while the system waits for the lung to reach the proper phase of respiration and the system checks to make sure the button is held before taking an image. If the clinician decides that it is not safe to make an exposure (e.g., if someone walks into the room), they can simply release the button and no exposure will occur. This allows us to keep a human in the loop as an additional safety precaution. Assuming the button is held down, when the lung reaches the proper phase the exposure is made and the webcam image is displayed on the screen.

6.1.3 X-Ray / Ventilator in CAML

The X-Ray / Ventilator use case includes seven state machines that run in parallel. These are the models for the top-level application, the four synchronization modes, the caregiver, and the patient. Four of these models are shown below. The figures are the result of visualizing the translated CEFSM in the UPPAAL tool.

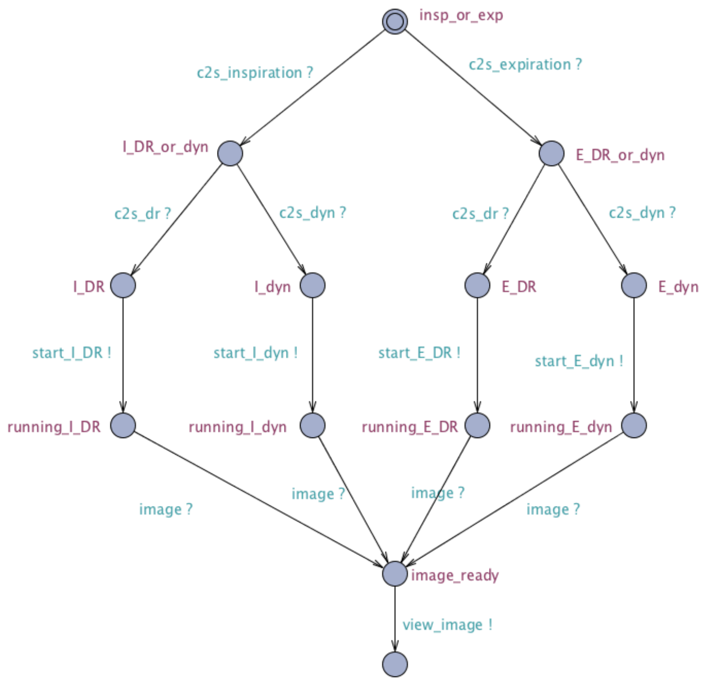


Figure 6.5: X-ray / Ventilator Example: Top Level Application Model

6.1.4 System Properties for X-Ray / Ventilator

The X-Ray machine and the Ventilator are both critical devices in that the failure of either may injure or kill the patient and harm bystanders. Failure to ventilate has obvious potential for harm, as does an excessive dose of x-rays. A subtler problem common to many diagnostic devices is that the failure to take an x-ray, or taking an x-ray at the wrong time in the respiratory cycle may lead to improper diagnosis or treatment of the patient.

Basic Safety Properties:

1. The ventilator must never stop ventilating for more than 30 seconds.
2. The patient must not receive more than one x-ray exposure per activation.

Basic Effectiveness Properties:

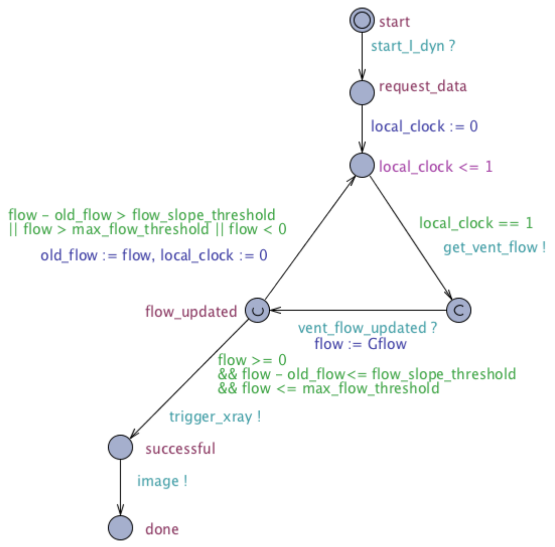

start
start_I_dyn ?
request_data
local_clock := 0
local_clock <= 1
flow - old_flow > flow_slope_threshold || flow > max_flow_threshold || flow < 0
old_flow := flow, local_clock := 0
local_clock == 1
get_vent_flow !
flow_updated
vent_flow_updated ?
flow := Gflow
flow >= 0 && flow - old_flow<= flow_slope_threshold && flow <= max_flow_threshold
trigger_xray !
successful
image !
done

Figure 6.6: X-ray / Ventilator Example: Supervisor Inspiratory Dynamic Model

1. X-ray must be triggered at the proper time in the respiratory cycle
2. X-ray must be taken with the proper beam strength and exposure time

These basic properties lead to a more detailed list of properties specific to the system used for the case study:

The system should not deadlock: $\forall\Box(!\text{deadlock})$

The system should have enough time to take an image:

$$\text{Trigger send latency} + \text{xray trigger time} + \text{xray trigger latency} + \text{exposure time} \leq$$
$$\text{dwell time where expiratory dwell time} = (60/\text{ respiratory rate} - \text{inhale time} - \text{exhale time})$$
$$\&\&\ \text{inspiratory dwell time} = \text{xray insp hold time}$$

Correctness of ventilator and patient models; the patient can not be exhaling while the ventilator is inflating: $\forall\Box!(\text{ventilator.inflate}\ \&\&\ \text{patient.exhale})$

Correctness of E Dyn algorithm, which should only trigger when flow is positive

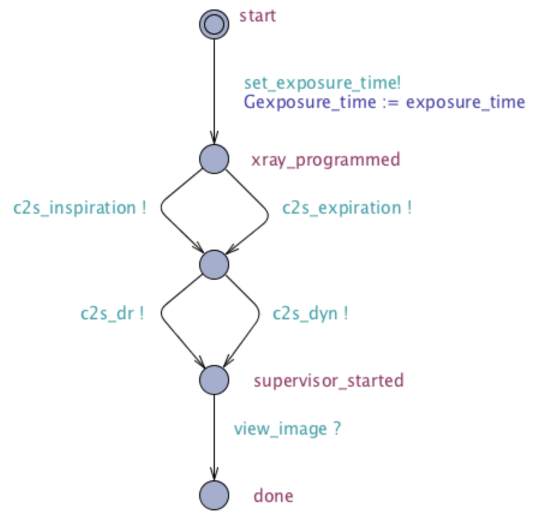


Figure 6.7: X-ray / Ventilator Example: Caregiver Model

and under the max flow threshold: $\forall\Box$(supervisor.running_E_dyn && xray.exposing $\Rightarrow$ Expiration_Dyn.flow $\leq$ Expiration_Dyn.max_flow_threshold && Expiration_Dyn.flow $\geq$ 0)

Correctness of I Dyn algorithm, which should only trigger when flow is positive and under the max flow threshold: $\forall\Box$(supervisor.running_I_dyn && xray.exposing $\Rightarrow$ Inspiration_Dyn.flow $\leq$ Inspiration_Dyn.max_flow_threshold && Expiration_Dyn.flow $\geq$ 0)

Inspiration DR method is possible (success state is reachable): $\exists\Diamond$(Inspiration_DR.successful)

Inspiration Dynamic method is possible: $\exists\Diamond$(Inspiration_Dyn.successful)

Expiration DR method is possible: $\exists\Diamond$(Expiration_DR.successful)

E Dyn algorithm is possible: $\exists\Diamond$(Expiration_Dyn.successful)

E DR algorithm must only allow exposure when the flow is zero: $\forall\Box$(supervisor.running_E_DR && xray.exposing $\Rightarrow$ Gflow $= 0$)

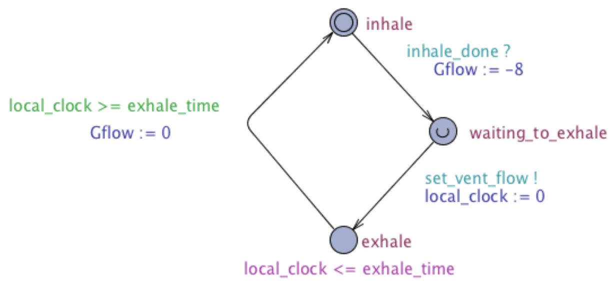


Figure 6.8: X-ray / Ventilator Example: Patient Model

Correctness of I Dyn algorithm, which should only trigger when flow slope is under the threshold:

$$\forall\Box(\text{ supervisor.running\_I\_dyn \&\& xray.exposing } \Rightarrow \text{ Inspiration\_Dyn.flow} - \text{Inspiration\_Dyn.old\_flow} \leq \text{Inspiration\_Dyn.flow\_slope\_threshold})$$

Patient will always eventually exhale: $\forall\Diamond(\text{patient.exhale})$

Patient will always eventually inhale: $\forall\Diamond(\text{patient.inhale})$

I DR algorithm must only allow exposure when the flow is zero:

$\forall\Box(\text{ supervisor.running\_I\_DR \&\& xray.exposing } \Rightarrow \text{Gflow} = 0)$

6.1.5 Device Requirements for X-Ray and Ventilator

These are the application's requirements on the devices, listing the inputs and outputs that the application needs. Any device that provides these data elements should work with the application.

X-Ray:

- Must Provide:

   1. exposure time
   2. image

- Must Accept:

   1. external trigger

- May Accept:

   1. exposure time

Ventilator:

- Must Provide:
   1. instantaneous flow rate
   2. respiratory rate
   3. local clock time
   4. respiratory rate change notification
   5. inspiratory time
   6. inspiratory hold time

6.1.6 X-Ray and Ventilator Device Models

Variable Device Model

X-Ray:

- Provides:
   1. exposure time
   2. image
   3. external trigger latency (optional)
- Accepts:
   1. external trigger

Ventilator:

- Provides:
  1. instantaneous flow rate
  2. instantaneous flow rate (optional)
  3. respiratory rate
  4. local clock time
  5. respiratory rate change notification
  6. inspiratory time
  7. inspiratory hold time

6.1.7 X-Ray / Ventilator Synchronization Summary

We successfully built a system which was able to synchronize an anesthesia machine ventilator with a simulated x-ray machine, demonstrating that the approach is feasible. In the process, we learned lessons for building more general systems. These include the importance of recognizing the limitations of device interfaces in the application algorithm design and the need to have applications that can respond to the changing settings of the devices. We had two synchronization algorithms, one which was more accurate but only usable at low breath rates and a less accurate but faster algorithm for high breath rates. We used formal methods in the development of the application and have presented a methodology for ensuring that the integrated device systems meet their specified safety properties.

This case study started with an unfortunate use case, resulting from the lack of a respiratory pause feature on the ventilator and the ventilator's inability to synchronize with the x-ray machine. The exposure that our demos brought to this problem has led to a proposed change to the international anesthesia workstation standard. Hopefully in the future such changes and the introduction of safe, inter-connected systems will help to improve patient safety.

6.2 Patient-Controlled Analgesia Smart Alarms and Safety Interlocks

Patient-Controlled Analgesia (PCA) infusion pumps are commonly used for pain management in hospitals. These infusion pumps are loaded with an analgesic drug such as morphine, fentanyl, or hydromorphone and can be programmed with a background, or basal, infusion rate as well as a bolus dose. The basal infusion rate is delivered constantly and is selected to be sufficient to control the patient's normal pain level. The bolus dose is an additional quantity of drug that is delivered only when the patient requests it by pressing a button. PCA pumps are also often configured with no basal rate so they only deliver medication when a bolus is requested. The pumps are also programmed with dose limits that are set for the specific patient.

PCA pumps are commonly used because they are an effective means of controlling the patient's pain level and they enable the patient to take some control over their level of medication [56]. They allow the patient to adjust their drug dose to match the level of pain they are feeling at a particular moment in time.

PCA-related Adverse Events. PCA pumps are also associated with a large number of adverse events [32] [23]. The most common type of adverse event is oversedation[56]. An excessive dose of the analgesic can cause neurologic depression which may lead to respiratory depression and eventually respiratory distress. In extreme cases the patient may not be able to breathe adequately, leading to death. Overdoses may have many causes including programming errors [29], the use of the wrong concentration of drug, drug interactions, and PCA-by-proxy.

Programming errors may be caused by confusing drug names, e.g., hydromorphone and morphine or morphine and meperidine [32], by making a mistake in dose or drug concentration calculations [79] [32] or entering the wrong values for bolus dose size, infusion rate, or lockout interval. A common source of error is entering a value that is off by a power of 10 or using the wrong units. For example, entering 5 mL / minute instead of 5 mG / minute or programming a pump with a drug concentration of 1 mG/mL when it is actually 10 mG/mL [32]. [79] discusses a number of cases where patients were fatally overdosed because of an improperly programmed drug concentration.

When someone other than the patient presses the button to request a bolus dose, it is called PCA-by-proxy. Normally if the patient is oversedated they are unable to press the button to get another bolus dose. If someone else presses the button, this safeguard is bypassed and an overdose may occur. In 2004 the Joint Commission recognized the importance of this problem by making PCA-by-proxy their 33rd sentinel-event. Sentinel events are occurrences that must be reported and investigated to their root cause or the facility risks losing their accreditation [22]. Healthcare facilities that have completed staff education programs and incoroprated warning about PCA-by-proxy into their patient education have seen lower overall rates of oversedation [23].

Oversedation from PCA may result from multiple causes. [73] enumerates 17 potential errors that may occur in PCA administration and relates a case in which six of them occured during one patient's PCA use.

An analysis of reports to the MAUDE database maintained by the Food and Drug Administration (FDA)'s Center for Devices and Radiological Health (CDRH) from 1984 to 1989 found that 67% of problems associated with PCA pumps were caused by operator error [18]. This early study took place before the 1990 change in Federal Reporting Guidelines that requires reporting of incidents involving "device malfunctions and serious injuries or deaths" to FDA. A later study [31] found that nearly 80% of the 2009 reported incidents in 2002 and 2003 were blamed on device malfunctions and that nearly 65% of these suspected device malfunctions were confirmed by the device manufacturers. The human factors of pump interface design are an important means of reducing use errors [53] [54]

Respiratory depression associated with PCA varies between 0.3% and 6% depending on the patient population and how respiratory depression is defined [66]. Most cases of respiratory depression do not lead to permanent harm to the patient, but these still represent serious incidents with the potential to harm or kill patients.

The Institute for Safe Medicine maintains a voluntary database of medication errors. This MedMarx database contains 9500 PCA related errors in the span 2000 - 2004 [32]. These account for only 1% of the medication errors submitted to the database, but this 1% accounts for 6.5% of harmful outcomes. This almost certainly under-reports the actual number of occurrences, since the voluntary database can only track the rate of reporting, not the rates of errors or adverse events [50].

Adequate pain control provides benefits including improved patient satisfaction, lower rates of complication, reduced length of hospital stays, and lower rates of litigation [32]. Some biomedical engineers take the attitude that "the only safe medical device is one that's never taken out of the box", but discontinuing use of PCA pumps is simply not an option. While providing inadequate levels of medication would indeed reduce the chance of overdose, pain management is an essential part of the care of these patients.

Monitoring of Patients on PCA. Patients receiving PCA therapy are usually also connected to a patient monitor that records their vital signs. These monitors typically measure at least heart rate, blood pressure, respiratory rate, and oxygen saturation (SpO_2). The monitor has simple alarms which sound when the vital signs go outside of some preset limits. If the patient receives an overdose, their vital signs will eventually go outside of the limits and the alarms will sound, summoning a caregiver to the bedside. However, by the time their vital signs drop far enough to cause the alarm to sound, damage may have already been done. Caregivers are desensitized by frequent false positive alarms, and they may not respond as quickly as would be optimal. Furthermore, the infusion pump continues running until it is manually stopped by a caregiver, which may not happen immediately on their arrival at the bedside.

An automatic system that could detect oversedation and the onset of respiratory depression could add an additional safeguard to the system and would help to protect the many patients who are not adequately protected by existing systems and procedures. Such a system would require minimal changes to nursing workflows and could reduce the number of false positive alarms that require an immediate nursing response.

PCA systems have undergone extensive scrutiny from HDOs, regulators, and manufacturers. A strategy for structuring safety arguments for PCA was laid out in [26]. Model-based UI developement for PCA IS covered in [58] and [57] gives a methodology for verifying interactive software and caregiver workflows.

6.2.1 PCA System Implementation

We created several implementations of an application that monitors patient data for the early signs of respiratory failure and can stop the PCA infusion and sound an alarm if the patient experiences an adverse event. These implementations all use a pulse oximeter device that measures physiological signals from a clip on the patient's finger and processes them to calculate heart rate and SpO_2 outputs, where SpO_2 is the measure of blood oxygenation. Some implementations incorporate other vital signs from a patient monitor in addition to heart rate and SpO_2 from the pulse oximeter. Because there are no commercially available PCA pumps capable of being remotely controlled, we use modified commercial pumps and prototype pumps based on our Generic Infusion Pump project [8]. Figure 6.9 is a photograph of a demo system implementing the PCA Safety Interlock with the Generic Infusion Pump. Figure 6.11 shows an OpenICE app implementing an infusion safety interlock, configured to monitor heart rate, oxygen saturation, and respiratory rate and to send a stop command to the pump if the vital signs deteriorate past configurable limits.

We have published implementations and analysis of PCA systems with several set of collaborators [9] [65] [46] [4] [8]. This Section presents and summarizes that work and extends it by mapping early work onto the ICE framework and applying the analysis techniques presented in Chapter 5.

6.2.2 System Architecture

Figure 6.10 shows the components of the PCA safety system. Figure 6.12 shows the devices and essential data flow in this control loop. The variables in the system are listed in Table 6.2. The pulse oximeter receives physiological signals from the patient and processes them to produce heart rate and SpO_2 outputs. The Supervisor gets these outputs and makes a control decision, possibly sending a stop signal to the PCA Pump. The PCA pump delivers a drug to the patient at its programmed rate unless it is stopped by the Supervisor. The patient model gets the drug rate as an input and calculates the level of drug in the patient's body. This in turn influences the physiological output signals through a drug absorption function.

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

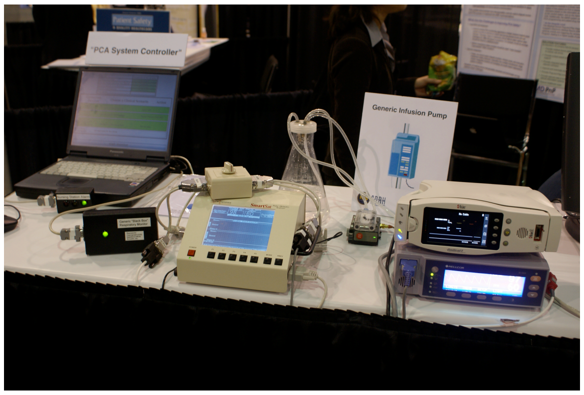


Figure 6.9: PCA Demo System

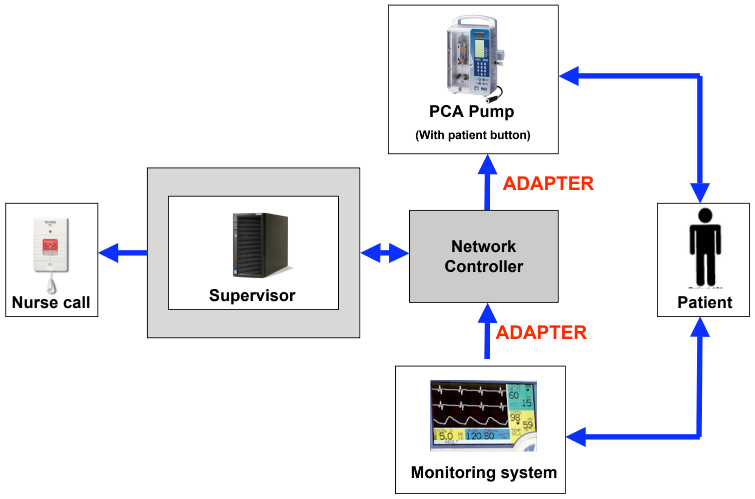


Figure 6.10: Hardware for PCA Demo System

source	description	name
pulse oximeter	signal processing time output HR and SpO$_2$ values	t_{po} hr, SpO$_2$
patient model	drug level drug absorption function output physiological signals	dl $f(dl)$ wf_1, wf_2
supervisor	algorithm processing time	t_{sup}
pca pump	pump stop delay infusion rate	t_{stop} rate

Table 6.2: Variables for critical timing loop

is a real problem, and finally stops the pump manually.

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

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

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

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

The pulse oximeter measures the patient's SpO_2 at regular intervals, processes them, and outputs an averaged result [20]. It calculates the average using a variable-sized sliding

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

Table 6.3: Sliding Window Size for Pulse Oximeter

window. The window size varies with the last output value. The reason for changing the window size is that smaller sample size gives faster, but potentially less accurate results. When SpO_2 values are low, quick response is more important than filtering out transient noise. When SpO_2 is high, increasing the window size helps to filter out transient low values at the expense of less frequent updates. Since the samples are at regular intervals and a varying number of samples are used to calculate the output, the output is updated irregularly. The size of the sliding window that we used in the case study is determined using a simple table shown in Table 6.2.2. Note that this table does not reflect the details of any real implementation but rather attempts to capture the essential behavior of a typical pulse oximeter.

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

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

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

Clinical Application Script for PCA Safety. The clinical application in this case study is to control the loop shown in Figure 6.12. The app receives the patient's heart rate and SpO_2 measurements from the pulse oximeter and uses this information to decide whether the PCA infusion pump should be allowed to run or immediately stopped.

The goal of this CAS is to detect when the patient's SpO_2 drops below a lower limit for longer than a threshold time and to stop the pump before the pump delivers more than a limited amount of drug. If the SpO_2 drops below t_s for longer then t_t, then the pump must be stopped before it delivers more than t_d quantity of drug d.

In order to accomplish this goal, it is necessary to know accurately when the SpO_2 drops. This requires a model of the pulse oximeter since pulse oximeters have a processing time which varies according to the current input values.

It is also necessary to have a model of the pump. The controller needs to know how long it will take to stop the pump and how much drug the pump will deliver before it is stopped. This is not always possible in practice since the control algorithm does not have all the necessary information.

In the case study, we designed a simple control algorithm for the supervisor, in which the decision to stop the pump is made as soon as the patient heart rate or SpO_2 readings fall below a fixed threshold. The choice of threshold needs to ensure that the patient does not enter the *Critical* region despite the delay in detecting the problem and delivering the control signal to the pump. For the case study, we defined the threshold as $H_1 = 90\%$ for the SpO_2 and $H_1 = 57$ *beats/min* for heart rate. Values below these thresholds typically indicate "a clinical concern" ([39], p. 45), meaning that a caregiver needs to be notified. The supervisor notifies the caregiver when the threshold is crossed, as it sends the message to stop the pump. Values between H_1 and H_2 are thus referred as the *Alarming* region. The width of the alarming region is denoted $\Delta H =\mid H_2 - H_1 \mid$. The OpenICE implementation shown in Figure 6.11 extends this with similar thresholds for heart rate and respiratory rate.

6.2.3 Verification and Validation of Components and System in UPPAAL

The structure of the UPPAAL model follows the architecture of the system. For each component in Figure 6.10, the model includes a separate automaton. The automata communicate using synchronization channels and shared variables. Figure 6.13 shows network of automata and communication between them. Solid arrows represent communication channels and dashed arrows represent shared variables.

The PCA automaton, which represents the pump, is shown in Figure 6.14. When the pump is operational, it is either in the state `running`, with the shared variable `pca_rate` set to default rate, or in the state `bolusing`, when `pca_rate` is increased by the bolus rate. Both rates are specified as parameters of the model. The pump can be bolusing for a fixed duration given by the value of the `bolus_time` parameter. The pump transitions to the `bolusing` state upon the signal received from the patient only if it is in the `running` state; in all other states, the signal is ignored. From either `running` or `bolusing` state, the pump can move to a stopped state (`Rstopped` or `Bstopped`, respectively) upon a signal from the network.

UPPAAL Component Models. The PO automaton, which represents the pulse oximeter, is shown in Figure 6.15. The operation of the automaton proceeds in rounds. Each round begins by setting the window size for the round based on the last sampled value. Then, the automaton collects the number of samples to fill the window. Samples are obtained periodically with the interval of 1 time unit, which corresponds to 100 ms. Finally, the result is stored in the `po_result` variable and delivered to the supervisor using the `resultready` channel.

The application automaton, shown in Figure 6.16, implements the simple control algorithm. Upon receiving a SpO_2 reading from the pulse oximeter, the app compares it with the pre-defined threshold value and, if the result is too low, sends the stop message to the pump across the network. The model also incorporates a delay, which represents the worst-case execution time of the app algorithm. Then, once the caregiver resolves the problem, the app sends another message to restart the pump. For simplicity of the presentation, the app automaton only deals with SpO_2 , not heart rate or respiratory rate.

The Patient automaton, shown in Figure 6.17, periodically updates the drug level based on the flow rate of the pump and drug absorption rate. At any time, it can deliver a sample as the function of the current drug level.

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

We express properties we verify in the subset of the Computational Tree Logic (CTL) used by UPPAAL.

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

$$\begin{aligned} &A\Box(samplebuffer < pain\_thresh \Rightarrow \\ &\quad A\Diamond\, samplebuffer \geq pain\_thresh). \end{aligned}$$

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

$$A\Box(\mathit{samplebuffer} < \mathit{alarm\_thresh} \Rightarrow$$
$$A \diamond (\mathit{PCA.Rstopped} \vee \mathit{PCA.Bstopped})).$$

We consider this property to be a sanity check rather than a safety requirement, because wrong parameters of the model – for example, too short bolus duration or too high drug absorption rate – can make the system appear safe (that is, SpO_2 level never goes too low), but it would be safe for the wrong reason. All sanity checks were passed by the UPPAAL model described above when no dropped messages are allowed. Clearly, property (6.2.1) does not hold if messages can be dropped.

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

6.2.4 PCA Safety Interlock Summary

PCA infusions are responsible for numerous injuries and deaths. Many of these adverse events would be preventable if PCA was delivered within a system that monitored the patient continuously and could stop the infusion when a problem was detected. Such systems need to incorporate multiple vital signs monitors to reduce the number of false alarms and to avoid stopping the infusion unnecessarily when, for instance, the pulse oximeter finger clip comes off the patient's finger. Building these systems requires interoperability and the ability to reason about the system's safety.

We have shown how our work on PCA systems [9] [65] [46] [4] [8] fits into the safety analysis framework described in Chapter 5.

6.3 Discussion

We present two case studies using the modeling language described in Chapter 3 together with the system architecture from Chapter 4 and the property checking techniques from Chapter 5. Many medical applications fit into the broad categories of smart alarms, safety interlocks, and closed-loop control. Smart alarms receive information from patient care devices and create alerts for clinicians. Clinical decision support algorithms fit into this category, often creating less time-sensitive alerts. Safety interlocks are a type of smart alarm that also includes a component of device control. A safety interlock will lock a device into a particular mode when a set of conditions are met. The PCA case study described in Section 6.2 is an example of a safety interlock where the pump is stopped when the algorithm detects the onset of respiratory depression. The X-Ray / Ventilator synchronization case study described in Section 6.1 can also be categorized as a safety interlock that restricts the x-ray machine to exposing only when the patient's lungs are in the correct state. Closed-loop control applications use a physiologic measurement as an input to an algorithm controlling one of the devices. In this sense, the x-ray / ventilator application is a closed-loop system that uses a measurement of lung inflation to control the timing of the x-ray exposure, but closed-loop control algorithms usually repeat the measurement and actuation cycle rather than running through the loop once.

Limitations. This Chapter covers two use cases built using the system described in Chapters 3, 4, and 5 and following the ASTM ICE architecture described in Section 2.3. The OpenICE platform (Section 2.4) was used for parts of the case studies, particularly the PCA safety interlock. The limitations, gaps, and future work described in this Section apply to the process of building implementations of clinical applications using this system.

Some of the limitations of these case studies follow from limitations of the available devices. We weren't always able to implement the algorithms we wanted because the available devices did not support the necessary functionality. This is the reality of building systems on top of legacy devices. These implementations were done in non-clinical spaces and were not used on patients. This allowed for rapid prototyping and change but means that they were tested against simulated patients that do not exhibit the same variability as populations of real patients. The validity of the model checking results is limited to patients who are accurately represented by the patient model. If the patients have unexpected or unknown co-morbidities that change the way they react to treatment relative to the patient model, the system safety properties may not hold.

Limitations discussed in previous chapters affected the case studies. We did not model the network, and real networks have latency, loss, jitter, and other attributes that are not included in CAML communications channels but do affect implementations. In our implementations, we over-provisioned the network so that the bandwidth used was a small fraction of what was available and network latency and jitter were kept low, but this does not guarantee performance.

Timing requirements for these case studies are realistic for medical use cases, where most changes occur over seconds or minutes. The exceptions are electrical activity, primarily brain and cardiac functions, where waveform data can be collected and analyzed at higher rates, typically 200 - 512 Hz for electrocardiograms. For the PCA use case, timing was on the order of seconds, while the X-Ray / Ventilator synchronization case had some faster timing requirements, triggering a 10ms exposure within a 50 - 200ms window. The generated Java code was run in a non-real-time Java virtual machine on a non-real-time operating system with enough extra capacity to minimize interruptions from garbage collection or other operating system processes and functioned well through a wide range of respiratory rates. This is a limitation of these use case implementations that means that we can not guarantee that the safety properties hold for the implementations. To guarantee the properties, we would need a platform that guarantees preservation of the model semantics; one possibility would be a real-time Java virtual machine running on a real-time operating system, with careful proofs that the model semantics are preserved through code generation and in the execution environment.

The PCA and X-Ray / Ventilator Synchronization applications included in the OpenICE 1.0 distribution do not include the generated code; instead these applications were written to demonstrate the applications and potential of interoperability in a portable way.

Gap Analysis. Many safety properties about closed-loop systems involve the mathematics used in the control algorithm. For instance, showing that the algorithm will not change the rate of an infusion too quickly, that the algorithm will converge on a steady state rather than causing oscillations between safe and unsafe conditions, and that the algorithm will be able to effectively control the condition of patients as modeled in the patient model are all typical control theory problems that would have to be modeled at a very high level of abstraction in CAML. In some cases, the resulting models may be too abstract to be useful in validating the system design. Modeling physiologic closed-loop control systems where the mathematical details of the control algorithm are critical to assuring the safety properties, is better done using hybrid modeling techniques. In general, the approach presented here seems most useful for smart alarm and safety interlock systems rather than closed-loop control.

Future Work. These use cases were chosen to exercise several design patterns that are common in clinical decision support and treatment algorithms. These include safety interlocks, smart alarm systems, and physiologic closed-loop control. However, two use cases - however carefully chosen - cannot cover the whole space of medical treatment. The architecture described in this thesis and used for these case studies works well for these use cases and for others like them, but there are likely to be other use cases that would inspire changes to the architecture, modeling language, and toolset.

More work is needed to further develop the modeling language, particularly to support multiple simultaneous applications. We believe the best way to identify gaps in the capabilities of CAML and the model checking system described here is to continue to use the system to build and test and ultimately deploy clinical applications in conjunction with clinicians and other domain experts.

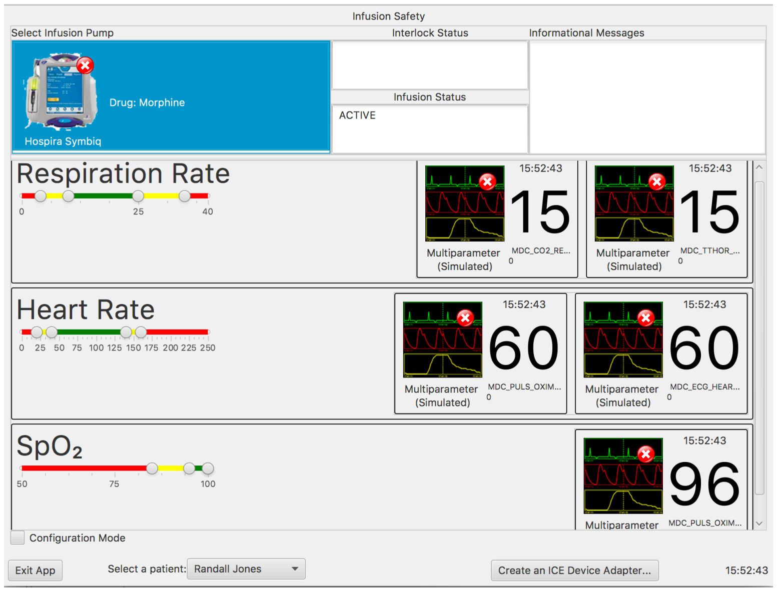


Figure 6.11: OpenICE Infusion Safety App

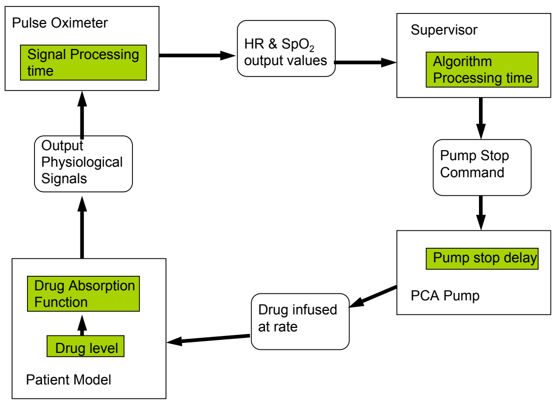


Figure 6.12: PCA System Control Loop

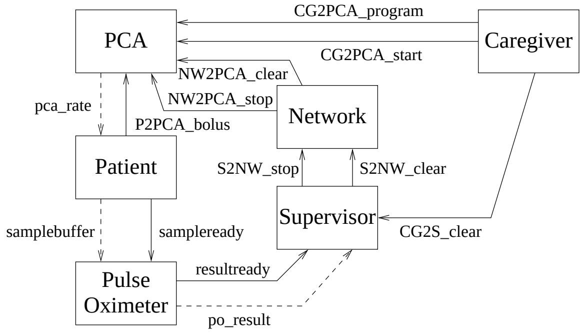


Figure 6.13: Communication structure of the UPPAAL model

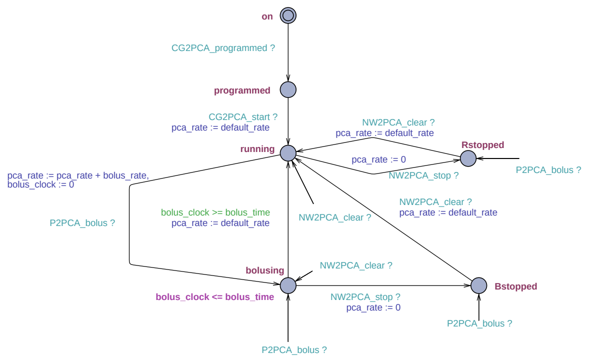
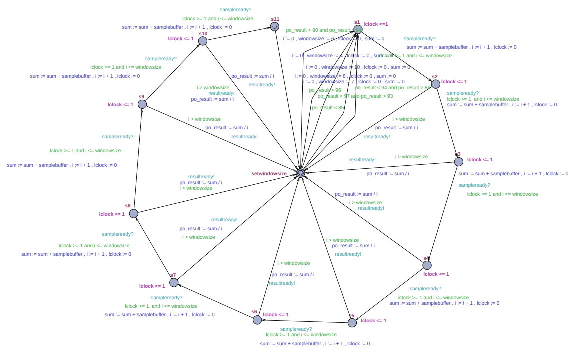


Figure 6.14: Timed automaton for the PCA pump



setwindowsize
s1
lclock <=1
s2
s3
s4
s5
s6
s7
s8
s9
s10
s11
lclock <= 1
sampleready?
lclock >= 1 and i <= windowsize
sum := sum + samplebuffer , i := i + 1 , lclock := 0
i > windowsize
po_result := sum / i
resultready!
po_result < 90 and po_result > 84
i := 0 , windowsize := 6 , lclock := 0 , sum := 0
i := 0 , windowsize := 4 , lclock := 0 , sum := 0
i := 0 , windowsize := 10 , lclock := 0 , sum := 0
i := 0 , windowsize := 8 , lclock := 0 , sum := 0
i := 0 , windowsize := 7 , lclock := 0 , sum := 0
po_result > 96
po_result < 94 and po_result > 89
po_result < 97 and po_result > 93
po_result < 85

Figure 6.15: Timed automaton for the pulse oximeter

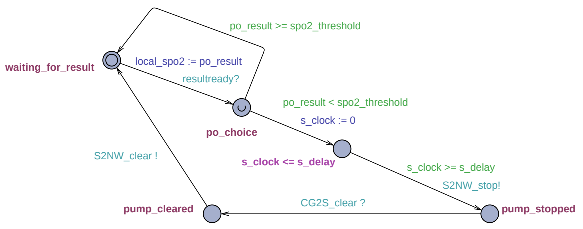


Figure 6.16: Timed automaton for the supervisor

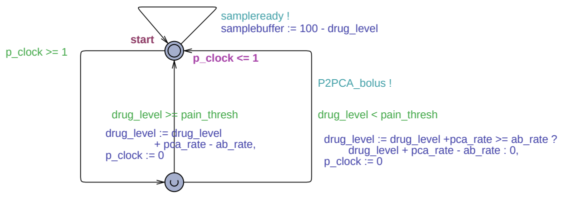


Figure 6.17: Timed automaton for the patient

Chapter 7

Conclusion

Medical device interoperability has the potential to vastly improve patient safety, add to medical knowledge, and even reduce healthcare costs while also improving patient outcomes.

Some of the significant barriers to device interoperability have been a lack of interface standardization, lack of common nomenclature, and a lack of a regulatory pathway for components of connected systems. These gaps have lead to an inability to reason about the safety of interoperable systems because of the fragmentary and heterogeneous information available about different components of the system.

In this work, we have tried to address these issues by presenting an architecture for modeling the interfaces and components necessary to allow building an interoperable system of medical devices that supports proving safety properties. We have addressed significant problems around modeling clinical environments, workflows, patients, and devices including software applications. Gathering and documenting a comprehensive set of hazards associated with devices and applications is a challenge all device manufacturers face. We discuss this process and how to go from a list of hazards to formally testable device requirements and system safety properties. We explain how to check an application's device requirements against these models and finally how system safety properties can be tested against a system composed of a clinical application, a workflow model representing caregiver actions, a patient model and a set of devices.

In the future, we hope that devices will be able to communicate normalized, time-synchronized data over standards-based communications networks. This will allow recording and analysis of data from devices, even when the devices come from different manufacturers. Extending these abilities with metadata about the measurements allows checking some simple device requirements and system safety properties, and enabling devices and applications to communicate models of their behavior opens up additional possibilities. Most important safety properties about treatments are closely tied to the intended use of the treatment application, which necessarily includes assumptions about the patient population being treated and the clinical environment within which the treatment is happening. Our goal is to be able to reason about safety properties for treatment applications and, ultimately, to make the practice of medicine safer and to improve patient outcomes.

Appendix A

Hazard Analyses

This section defines terms used in assessing the probability and severity of hazards and lists sources used in the Hazard Table. The definitions of probability and severity, the format of the Hazard Table, and about 25 of the hazards listed in the table, are taken directly from [59].

The accompanying Hazard Tables attempt to identify potential hazards and assess their severity and probability to define a resultant risk. Identified hazards are then addressed with a mitigation action that should reduce this risk. Device verification testing is still required to ensure that firstly, the required mitigation has been met and secondly that the proposed mitigation has the desired outcome. For full details see ISO 14971 [35] with which this analysis is designed to comply.

Mitigation - This field will contain a brief description of the control mechanism(s) required to reduce the risk of the hazard event, if required.

Verification - This field is used to specify the verification activity required to verify the mitigation implementation.

Severity - The initial and final field on the chart indicates the seriousness of the hazard event before and after mitigation, respectively.

The severity is defined as:

NEGLIGIBLE will not result in injury or illness to the patient or system operator. No damage to the user environment (e.g. physical, contamination, EMC).

MINOR could result in minor injury to the patient or user. Little or no damage to the environment.

MODERATE could result in moderate injury or illness to the patient or user. May cause moderate damage to the environment.

MAJOR could result in death or serious injury or illness to the patient or user without intervention. May cause significant damage to the user environment.

CATASTROPHIC could result in death to more than one patient or user. May cause severe damage to the user environment.

Probability - The initial and final field on the chart also indicates the probability of the hazard event occurring.

The probability is defined as:

IMPROBABLE So unlikely to occur, it can be assumed that this hazard will not occur.

REMOTE Unlikely to occur but possible.

OCCASIONAL Likely to occur sometime in the life of the product.

PROBABLE likely to occur more than once in the life of the product.

FREQUENT likely to occur several times in the life of the product.

Risk - The initial and final field on the chart also indicates the risk associated with a hazard event.

Given the severity of the outcome and the probability of failure, the table below is used to identify the risk level of each identified hazard. Adjustments up or down may be warranted in the case of hazards with unclear failure modes or an unusually severe hazard.

Risk	Severity				
Probability of Failure	I Negligible	II Minor	III Moderate	IV Major	V Catastrophic
A. Improbable	Minimum	Minimum	Minimum	Minimum	Low
B. Remote	Minimum	Low	Low	Low	Medium
C. Occasional	Minimum	Low	Medium	Medium	High
D. Probable	Minimum	Low	Medium	High	High
E. Frequent	Low	Medium	High	High	High

Hazards are listed in one of the following categories:

- H1 Energy Hazards
- H2 Biological Hazards
- H3 Environmental Hazards
- H4 Hazards Relating to Use
- H5 Functional Failure, Maintenance, and Aging Hazards
- H6 PCA Hazards: Additional hazards introduced by the addition of a PCA module.
- H7 Network Hazards: Additional Hazards introduced by the addition of network connectivity.

- H8 Drug Library Hazards

Future work could include adding additional hazards for:

- H9 Insulin Pump Hazards
- H10 Home Use Environment Hazards

Sources: 1. PRS Level 1 Hazard Analysis, accessed 09 November, 2010 2. The Generic Patient Controlled Analgesia Pump Hazard Analysis from the Generic PCA (GPCA) Model ver 0.9 3. Hazard review by GIP team, 11/2010 4. First, Do No Harm: Making Infusion Pumps Safer BI&T Set/Oct 2010 Vol 44 No 5. 5. Notes from AAMI / FDA Infusion Device Summit 10/5 - 6 Silver Spring, MD 6. AAMI / FDA Infusion Device Summit Pre-Summit Survey Summary 10/1/2010

A.1 Generic Infusion Pump Hazard Analysis

Number	Category	Subcategory	Description	Initial	Predicted	Mitigation	Verification	Source
H1	Energy Hazards							
C1.1	Energy Hazards	Electricity						
C1.1.1	Energy Hazards	Electricity	Possibility of supply voltage leakage when Patient and mains power simultaneously connected to system.	Occasional Major Medium Risk	Improbable Major Min Risk	Ensure GIP fitted with approved power supply meeting [standard] and that all physically connected devices meet ISO 60601-1	Verify design requirement fulfilled. Ensure GIP User Documentation emphasises need to confirm approval of connected devices.	1
C1.1.2	Energy Hazards	Electricity	Possibility of voltage leakage between physically connected devices.	Occasional Moderate Medium Risk	Improbable Moderate Medium Risk	Ensure all electrical connection ports in GIP provide electrical isolation between each other to [standard]	Verify design requirement fulfilled and test as part of electrical safety verification testing.	1
C1.1.3	Energy Hazards	Electricity	Supply voltage leakage could occur between applied parts.	Remote Moderate Low Risk	Remote Moderate Low Risk	Design to EN60601-1		1
C1.1.4	Energy Hazards	Electricity	Possibility of shorting due to gas or fluid ingress.	Occasional Moderate Medium Risk	Improbable Moderate Medium Risk	Design GIP to appropriate IP rating for the environment in which it is to be used	Verify design requirement fulfilled and test as part of environmental testing.	1
C1.1.5	Energy Hazards	Electricity	Inductive coupling of pump noise to patient vis conductive infusate	Remote Moderate Low Risk	Improbable Moderate Min Risk	Shield pump motor, power supply, other electronics.	Verify design requirement fulfilled and test as part of electrical safety verification testing. Test for interferance with other devices such as patient monitors and EEG.	2

C1.1.6	Energy Hazards	Electricity	Shock hazard from frayed system power cord	Occasional Moderate Medium Risk	Remote Moderate Low Risk	Cord must be durable. Build to applicable standards.	Design verification, testing	5
C1.2	Energy Hazards	Heat						
C1.2.1	Energy Hazards	Heat	System circuitry gets hot.	Remote Minor Low Risk	Improbable Minor Low Risk	Design system to ensure adequate cooling. Draw attention to the need to work within the operating environment in the User documentation	Design verification. User documentation verification.	1
C1.3	Energy Hazards	Mechanical Force						
C1.3.1	Energy Hazards	Mechanical Force	Accidental movement causes unit to strike patient or user	Occasional Minor Low Risk	Occasional Minor Low Risk	Reduce risk by appropriate design for intended environment. Draw attention to patient safety in user documentation.	”Design output. User documentation verification”	1
C1.4	Energy Hazards	Acoustic Pressure						
C1.4.1	Energy Hazards	Acoustic Pressure	Noise from the system causes discomfort or injury	Remote Negligable Min Risk	Improbable Negligable Min Risk	User Documentation to highlight need for audio alarms to be set at an appropriate level	User documentation verification / review	1
C1.5	Energy Hazards	Vibration	No Hazard Identified					
H2	Biological Hazards							
C2.1	Biological Hazards	Contamination						

C2.1.1	Biological Hazards	Contamination	Spillage or exposure to toxins	Probable Moderate Medium Risk	Probable Minor Low Risk	Design pump and infusion set to resist incursion of outside material. Seal pump case adequately for the use environment.	Design verification.	2
C2.1.2	Biological Hazards	Contamination	Battery leak	Remote Moderate Low Risk	Remote Improbable Min Risk	Use appropriate battery and case design.	Design verification.	2
H3	Environmental Hazards							
C3.1	Environmental Hazards	Power Supply						
C3.1.1	Environmental Hazards	Inadequate power supply	The System acts in an unintentional manner when power becomes low	Occasional Major Medium Risk	Improbable Major Min Risk	Design GIP to monitor power to ensure early notification and if necessary safe shut down	Test unit to specified standards	1
C3.2	Environmental Hazards	Operation outside of Prescribed Environmental Conditions						
C3.2.1	Environmental Hazards	Operation outside of Prescribed Environmental Conditions	Excessive humidity	Probable Minor Low Risk	Probable Minor Min Risk	Design system to ensure adequate ventilation. Draw attention to the need to work within the operating environment in the User documentation. Humidity sensor and alarm may be necessary.	Design verification. User documentation verification.	2

C3.2.2	Environmental Hazards	Operation outside of Prescribed Environmental Conditions	Temperature extremes	Probable Minor Low Risk	Probable Minor Min Risk	Design system to ensure adequate cooling. Draw attention to the need to work within the operating environment in the User documentation. Temperature sensor and alarm may be necessary.	Design verification. User documentation verification.	2
C3.2.3	Environmental Hazards	Operation outside of Prescribed Environmental Conditions	Noisy environment masks alarm sounds	Probable Moderate Medium Risk	Probable Minor Low Risk	Use auditory and visual alarms. Allow users to modify alarm volume. Maximum alarm volume should be loud enough to be heard in typical environments. Conform to relevant alarm standards.	Design verification.	2
C3.3	Environmental Hazards	Incompatible with other devices						
C3.3.1	Environmental Hazards	Incompatible with other devices	GIP interferes with nearby electrical systems.	Occasional Major Medium Risk	Improbable Major Min Risk	Design unit to specific standards relating to possible environmental scenarios of intended use.	Test unit to specified standards.	1
C3.4	Environmental Hazards	Electromagnetic Incompatibility						
C3.4.1	Environmental Hazards	Electromagnetic Incompatibility	GIP not protected against external electromagnetic radiation	Probable Minor Low Risk	Improbable Minor Min Risk	Design product to meet IEC 60601-1-2 EMC requirements.	Design Output	1
H4	Hazards Relating to Use							

C4.1	Hazards Relating to Use	Incorrect Application						
C4.1.1	Hazards Relating to Use	Incorrect Application	The GIP is used incorrectly	Frequent Moderate High Risk	Improbable Moderate Medium Risk	Instructions for use to include Quick Start guide immediately visible on attempting to use system. Instructions for use to include interactive training materials and remote support via web-site, including links to sites for connected devices.	Verify user instruction materials are as required.	1
C4.2	Hazards Relating to Use	Inadequate Labeling						
C4.2.1	Hazards Relating to Use	Inadequate Labeling	The system labelling is too complex.	Occasional Minor Low Risk	Improbable Minor Min Risk	Include Quick Start guide and make use of visual labels. Conduct user acceptance trials to assess quality of instructions and modify accordingly	Verify user instructions and manuals are as required.	1
C4.2.2	Hazards Relating to Use	Inadequate Labeling	The system labelling is incomplete.	Occasional Minor Low Risk	Improbable Minor Min Risk	Conduct user acceptance trials to assess quality of instructions and modify accordingly.	Verify during protocol verification.	1
C4.2.3	Hazards Relating to Use	Inadequate Labeling	The system labeling is confusing	Occasional Minor Low Risk	Improbable Minor Min Risk	Design instructions to current human factors practices to be clarified as part of design phase.	Design output.	1
C4.3	Hazards Relating to Use	Customer Purchases Wrong Accessories						

C4.3.1	Hazards Relating to Use	Customer Purchases Wrong Accessories	Inadequate specifications for patient accessories.	Occasional Minor Low Risk	Improbable Minor Min Risk	Clearly state in the instructions for use that only approved accessories can be used with the system. Maintain accurate list of approved accessories on website	Verify user instruction materials are as required. Verify procedures to automatically update website with new accessory information upon successful test of new devices	1
C4.3.2	Hazards Relating to Use	Customer Purchases Wrong Accessories	Use of unapproved accessories, e.g., third-party infusion sets	Remote Major Low Risk	Improbable Major Min Risk	Clearly state in the instructions for use that only approved accessories can be used with the system. Maintain accurate list of approved accessories on website	Verify user instruction materials are as required. Verify procedures to automatically update website with new accessory information upon successful test of new devices	5
C4.4	Hazards Relating to Use	Use by Unskilled or Untrained Personnel						
C4.4.1	Hazards Relating to Use	Use by Unskilled or Untrained Personnel	Users have not had appropriate training.	Occasional Minor Low Risk	Improbable Minor Min Risk	System to only allow authenticated users to use the system. System administrator only to add users once appropriate training given. Ongoing training records to be kept.	"System verification testing. User documentation verification. "	1

C4.4.2	Hazards Relating to Use	Use by Unskilled or Untrained Personnel	Children and other animals	Probable Major High Risk	Probable Minor Low Risk	Lock out keypad, physically secure power and other connectors, appropriate strain relief for all electrical and fluid connections.	Design Verification. System Testing.	2
C4.5	Hazards Relating to Use	Human Error						
C4.5.1	Hazards Relating to Use	Human Error	The GIP user is color blind	Remote Minor Low Risk	Improbable Minor Min Risk	Ensure that the design does not rely solely on color-coding for correct operation	Design Output	1
C4.5.2	Hazards Relating to Use	Human Error	GIP user types wrong number while programming pump	Frequent Major High Risk	Frequent Minor Medium Risk	Validate user interface design with use testing. For critical infusions, force two different users to program pump and check that they match. Automate programming (e.g., with barcodes) to elimate manual data entry. Use Drug Library to 'sanity check' input	Design Verification	4
C4.5.3	Hazards Relating to Use	Human Error	GIP user enters value using wrong units. E.g., milliliters instead of microliters.	Frequent Major High Risk	Frequent Minor Medium Risk	Validate user interface design with use testing. For critical infusions, force two different users to program pump and check that they match. Automate programming (e.g., with barcodes) to elimate manual data entry. Use Drug Library to 'sanity check' input	Design Verification	4

C4.5.4	Hazards Relating to Use	Human Error	GIP user selects wrong drug from drug list.	Frequent Major High Risk	Frequent Minor Medium Risk	Validate user interface design with use testing. For critical infusions, force two different users to program pump and check that they match. Automate programming (e.g., with barcodes) to elimate manual data entry. Use Drug Library to 'sanity check' input	Design Verification	4
C4.5.5	Hazards Relating to Use	Human Error	Wrong drug installed on a channel of a multi-channel pump	Frequent Major High Risk	Frequent Minor Medium Risk	Display drug name on pump channel. Educate users.	Design Verification. System Testing. Verify that user manual addresses the issue.	5
C4.5.6	Hazards Relating to Use	Human Error	Infusion connected to the wrong port on a patient	Probable Major High Risk	Remote Major Low Risk	Use distinct connectors for different types of ports. Color code IV sets. Educate caregivers about the danger.	Design Verification. Verify that user manual addresses the issue.	5
C4.5.7	Hazards Relating to Use	Human Error	Use on inappropriate patient	Probable Major High Risk	Remote Major Low Risk	Check patient age, weight, etc. against orders and drug library. Check against barcode input of patient ID bracelet with pertinant information or prompt the caregiver.	Design Verification. System Testing.	2
C4.5.8	Hazards Relating to Use	Human Error	Alarms ignored	Frequent Major High Risk	Occasional Major Medium Risk	Pumps should minimize alarm fatigue by reducing false or nuisance alarms. Varying levels of alarms can inform caregivers when a condition is life threatening vs. a warning.	Design Verification	5

C4.5.9	Hazards Relating to Use	Human Error	Infusion Order is incorrect	Probable Major High Risk	Probable Minor Low Risk	Check against Drug Library	Design Verification	3
C4.5.10	Hazards Relating to Use	Human Error	Pump workflow doesn't match User workflow	?	?	Design must match how pump is used in intended environment.	Design Verification, Use Testing	5
C4.5.11	Hazards Relating to Use	Human Error	Not lowering primary bag below the secondary bag	Frequent Major High Risk	Occasional Major Medium Risk	User training and documentation	Use testing, Verification of documentation	6
C4.5.12	Hazards Relating to Use	Human Error	Not unclamping the roller on the infusion set	Frequent Major High Risk	Occasional Minor Low Risk	Occlusion sensor and alarm, caregiver training	Design Verification. Functional Testing. Verify that user manual addresses the issue.	6
C4.6	Hazards Relating to Use	Incorrect Measurements						
C4.6.1	Hazards Relating to Use	Incorrect Measurements	user interface does not truly represent an item(s) of data correctly.	Probable Moderate Medium Risk	Remote Moderate Low Risk	Consult with users to match data representations to their mental models	Design verification. Use Testing. System testing.	1
C4.6.2	Hazards Relating to Use	Incorrect Measurements	Pump doesn't account for residual volume in the tubing	Probable Moderate Medium Risk	Remote Moderate Low Risk	Infusion set information entered or set automatically then used to adjust programming	Design verification. System testing.	5
C4.6.3	Hazards Relating to Use	Incorrect Measurements	Volume of drug added to bag not accounted for	Probable Moderate Medium Risk	Remote Moderate Low Risk	Automatic labeling of bags in pharmacy, bar coding, UI that helps with calculations	Design verification. System testing.	5
C4.7	Hazards Relating to Use	Corrupted or Incomplete Data						

C4.7.1	Hazards Relating to Use	Corrupted or Incomplete Data	An item of data from a medical device is delayed, not received or corrupted between patient and user display.	Probable Moderate Medium Risk	Remote Moderate Low Risk	System topology to be designed with robust error checking, data and communications verification with extensive error and notice output with clear indication of error location.	"Design output verification. System testing verification. User acceptance verification. "	1
C4.8	Hazards Relating to Use	Accessory Misuse						
C4.8.1	Hazards Relating to Use	Accessory Misuse	An accessory is not connected to the system correctly.	Probable Minor Low Risk	Improbable Minor Min Risk	User Interface to differentiate between attached devices i.e those that are physically connected or can be seen by the system and connected devices where the device interface connection verification routine has passed.	"Design output verification. Device interface verification "	1
C4.8.2	Hazards Relating to Use	Accessory Misuse	An accessory is not used in accordance with its intended use or user documentation.	Probable Moderate Medium Risk	Remote Moderate Low Risk	Train Users, ensure that documentation is adequate.	"User documentation verification. System verification. "	1
C4.9	Hazards Relating to Use	Tampering						
C4.9.1	Hazards Relating to Use	Tampering	Unauthorized person changes pump settings	Remote Major Low Risk	Improbable Major Min Risk	Lock out pump UI, require passcode to change settings. Don't ship pumps with a common default passcode.	Design verification	2

H5	Functional Failure, Maintenance, and Aging Hazards							
C5.1	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics						
C5.1.1	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Flow rate does not match programmed rate	Remote Major Low Risk	Improbable Major Min Risk	Design pump hardware to be robust. Minimize opportunities for freeflow of fluid. If feasible, use a flow sensor or other means of directly measuring fluid flow rate.	System verification. Functional testing.	2
C5.1.2	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Excessive variation in flow rate	Remote Major Low Risk	Improbable Major Min Risk	Limit minimum pump flow rate in accordance with pump hardware capabilities. E.g., pumping a 1mL bolus once an hour is not a good way to deliver a 1mL/hr constant flow rate.	Design verification. Functional testing.	2
C5.1.3	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Bleed back; reflux within device	Remote Major Low Risk	Improbable Major Min Risk	Design pump hardware and accessories (infusion set) to be robust and enforce one-way flow of fluid	System verification. Functional testing.	2

C5.1.4	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Failure of alarm	Remote Major Low Risk	Improbable Major Min Risk	Design alarm hardware to be robust. Use multiple alarm modalities (audio and visual). Test hardware during power-on self test, e.g., using a microphone to check if the speaker functions correctly.	Design verification. Functional testing.	2
C5.1.5	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Failure to prime	Occasional Major Medium Risk	Remote Major Low Risk	Design UI to include priming step before starting infusion. Air in line alarm to catch when priming not done.	Design verification. Functional testing.	2
C5.1.6	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Air in line	Occasional Major Medium Risk	Occasional Minor Low Risk	Design infusion set and pump mechanism to reduce opportunities for air to enter the system. Air alarm to detect air bubbles and stop pump.	Design verification. Functional testing.	2
C5.1.7	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Incorrect, loose, or corroded connections between GIP components	Remote Major Low Risk	Improbable Major Min Risk	Detect unexpected loss of connectivity and alarm.	Design verification. Functional testing.	2
C5.1.8	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Realtime Clock (RTC) failure	Remote Major Low Risk	Improbable Major Min Risk	Battery backup of clock, power-on self test.	Design verification. Functional testing.	2
C5.1.9	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Stuck or improperly debounced keys	Remote Major Low Risk	Improbable Major Min Risk	Power-on self test, drug library in case of misentry of programming parameters	Design verification. Functional testing.	2

C5.1.10	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Sensor Failure	Remote Major Low Risk	Improbable Major Min Risk	Power-on self test, Test sensor values for stuck-on, stuck-off, and repeating failures	Design verification. Functional testing.	2
C5.1.11	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Can't easily back up in programming, have to re-enter entire sequence	Probable Minor Low Risk	Improbable Minor Low Risk	UI should allow users to easily correct mistakes	Design Verification, Use testing	5
C5.1.12	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Can't read log data from pump	Probable Minor Low Risk	Improbable Minor Low Risk	Pumps should use standard log format, manufacturers must provide software to download and interpret pump logs.	Design verification. Functional testing, Use testing	5
C5.1.13	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Inadequate battery life	Probable Minor Low Risk	Improbable Minor Low Risk	Pumps must accurately calculate remaining battery life as batteries age.	Design verification. Functional testing.	5
C5.1.14	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Screen not readable	Probable Minor Low Risk	Improbable Minor Low Risk	Make screen large and bright. Screen angle adjustable and at the proper height for its intended use environment	Design Verification, Use Testing	6
C5.1.15	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Excessive force of infusion	Remote Major Low Risk	Improbable Major Min Risk	Pressure sensors to detect and alarm on excessive output pressure	Design verification, functional testing	6
C5.2	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.						

C5.2.1	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	The lack of training for service personnel impacts corrective service actions.	Probable Minor Low Risk	Improbable Minor Min Risk	Design system to be user serviceable to a certain extent. Give details of all service activities and their correct operation in user instructions.	"Design output. Verify user instructions are as required. "	1
C5.2.2	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Failure to flush	Occasional Major Medium Risk	Remote Major Low Risk	UI must enforce flushing infusion set where appropriate. User manuals should emphasize importance.	"Design output. Verify user instructions are as required. "	6
C5.2.3	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Failure to disinfect	Occasional Major Medium Risk	Remote Major Low Risk	UI must enforce disinfecting pump and infusion set where appropriate. User manuals should emphasize importance.	"Design output. Verify user instructions are as required. "	6
C5.2.4	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Reuse of infusion sets	Remote Major Low Risk	Improbable Major Min Risk	User training and User manuals must emphasize importance of using fresh, sterile infusion accessories	Verify user instructions are as required.	6
C5.3	Functional Failure, Maintenance, and Aging Hazards	Accessory Failure						
C5.3.1	Functional Failure, Maintenance, and Aging Hazards	Accessory Failure	Supply-side Occlusion	Frequent Moderate High Risk	Frequent Minor Medium Risk	Caregiver training to ensure proper infusion set setup. Pump sensors to detect supply side blockage and alarms to notify caregivers when detected.	Design verification. Functional testing. Verify user instructions are as required.	2

C5.3.2	Functional Failure, Maintenance, and Aging Hazards	Accessory Failure	Patient-side Occlusion	Frequent Moderate High Risk	Frequent Minor Medium Risk	Caregiver training to ensure proper infusion set placement and securing. Pump sensors to detect output pressure changes indicative of improper set placement and alarm if condition is detected.	Design verification. Functional testing. Verify user instructions are as required.	2
H6	PCA Hazards							
C6.1	PCA Hazards	Patient Button						
C6.1.1	PCA Hazards	Patient Button	PCA by Proxy	Frequent Major High Risk	Probable Major High Risk	Training of patient and family	Verify that caregiver training emphasizes teaching patients and family members or other visitors	3
C6.1.2	PCA Hazards	Patient Button	A rapid series of bolus requests leads to overinfusion	Occasional Major Medium Risk	Occasional Minor Low Risk	Drug Library for bolus rate, total volume, and lockout intervals.	Design Verification	2
C6.2	PCA Hazards	Tampering						
C6.2.1	PCA Hazards	Tampering	Unauthorized person removes or tampers with drug supply	Occasional Major Medium Risk	Occasional Minor Low Risk	Physically lock the drug supply. Require lock to be closed before starting infusion. Alarm if supply is removed without unlocking.	Design verification. Functional testing	2
H7	Network Hazards							
C7.1	Network Hazards	Conflict						

C7.1.1	Network Hazards	Conflict	Handling network communication swamps pump processor, infusion operations suffer (stop, can't be controlled, etc)	Remote Major Low Risk	Improbable Major Min Risk	Design pump controller and network interface as separate components; limit potential for interferrence.	Design Verification, Functional Testing	3
C7.1.2	Network Hazards	Conflict	Commands from network conflict with front panel commands	Remote Major Low Risk	Improbable Major Min Risk	User interface must clearly display when pump is in a network command mode and restrict the programming options available at the front panel accordingly.		3
C7.2	Network Hazards	Network Failures						
C7.2.1	Network Hazards	Network Failures	Unable to retrieve data from network	Frequent Moderate High Risk	Frequent Minor Low Risk	Devices must be able to fall back to a non-networked mode of operation, possibly raising alarms if they require manual entry of information to continue (e.g., a glucose pump that needs insulin values)		2
C7.2.2	Network Hazards	Network Failures	Unable to propogate alarms	Probable Major High Risk	Probable Minor Low Risk	Pumps must be able to annunciate alarms locally in case networked or central alarms are unavailable.		2
C7.3	Network Hazards	Tampering						
C7.3.1	Network Hazards	Tampering	Unauthorized or illegitimate firmware update pushed	Remote Major Low Risk	Improbable Major Min Risk	Firmware updates must be authenticated by the device before installation.		3

C7.3.2	Network Hazards	Tampering	Monitoring network traffic reveals protected patient information	Remote Minor Low Risk	Improbable Minor Min Risk	Network communications must be encrypted.		3
C7.3.3	Network Hazards	Tampering	Network traffic patterns reveal protected patient information	Remote Minor Low Risk	Improbable Minor Min Risk			3
C7.3.4	Network Hazards	Tampering	Attack makes network unavailable	Probable Minor Low Risk	Probable Minor Min Risk	Devices must be able to fall back to a non-networked mode of operation, possibly raising alarms if they require manual entry of information to continue (e.g., a glucose pump that needs insulin values)		3
C7.4	Network Hazards	Network Use						
C7.4.1	Network Hazards	Network Use	Inability to push firmware updates	Occasional Minor Low Risk	Occasional Minor Min Risk	Pumps should accept properly authenticated updates from the network	Design Verification. System testing.	6
C7.4.2	Network Hazards	Network Use	Inability to push drug library updates	Occasional Minor Low Risk	Occasional Minor Min Risk	Pumps should accept properly authenticated updates from the network	Design Verification. System testing.	6
C7.5	Network Hazards	Incorrect Measurements						
C7.5.1	Network Hazards	Incorrect Measurements	Pump clock differs from other system clocks	Frequent Major High Risk	Remote Major Low Risk	Synchronize clocks using NTP	Design verification. System testing.	3

C7.5.2	Network Hazards	Incorrect Measurements	pump and EMR use different units, conversion incorrect	Occasional Moderate Medium Risk	Remote Moderate Low Risk	Check units when communicating with other systems.	Design verification. System testing.	6
H8	Drug Library Hazards							
C8.1	Drug Library Hazards	Updating						
C8.1.1	Drug Library Hazards	Updating	Incorrect DL version	Remote Minor Low Risk	Improbable Minor Min Risk	Pump management tools must support tracking which pumps have which version of the drug library. Pumps that are network connected can query to see what the current version is. Pumps not on networks can prompt the user periodically (e.g., once per year) if	Design Verification, Functional Testing, Use Testing	6
C8.1.2	Drug Library Hazards	Updating	DL version not visible	Remote Minor Low Risk	Improbable Minor Min Risk	Make DL version visible to users	Design Verification, Functional Testing, Use Testing	6
C8.2	Drug Library Hazards	Contents						
C8.2.1	Drug Library Hazards	Contents	Drug library too small	Remote Minor Low Risk	Improbable Minor Min Risk	Ensure adequate storage on pump for large drug libraries	Design Verification	6
C8.2.2	Drug Library Hazards	Contents	Non-standard drug names	Probable Minor Low Risk	Probable Minor Min Risk	Use standard drug names where available	Design Verification, Use Testing	6
C8.2.3	Drug Library Hazards	Contents	Non-standard drug concentrations	Probable Minor Low Risk	Probable Minor Min Risk	Use standard drug concentrations where available	Design Verification, Use Testing	6
C8.2.4	Drug Library Hazards	Contents	Fields too small for drug names	Remote Minor Low Risk	Improbable Minor Min Risk	Ensure that field sizes are not a constraint.	Design Verification, Use Testing	6

C8.2.5	Drug Library Hazards	Contents	No support for generic + brand name	Probable Minor Low Risk	Probable Minor Min Risk	Allow drugs to have multiple names or nicknames	Design Verification, Use Testing	6
C8.2.6	Drug Library Hazards	Contents	Inability to list different size bags of one concentration	Probable Minor Low Risk	Probable Minor Min Risk	Allow multiple concentrations and bag sizes for drugs	Design Verification, Use Testing	6
C8.2.7	Drug Library Hazards	Contents	Limited ability to link specific alerts to individual drugs	Probable Minor Low Risk	Probable Minor Min Risk	Allow alerts to be linked to drugs	Design Verification, Use Testing	6
C8.2.8	Drug Library Hazards	Contents	Upper and Lower hard and soft limits are not sufficient to address all administration errors	Probable Major High Risk	Probable Minor Low Risk	Expand drug library to accept other kinds of limits, to match how users intend to use the pump in the use environment	Design Verification, Functional Testing, Use Testing	6

A.2 X-Ray Ventilator Synchronization Application Hazard Analysis

Number	Category	Subcategory	Description	Initial	Predicted	Mitigation	Verification	Source
H1	Energy Hazards							
C1.1	Energy Hazards	Electricity						
C1.1.1	Energy Hazards	Electricity	Possibility of supply voltage leakage when Patient and mains power simultaneously connected to system.	Occasional Major Medium Risk	Improbable Major Min Risk	Ensure system fitted with approved power supply meeting [standard] and that all physically connected devices meet ISO 60601-1	Verify design requirement fulfilled. Ensure GIP User Documentation emphasises need to confirm approval of connected devices.	1
C1.1.2	Energy Hazards	Electricity	Possibility of voltage leakage between physically connected devices.	Occasional Moderate Medium Risk	Improbable Moderate Medium Risk	Ensure all electrical connection ports in system provide electrical isolation between each other to [standard]	Verify design requirement fulfilled and test as part of electrical safety verification testing.	1
C1.1.3	Energy Hazards	Electricity	Supply voltage leakage could occur between applied parts.	Remote Moderate Low Risk	Remote Moderate Low Risk	Design to EN60601-1		1
C1.1.4	Energy Hazards	Electricity	Possibility of shorting due to gas or fluid ingress.	Occasional Moderate Medium Risk	Improbable Moderate Medium Risk	Design system to appropriate IP rating for the environment in which it is to be used	Verify design requirement fulfilled and test as part of environmental testing.	1
C1.1.5	Energy Hazards	Electricity	Shock hazard from frayed system power cord	Occasional Moderate Medium Risk	Remote Moderate Low Risk	Cord must be durable. Build to applicable standards.	Design verification, testing	5
C1.1.6	Energy Hazards	Electricity	AC power is lost	Occasional Moderate Medium Risk	Remote Moderate Low Risk	System must switch to battery power	Design verification, testing	
C1.1.7	Energy Hazards	Electricity	Circuit disconnect and ventilator switches to idle mode	Occasional Moderate Low Risk	Remote Moderate Low Risk	System must alert technician to reconnect circuit	Design verification. User documentation verification.	

C1.2	Energy Hazards	Heat						
C1.2.1	Energy Hazards	Heat	System circuitry gets hot.	Remote Minor Low Risk	Improbable Minor Low Risk	Design system to ensure adequate cooling. Draw attention to the need to work within the operating environment in the User documentation	Design verification. User documentation verification.	1
C1.3	Energy Hazards	Mechanical Force						
C1.3.1	Energy Hazards	Mechanical Force	Accidental movement causes unit to strike patient or user	Occasional Minor Low Risk	Occasional Minor Low Risk	Reduce risk by appropriate design for intended environment. Draw attention to patient safety in user documentation.	”Design output. User documentation verification”	1
C1.4	Energy Hazards	Acoustic Pressure						
C1.4.1	Energy Hazards	Acoustic Pressure	Noise from the system causes discomfort or injury	Remote Negligable Min Risk	Improbable Negligable Min Risk	User Documentation to highlight need for audio alarms to be set at an appropriate level	User documentation verification / review	1
C1.5	Energy Hazards	Vibration	No Hazard Identified					
C1.6	Energy Hazards	Radiation						
C1.6.1	Energy Hazards	Radiation	X-ray machine overexposes patient and/or operator	”Occassional Moderate Medium Risk”	”Improbable Moderate Minimum Risk”	Design system to work with a dead-man switch to avoid unintentional X-ray exposure.	Design verification.	
H2	Biological Hazards							
C2.1	Biological Hazards	Contamination						
C2.1.1	Biological Hazards	Contamination	Battery leak	Remote Moderate Low Risk	Remote Improbable Min Risk	Use appropriate battery and case design.	Design verification.	2

H3	Environmental Hazards							
C3.1	Environmental Hazards	Power Supply						
C3.1.1	Environmental Hazards	Inadequate power supply	The System acts in an unintentional manner when power becomes low	Occasional Major Medium Risk	Improbable Major Min Risk	Design system to monitor power to ensure early notification and if necessary safe shut down	Test unit to specified standards	1
C3.1.2	Environmental Hazards	External battery polarity reversed	The System acts in an unintentional manner when external battery polarity is reversed	Occasional Major Medium Risk	Improbable Major Min Risk	Design system to monitor reverse polarity	Test unit to specified standards	
C3.2	Environmental Hazards	Operation outside of Prescribed Environmental Conditions						
C3.2.1	Environmental Hazards	Operation outside of Prescribed Environmental Conditions	Excessive humidity	Probable Minor Low Risk	Probable Minor Min Risk	Design system to ensure adequate ventilation. Draw attention to the need to work within the operating environment in the User documentation. Humidity sensor and alarm may be necessary.	Design verification. User documentation verification.	2

C3.2.2	Environmental Hazards	Operation outside of Prescribed Environmental Conditions	Temperature extremes	Probable Minor Low Risk	Probable Minor Min Risk	Design system to ensure adequate cooling. Draw attention to the need to work within the operating environment in the User documentation. Temperature sensor and alarm may be necessary.	Design verification. User documentation verification.	2
C3.2.3	Environmental Hazards	Operation outside of Prescribed Environmental Conditions	Noisy environment masks alarm sounds	Probable Moderate Medium Risk	Probable Minor Low Risk	Use auditory and visual alarms. Allow users to modify alarm volume. Maximum alarm volume should be loud enough to be heard in typical environments. Conform to relevant alarm standards.	Design verification.	2
C3.3	Environmental Hazards	Incompatible with other devices						
C3.3.1	Environmental Hazards	Incompatible with other devices	System interferes with nearby electrical systems.	Occasional Major Medium Risk	Improbable Major Min Risk	Design unit to specific standards relating to possible environmental scenarios of intended use.	Test unit to specified standards.	1
C3.4	Environmental Hazards	Electromagnetic Incompatibility						
C3.4.1	Environmental Hazards	Electromagnetic Incompatibility	System not protected against external electromagnetic radiation	Probable Minor Low Risk	Improbable Minor Min Risk	Design product to meet IEC 60601-1-2 EMC requirements.	Design Output	1
H4	Hazards Relating to Use							

C4.1	Hazards Relating to Use	Incorrect Application						
C4.1.1	Hazards Relating to Use	Incorrect Application	The system is used incorrectly	Frequent Moderate High Risk	Improbable Moderate Medium Risk	Instructions for use to include Quick Start guide immediately visible on attempting to use system. Instructions for use to include interactive training materials and remote support via web-site, including links to sites for connected devices.	Verify user instruction materials are as required.	1
C4.2	Hazards Relating to Use	Use by Unskilled or Untrained Personnel						
C4.2.1	Hazards Relating to Use	Use by Unskilled or Untrained Personnel	Users have not had appropriate training.	Occasional Minor Low Risk	Improbable Minor Min Risk	System to only allow authenticated users to use the system. System administrator only to add users once appropriate training given. Ongoing training records to be kept.	”System verification testing. User documentation verification. ”	1
C4.2.2	Hazards Relating to Use	Use by Unskilled or Untrained Personnel	Children and other animals	Probable Major High Risk	Probable Minor Low Risk	Lock out keypad, physically secure power and other connectors, appropriate strain relief for all electrical and fluid connections.	Design Verification. System Testing.	2
C4.3	Hazards Relating to Use	Human Error						
C4.3.1	Hazards Relating to Use	Human Error	The system user is color blind	Remote Minor Low Risk	Improbable Minor Min Risk	Ensure that the design does not rely solely on color-coding for correct operation	Design Output	1

C4.3.2	Hazards Relating to Use	Human Error	Technician enters value using wrong units. E.g., cmH20 instead of millibars.	Frequent Major High Risk	Frequent Minor Medium Risk	Validate user interface design with use testing. For critical infusions, force two different users to program vent and check that they match. Automate programming (e.g., with barcodes) to elimate manual data entry. Prominantly pressure and volume units.	Design Verification	4
C4.3.3	Hazards Relating to Use	Human Error	Technician selects wrong ventilator mode from mode list	Frequent Minor Medium Risk	Frequent Minor Medium Risk	Validate user interface design with use testing. For critical infusions, force two different users to program pump and check that they match. Automate programming (e.g., with barcodes) to elimate manual data entry.	Design Verification	4
C4.3.4	Hazards Relating to Use	Human Error	System connected to the wrong port on the ventilator	Remote Major Low Risk	Remote Major Low Risk	Use distinct connectors for different types of ports. Color code ports and wires	Design Verification. Verify that user manual addresses the issue.	5
C4.3.5	Hazards Relating to Use	Human Error	Use on inappropriate patient	Probable Major High Risk	Remote Major Low Risk	Check patient age, weight, etc. against orders and drug library. Check against barcode input of patient ID bracelet with pertinant information or prompt the caregiver.	Design Verification. System Testing.	2

C4.3.6	Hazards Relating to Use	Human Error	Alarms ignored	Frequent Major High Risk	Occasional Major Medium Risk	Ventilators should minimize alarm fatigue by reducing false or nuisance alarms. Varying levels of alarms can inform caregivers when a condition is life threatening vs. a warning.	Design Verification	5
C4.3.7	Hazards Relating to Use	Human Error	X-ray order is incorrect	Probable Major High Risk	Probable Minor Low Risk	Check against Order report	Design Verification	3
C4.3.8	Hazards Relating to Use	Human Error	X-ray workflow doesn't match User workflow	?	?	Design must match how x-ray is used in intended environment.	Design Verification, Use Testing	5
C4.3.9	Hazards Relating to Use	Human Error	Not confirming start of ventilation. E.g. ventilator remains in set-up mode.	Frequent Major High Risk	Occasional Minor Low Risk	Alarm, caregiver training	Design Verification. Functional Testing. Verify that user manual addresses the issue.	6
C4.4	Hazards Relating to Use	Incorrect Measurements						
C4.4.1	Hazards Relating to Use	Incorrect Measurements	user interface does not truly represent an item(s) of data correctly.	Probable Moderate Medium Risk	Remote Moderate Low Risk	Consult with users to match data representations to their mental models	Design verification. Use Testing. System testing.	1
C4.4.2	Hazards Relating to Use	Incorrect Measurements	Pressure Measurment Inopperative	Occasional Moderate Medium Risk	Remote Moderate Low Risk	Add extra pressure sensor to tubing. Recalibrate system.	Design verification. Use Testing. System testing	
C4.4.3	Hazards Relating to Use	Incorrect Measurements	Ventilator doesn't account for residual volume in the tubing	Probable Moderate Medium Risk	Remote Moderate Low Risk	Ventilation information entered or set automatically then used to adjust programming	Design verification. System testing.	5

C4.5	Hazards Relating to Use	Corrupted or Incomplete Data						
C4.5.1	Hazards Relating to Use	Corrupted or Incomplete Data	An item of data from a medical device is delayed, not received or corrupted between patient and user display.	Probable Moderate Medium Risk	Remote Moderate Low Risk	System topology to be designed with robust error checking, data and communications verification with extensive error and notice output with clear indication of error location.	"Design output verification. System testing verification. User acceptance verification. "	1
C4.5.2	Hazards Relating to Use	Human Error	Technician enters value using wrong units. E.g., cmH20 instead of millibars.	Frequent Major High Risk	Frequent Minor Medium Risk	Validate user interface design with use testing. For critical infusions, force two different users to program vent and check that they match. Automate programming (e.g., with barcodes) to elimate manual data entry. Prominantly pressure and volume units.	Design Verification	4
C4.5.3	Hazards Relating to Use	Human Error	Technician selects wrong ventilator mode from mode list	Frequent Minor Medium Risk	Frequent Minor Medium Risk	Validate user interface design with use testing. For critical infusions, force two different users to program pump and check that they match. Automate programming (e.g., with barcodes) to elimate manual data entry.	Design Verification	4

C4.5.4	Hazards Relating to Use	Human Error	System connected to the wrong port on the ventilator	Remote Major Low Risk	Remote Major Low Risk	Use distinct connectors for different types of ports. Color code ports and wires	Design Verification. Verify that user manual addresses the issue.	5
C4.5.5	Hazards Relating to Use	Human Error	Use on inappropriate patient	Probable Major High Risk	Remote Major Low Risk	Check patient age, weight, etc. against orders and drug library. Check against barcode input of patient ID bracelet with pertinant information or prompt the caregiver.	Design Verification. System Testing.	2
C4.5.6	Hazards Relating to Use	Human Error	Alarms ignored	Frequent Major High Risk	Occasional Major Medium Risk	Ventilators should minimize alarm fatigue by reducing false or nuisance alarms. Varying levels of alarms can inform caregivers when a condition is life threatening vs. a warning.	Design Verification	5
C4.5.7	Hazards Relating to Use	Human Error	Alarm turned off	Frequent Major High Risk	Occasional Major Medium Risk		Design Verification	
C4.5.8	Hazards Relating to Use	Human Error	X-ray order is incorrect	Probable Major High Risk	Probable Minor Low Risk	Check against Order report	Design Verification	3
C4.5.9	Hazards Relating to Use	Human Error	X-ray workflow doesn't match User workflow	?	?	Design must match how x-ray is used in intended environment.	Design Verification, Use Testing	5
C4.5.10	Hazards Relating to Use	Human Error	Not confirming start of ventilation. E.g. ventilator remains in set-up mode.	Frequent Major High Risk	Occasional Minor Low Risk	Alarm, caregiver training	Design Verification. Functional Testing. Verify that user manual addresses the issue.	6
C4.6	Hazards Relating to Use	Incorrect Measurements						

C4.6.1	Hazards Relating to Use	Incorrect Measurements	User interface does not truly represent an item(s) of data correctly.	Probable Moderate Medium Risk	Remote Moderate Low Risk	Consult with users to match data representations to their mental models	Design verification. Use Testing. System testing.	1
C4.6.2	Hazards Relating to Use	Incorrect Measurements	Pressure Measurment Inopperative	Occasional Moderate Medium Risk	Remote Moderate Low Risk	Add extra pressure sensor to tubing. Recalibrate system.	Design verification. Use Testing. System testing	
C4.6.3	Hazards Relating to Use	Incorrect Measurements	Ventilator doesn't account for residual volume in the tubing	Probable Moderate Medium Risk	Remote Moderate Low Risk	Ventilation information entered or set automatically then used to adjust programming	Design verification. System testing.	5
C4.7	Hazards Relating to Use	Corrupted or Incomplete Data						
C4.7.1	Hazards Relating to Use	Corrupted or Incomplete Data	An item of data from a medical device is delayed, not received or corrupted between patient and user display.	Probable Moderate Medium Risk	Remote Moderate Low Risk	System topology to be designed with robust error checking, data and communications verification with extensive error and notice output with clear indication of error location.	"Design output verification. System testing verification. User acceptance verification. "	1
C4.8	Hazards Relating to Use	Accessory Misuse						

C4.8.1	Hazards Relating to Use	Accessory Misuse	An accessory is not connected to the system correctly.	Probable Minor Low Risk	Improbable Minor Min Risk	User Interface to differentiate between attached devices i.e those that are physically connected or can be seen by the system and connected devices where the device interface connection verification routine has passed.	”Design output verification. Device interface verification ”	1
C4.8.2	Hazards Relating to Use	Accessory Misuse	An accessory is not used in accordance with its intended use or user documentation.	Probable Moderate Medium Risk	Remote Moderate Low Risk	Train Users, ensure that documentation is adequate.	”User documentation verification. System verification. ”	1
C4.9	Hazards Relating to Use	Tampering						
C4.9.1	Hazards Relating to Use	Tampering	Unauthorized person changes system settings	Remote Major Low Risk	Improbable Major Min Risk	Lock out system UI, require passcode to change settings. Don’t ship systems with a common default passcode.	Design verification	2
H5	Functional Failure, Maintenance, and Aging Hazards							
C5.1	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics						

C5.1.1	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Flow rate does not match programmed rate	Remote Major Low Risk	Improbable Major Min Risk	Design ventilator hardware to be robust. Minimize opportunities for freeflow of air. If feasible, use a flow sensor or other means of directly measuring fluid flow rate.	System verification. Functional testing.	2
C5.1.2	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Excessive variation in flow rate	Remote Major Low Risk	Improbable Major Min Risk	Limit minimum ventilator flow rate in accordance with ventilator hardware capabilities.	Design verification. Functional testing.	2
C5.1.3	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Flow back; reflux within device	Remote Major Low Risk	Improbable Major Min Risk	Design ventilator hardware and accessories (infusion set) to be robust and enforce one-way flow of fluid	System verification. Functional testing.	2
C5.1.4	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Failure of alarm	Remote Major Low Risk	Improbable Major Min Risk	Design alarm hardware to be robust. Use multiple alarm modalities (audio and visual). Test hardware during power-on self test, e.g., using a microphone to check if the speaker functions correctly.	Design verification. Functional testing.	2
C5.1.5	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Incorrect, loose, or corroded connections between ventilator components	Remote Major Low Risk	Improbable Major Min Risk	Detect unexpected loss of connectivity and alarm.	Design verification. Functional testing.	2
C5.1.6	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Realtime Clock (RTC) failure	Remote Major Low Risk	Improbable Major Min Risk	Battery backup of clock, power-on self test.	Design verification. Functional testing.	2

C5.1.7	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Stuck or improperly debounced keys	Remote Major Low Risk	Improbable Major Min Risk	Power-on self test, drop-down library in case of misentry of programming parameters	Design verification. Functional testing.	2
C5.1.8	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Sensor Failure	Remote Major Low Risk	Improbable Major Min Risk	Power-on self test, Test sensor values for stuck-on, stuck-off, and repeating failures	Design verification. Functional testing.	2
C5.1.9	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Can't easily back up in programming, have to re-enter entire sequence	Probable Minor Low Risk	Improbable Minor Low Risk	UI should allow users to easily correct mistakes	Design Verification, Use testing	5
C5.1.10	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Can't read log data from ventilator	Probable Minor Low Risk	Improbable Minor Low Risk	Ventilators should use standard log format, manufacturers must provide software to download and interpret pump logs.	Design verification. Functional testing, Use testing	5
C5.1.11	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Inadequate battery life	Probable Minor Low Risk	Improbable Minor Low Risk	Ventilators must accurately calculate remaining battery life as batteries age.	Design verification. Functional testing.	5
C5.1.12	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Screen not readable	Probable Minor Low Risk	Improbable Minor Low Risk		Design Verification, Use Testing	6
C5.1.13	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	"Apnea: The set apnea interval has elapsed without the ventilator, patient, or operator triggering a breath."	Probable Major High Risk	"Occasional Moderate Medium Risk"	Check patient and settings. Resets when patient initiates 2 consecutive breaths.		

C5.1.14	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	"Compliance Limited: Compliance volume required to compensate delivery of a volume controlled breath exceeds the maximum allowed for 3 of the last 4 breaths."	Occasional Minor Low Risk	Remote Minor Low Risk	"Inspired volume may be < set. Check patient and circuit type."	Design Verification
C5.1.15	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	"Inc. O2%: The O2% measured during any phase of a breath cycle is 7% (12% during the first hour of operation) or more above the O2% setting for at least 30 seconds. (These percentages increase by 5% for 4 minutes following a decrease in the O2% setting.)"	Probable Major High Risk	"Occasional Moderate Medium Risk"	"Check patient, gas sources, O2 analyzer and ventilator."	Design Verification
C5.1.16	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Inc. PCOMP	Probable Major High Risk	"Occasional Moderate Medium Risk"	Check settings, changes in patient's R and C.	Design Verification

C5.1.17	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Inc. PMEAN	Probable Major High Risk	”Occasional Moderate Medium Risk”	Check settings, changes in patient’s R and C.	Design Verification
C5.1.18	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	”Inc. PPEAK : Measured airway pressure ? set limit. Ventilator truncates current breath unless already in exhalation.”	Probable Major High Risk	Occasional Major Medium Risk	Check settings, changes in patient’s R and C.	Design Verification
C5.1.19	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Inc. PVENT :	Probable Major High Risk	Occasional Major Medium Risk	Check settings, changes in patient’s R and C.	Design Verification
C5.1.20	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	”Inc. VTE : Exhaled tidal volume ? set limit. Alarm updated whenever exhaled tidal volume is recalculated.”	Probable Moderate Medium Risk	”Occasional Moderate Medium Risk”	Check settings, changes in patient’s R and C.	Design Verification
C5.1.21	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	”Inc. Exp Flow : Expiratory minute volume ? set limit. Alarm updated whenever an exhaled minute volume is recalculated. Possible dependent alarm: Inc. VTE”	Probable Moderate Medium Risk	”Occasional Moderate Medium Risk”	Check patient and settings.	Design Verification

C5.1.22	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	”Inc. VTI MAND: Inhaled mandatory tidal volume ? set limit. Alarm updated whenever inhaled mandatory tidal volume is recalculated.”	Probable Moderate Medium Risk	”Occasional Moderate Medium Risk”	Check for leaks, changes in patient’s R and C	Design Verification	
C5.1.23	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	”Inc. VTI SPONT: Inhaled spontaneous tidal volume ? set limit. ”	Probable Moderate Medium Risk	”Occasional Moderate Medium Risk”	”Check patient and settings. Alarm updated whenever inhaled spontaneous tidal volume is recalculated.”	Design Verification	
C5.1.24	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	”Inc. fTOT: Total respiratory rate ? set limit. Alarm updated at the beginning of each inspiration. ”	Probable Major High Risk	Occasional Major Medium Risk	”Check patient and settings. Reset when measured respiratory rate falls below the alarm limit.”		
C5.1.25	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	”Inspiration Too Long: Inspiratory time for spontaneous breath ? IBW-based limit. Ventilator transitions to exhalation. Active only when Vent Type is INVASIVE.”	Probable Major High Risk	Occasional Major Medium Risk	”Check patient. Check for leaks. Resets when TI falls below IBW-based limit. ”	”Design output. Verify user instructions are as required. ”	1

C5.1.26	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	"Dec. O2%: The O2% measured during any phase of a breath cycle is 7% (12% during the first hour of operation) or more below the O2% setting for at least 30 seconds, or below 18%. (These percentages increase by 5% for 4 minutes following an increase in the "	Probable Major High Risk	Occasional Major Medium Risk	Check patient, gas sources, O2 analyzer and ventilator	Design Verification
C5.1.27	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	"Dec. VTE MAND: Exhaled mandatory tidal volume ? set limit."	Probable Major High Risk	Occasional Major Medium Risk	"Check for leaks, changes in patient's R and C Alarm updated whenever exhaled mandatory tidal volume is recalculated."	Design Verification
C5.1.28	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	"Dec. VTE SPONT: Exhaled spontaneous tidal volume ? set limit. "	Probable Major High Risk	Occasional Major Medium Risk	"Check patient and settings. Alarm updated whenever exhaled spontaneous tidal volume is recalculated."	Design Verification
C5.1.29	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	"Dec. Exp Flow: Total minute volume ? set limit. "	Probable Major High Risk	Occasional Major Medium Risk	"Check patient and settings. Alarm updated whenever exhaled minute volume is recalculated."	Design Verification

C5.1.30	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Volume Not Delivered	Probable Major High Risk	Occasional Major Medium Risk		Design Verification
C5.1.31	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	PPS-Insp. > 4s	Probable Moderate Medium Risk	"Remote Minor Low Risk"	Check the ventilator for leaks	Design Verification
C5.1.32	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	PPS-Insp. > 1.5s	Probable Minor Low Risk	Improbable Minor Low Risk	Check the ventilator for leaks	Design Verification
C5.1.33	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Pressure limited	Probable Minor Low Risk	Improbable Minor Low Risk	Check the condition of the patient	Design Verification
C5.1.34	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Pulse rate high	Probable Moderate Medium Risk	"Remote Minor Low Risk"	Check the condition of the patient	Design Verification
C5.1.35	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Pulse rate low	Probable Moderate Medium Risk	"Remote Minor Low Risk"	Check the condition of the patient	Design Verification
C5.1.36	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Rotary knob xx overused?	Probable Minor Low Risk	Improbable Minor Low Risk	Disconnect patient from ventilator and continue ventilation with another device	Design Verification

C5.1.37	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	SpO2 high	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Check the condition of the patient	
C5.1.38	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	SpO2 low	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Check the condition of the patient	
C5.1.39	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Tidal volume high	Probable Minor Low Risk	Improbable Minor Low Risk	Check hose system connections for leaks.	
C5.1.40	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	Vol. not constant, pressure limited	Probable Minor Low Risk	Improbable Minor Low Risk	Prolong inspiratory time, increase inspiratory flow, and increase pressure limit.	
C5.1.41	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	ASB>1.5 s?	Probable Minor Low Risk	Improbable Minor Low Risk	Test Ventilation system for leaks.	
C5.1.42	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	ASB>Tinsp?	Probable Minor Low Risk	Improbable Minor Low Risk	Test Ventilation system for leaks.	
C5.1.43	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	etCO2 High	Probable Minor High Risk	Improbable Minor Low Risk	Check condition of patient. Check Ventialtion pattern, correct alarm limit if necessary. Perform CO2 Zero if applicable.	

C5.1.44	Functional Failure, Maintenance, and Aging Hazards	Inadequacy of Performance and Characteristics	etCO2 Low	Probable Minor High Risk	Improbable Minor Low Risk	Check condition of patient. Check Ventialtion pattern, correct alarm limit if necessary. Perform CO2 Zero if applicable.		
C5.2	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.						
C5.2.1	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	The lack of training for service personnel impacts corrective service actions.	Probable Minor Low Risk	Improbable Minor Min Risk	Design system to be user serviceable to a certain extent. Give details of all service activities and their correct operation in user instructions.	Verify that User Manual emphasizes teaching all users emergency protocol.	3
C5.2.2	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Failure to trigger mechanical switch of X-ray Machine	Occasional Major Medium Risk	Remote Major Low Risk	UI must enforce verification of mechanical trigger connections where appropriate. User manuals should emphasize importance.	Design Verification	2
C5.2.3	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Compressor Inoperative: Compressor ready indicator turns off.	Occasional Minor Low Risk	Remote Minor Low Risk	Resets when full AC power is restored.		
C5.2.4	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Device Alert: Background checks have detected a problem.	Occasional Minor Low Risk	Remote Minor Low Risk	”Replace and service ventilator. Resets when ventilator passes EST”		

C5.2.5	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Inoperative Battery: Inadequate charge or nonfunctional battery system. BPS installed but not functioning. Resets when BPS is functional.	Occasional Minor Low Risk	Remote Minor Low Risk	Service/replace battery.	
C5.2.6	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Low Battery: Operational time $<$ 2 minutes.	Occasional Minor Low Risk	Remote Minor Low Risk	"Replace or allow recharge. Resets when BPS has more than approximately 2 minutes of operational time remaining."	
C5.2.7	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	No Air Supply: Ventilation continues as set. Only O2 available. Operator- set O2% equals 100%.	Probable Major High Risk	Occasional Major Medium Risk	Check patient and air source.	
C5.2.8	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Procedure Error: Patient connected before setup complete.	Probable Major High Risk	Occasional Major Medium Risk	"Provide alternate ventilation. Complete setup process. Ventilator begins safety ventilation. Resets when ventilator startup procedure is complete."	
C5.2.9	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	"Screen Block: Possible blocked beam or touch screen fault."	Probable Moderate Medium Risk	"Occasional Moderate Medium Risk"	"Remove obstruction or service ventilator. Resets when ventilator passes EST or when blockage is removed."	

C5.2.10	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Severe Occlusion: Little/no ventilation	Probable Major High Risk	Occasional Major Medium Risk	Check patient. Provide alternate ventilation. Clear occlusions; drain circuit.		
C5.2.11	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	O2 measurement inop.	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Calibrate O2 sensor.		
C5.2.12	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	O2 monitoring off	Probable Minor Low Risk	Improbable Minor Low Risk	Switch O2 monitoring on again.		3
C5.2.13	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	O2 supply down	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Make sure supply pressure is greater than 3 bar.		
C5.2.14	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	O2 supply pressure high	Probable Minor Low Risk	Improbable Minor Low Risk	Make sure supply pressure is less than 6 bar.		
C5.2.15	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	O2 therapy active	Probable Minor Low Risk	Improbable Minor Low Risk	Switch off O2 therapy.		
C5.2.16	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	PEEP high	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Check hose system and expiration valve.		

C5.2.17	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	PEEP valve inop.	Probable Moderate Medium Risk	"Remote Minor Low Risk"	Disconnect patient from ventilator and continue ventilation with another device	
C5.2.18	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Rotary knob xx failed	Probable Minor Low Risk	Improbable Minor Low Risk	Disconnect patient from ventilator and continue ventilation with another device	
C5.2.19	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	SpO2 meas. inop.	Probable Moderate Medium Risk	"Remote Minor Low Risk"	Change sensor.	
C5.2.20	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	SpO2 sensor?	Probable Moderate Medium Risk	"Remote Minor Low Risk"	Reconnect the sensor and test.	
C5.2.21	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Temperature meas. inop.	Probable Moderate Medium Risk	"Remote Minor Low Risk"	Change temperature sensor.	
C5.2.22	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Temperature sensor?	Probable Moderate Medium Risk	"Remote Minor Low Risk"	Reconnect the sensor and test.	
C5.2.23	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Clean CO2 cuvette	Probable Moderate Medium Risk	"Remote Minor Low Risk"	Clean the CO2 cuvette and reconnect	

C.5.2.24	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	CO2 measurement inoperational	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Reconnect CO2 sensor and test. Replace if problem persists.	
C.5.2.25	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	CO2 sensor?	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Reconnect CO2 sensor and test. Replace if problem persists.	
C.5.2.26	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Fan Failure	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Check fan function,clean or replace cooling air filter.	
C.5.2.27	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Fan Malfunction	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Check fan function,clean or replace cooling air filter.	
C.5.2.28	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	FiO2 high	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Calibrate O2 sensor.	
C.5.2.29	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	FiO2 low	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Calibrate O2 sensor.	
C.5.2.30	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Flow Sensor	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Calibrate Flow sensor.	

C.5.2.31	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	ILV Sync. Inoperation	Probable Moderate Medium Risk	”Remote Minor Low Risk”		
C.5.2.32	Functional Failure, Maintenance, and Aging Hazards	Inadequate or Improper Maintenance.	Flow Measurement Inoperational	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Calibrate Flow sensor.	
H6	System Hazards						
C6.1	System Hazards	Trigger Button/ Dead Man Switch					
C6.1.1	System Hazards	Trigger Button/ Dead Man Switch	Failure to abort X-ray during Emergency	Probable Moderate Medium Risk	”Remote Minor Low Risk”	Training of X-ray to abort image acquisition in emergency. UI Assistance for the same.	
C6.1.2	System Hazards	Trigger Button/ Dead Man Switch	A rapid series of trigger requests leads to overexposure to and overdosage of radiation	Frequent Major High Risk	Probable Major High Risk	System Timer of 30 s aborts application after timer expires to avoid repeat triggering.	
C6.2	System Hazards	Tampering					
C6.2.1	System Hazards	Tampering	Unauthorized person removes or tampers with mechanical X-ray trigger assembly.	Occasional Major Medium Risk	Occasional Minor Low Risk	Physically lock the assembly. Require lock to be closed before starting application. Alarm if application is run with switch unlocked.	
H7	Network Hazards						

C7.3	Network Hazards	Tampering						
C7.3.1	Network Hazards	Tampering	Unauthorized or illegitimate firmware update pushed	Remote Major Low Risk	Improbable Major Min Risk	Firmware updates must be authenticated by the device before installation.	Design verification. Functional testing	2
C7.4	Network Hazards	Network Use						
C7.4.1	Network Hazards	Network Use	Inability to push firmware updates	Occasional Minor Low Risk	Occasional Minor Min Risk	Systems should accept properly authenticated updates from the network	Design Verification. System testing.	6
C7.5	Network Hazards	Incorrect Measurements						
C7.5.1	Network Hazards	Incorrect Measurements	System clock differs from other system clocks	Frequent Major High Risk	Remote Major Low Risk	Synchronize clocks using NTP	Design verification. System testing.	3
C7.5.2	Network Hazards	Incorrect Measurements	System and EMR use different units, conversion incorrect	Occasional Moderate Medium Risk	Remote Moderate Low Risk	Check units when communicating with other systems.	Design verification. System testing.	6

Appendix B

UPPAAL Export Example

CFR is a small EFSM which is used as an example for the UPPAAL exporter. It is shown graphically in Figure B.1. This is an example of a single EFSM which does not use communications channels.

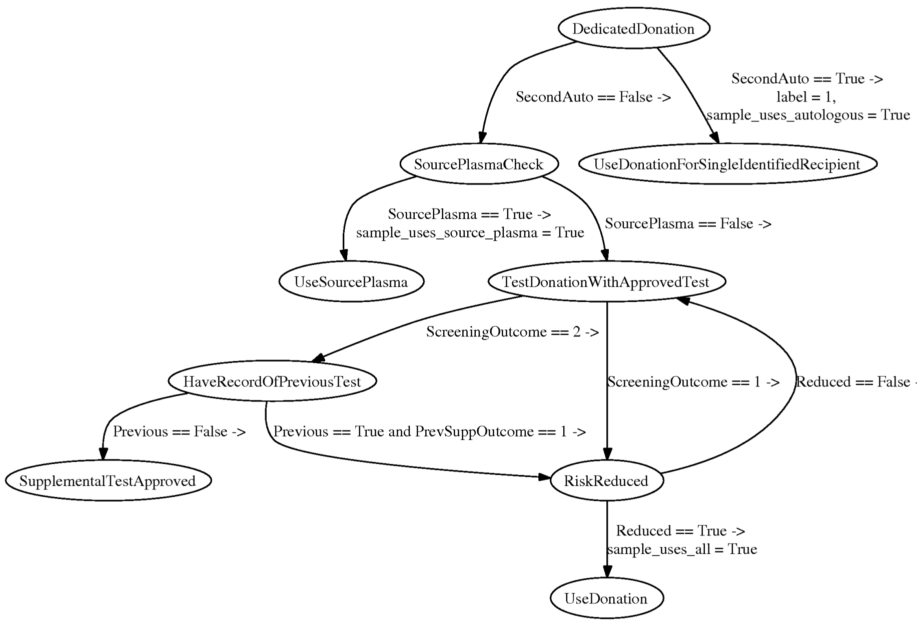


Figure B.1: CFR example EFSM

```
<?xml version="1.0"?>
<!DOCTYPE nta PUBLIC "-//Uppaal Team//DTD Flat System 1.0//EN"
"http://www.docs.uu.se/docs/rtmv/uppaal/xml/flat-1_0.dtd">
<nta>
<declaration>
</declaration>

<template>
<name>CFR</name>

<declaration>
int[0,1] LicensedFacility := 0;
int[0,1] PrevReact := 0;
int[0,1] ShownSuitable := 0;
int[0,1] Emergency := 0;
int[0,1] Manufacturing := 0;
int[0,1] SecondAuto := 0;
int[0,1] SourcePlasma := 0;
int[0,1] MedDevice := 0;
int[0,1] AutologousUse := 0;
int[0,1] aa0 := 0;
int[0,1] aa1 := 0;
int[0,1] First30 := 0;
int[0,1] aa2 := 0;
int[0,2] ScreeningOutcome := 0;
int[0,1] Reduced := 0;
int[0,1] Previous := 0;
int[0,2] PrevSuppOutcome := 0;
```

```
int[0,1] Supplemental := 0;
int[0,2] SuppOutcome := 0;
int[0,1] Research := 0;
int[0,1] sample_uses_all := 0;
int[0,1] sample_uses_manufacturing := 0;
int[0,1] sample_uses_research := 0;
int[0,1] sample_uses_autologous := 0;
int[0,1] sample_uses_source_plasma := 0;
int[0,1] sample_uses_device := 0;
int[0,1] donor_uses := 0;
int[0,5] label := 0;

</declaration>
<location id="id0">
<name> SourcePlasmaCheck </name>
</location>

<location id="id1">
<name> IsItAnEmergency </name>
</location>

<location id="id2">
<name> ShipToAllowsAllogenic </name>
</location>

<location id="id3">
<name> IsThisALicensedFacility </name>
</location>
```

```
<location id="id4">
<name> UseDonationForSingleIdentifiedRecipient </name>
</location>

<location id="id5">
<name> AllowAllogenic </name>
</location>

<location id="id6">
<name> UseAutologousDonation </name>
</location>

<location id="id7">
<name> RiskReduced </name>
</location>

<location id="id8">
<name> ResearchUse </name>
</location>

<location id="id9">
<name> UseForMedicalDevices </name>
</location>

<location id="id10">
<name> DedicatedDonation </name>
</location>
```

```
<location id="id11">
<name> MedicalDevice </name>
</location>

<location id="id12">
<name> DoNotShipOrUseRejectDonor </name>
</location>

<location id="id13">
<name> Stop </name>
</location>

<location id="id14">
<name> HaveRecordOfPreviousTest </name>
</location>

<location id="id15">
<name> SupplementalTestApproved </name>
</location>

<location id="id16">
<name> Autologous </name>
</location>

<location id="id17">
<name> PreviouslyReactive </name>
</location>
```

```
<location id="id18">
<name> UseDonation </name>
</location>

<location id="id19">
<name> UseForResearch </name>
</location>

<location id="id20">
<name> FurtherManufacturing </name>
</location>

<location id="id21">
<name> AutologousDonation </name>
</location>

<location id="id22">
<name> StopCanNotPerformTesting </name>
</location>

<location id="id23">
<name> UseSourcePlasma </name>
</location>

<location id="id24">
<name> TestWithSupplementalTest </name>
</location>
```

```
<location id="id25">
<name> TestDonationWithApprovedTest </name>
</location>

<location id="id26">
<name> DoNotUseUnit </name>
</location>

<init ref="id3"/>

<transition>
<source ref = "id3"/>
<target ref = "id17"/>
<label kind="guard">LicensedFacility == 1</label>
</transition>

<transition>
<source ref = "id3"/>
<target ref = "id22"/>
<label kind="guard">LicensedFacility == 0</label>
</transition>

<transition>
<source ref = "id17"/>
<target ref = "id22"/>
<label kind="guard">PrevReact == 1 and ShownSuitable == 0</label>
</transition>
```

```
<transition>
<source ref = "id17"/>
<target ref = "id1"/>
<label kind="guard">PrevReact == 1 and ShownSuitable == 1</label>
</transition>

<transition>
<source ref = "id17"/>
<target ref = "id1"/>
<label kind="guard">PrevReact == 0</label>
</transition>

<transition>
<source ref = "id1"/>
<target ref = "id20"/>
<label kind="guard">Emergency == 0</label>
</transition>

<transition>
<source ref = "id1"/>
<target ref = "id13"/>
<label kind="guard">Emergency == 1</label>
<label kind="action">sample_uses_all := 1</label>
</transition>

<transition>
<source ref = "id20"/>
```

```
<target ref = "id13"/>
<label kind="guard">Manufacturing == 1</label>
<label kind="action">sample_uses_manufacturing := 1</label>
</transition>

<transition>
<source ref = "id20"/>
<target ref = "id10"/>
<label kind="guard">Manufacturing == 0</label>
</transition>

<transition>
<source ref = "id10"/>
<target ref = "id0"/>
<label kind="guard">SecondAuto == 0</label>
</transition>

<transition>
<source ref = "id10"/>
<target ref = "id4"/>
<label kind="guard">SecondAuto == 1</label>
<label kind="action">label := 1 , sample_uses_autologous := 1</label>
</transition>

<transition>
<source ref = "id0"/>
<target ref = "id23"/>
<label kind="guard">SourcePlasma == 1</label>
```

```
<label kind="action">sample_uses_source_plasma := 1</label>
</transition>

<transition>
<source ref = "id0"/>
<target ref = "id11"/>
<label kind="guard">SourcePlasma == 0</label>
</transition>

<transition>
<source ref = "id11"/>
<target ref = "id9"/>
<label kind="guard">MedDevice == 1</label>
<label kind="action">label := 2 , sample_uses_device := 1</label>
</transition>

<transition>
<source ref = "id11"/>
<target ref = "id21"/>
<label kind="guard">MedDevice == 0</label>
</transition>

<transition>
<source ref = "id21"/>
<target ref = "id25"/>
<label kind="guard">AutologousUse == 0</label>
</transition>
```

```
<transition>
<source ref = "id21"/>
<target ref = "id16"/>
<label kind="guard">AutologousUse == 1</label>
</transition>

<transition>
<source ref = "id16"/>
<target ref = "id5"/>
<label kind="guard">aa0 == 0</label>
</transition>

<transition>
<source ref = "id16"/>
<target ref = "id25"/>
<label kind="guard">aa0 == 1</label>
</transition>

<transition>
<source ref = "id5"/>
<target ref = "id25"/>
<label kind="guard">aa1 == 0</label>
</transition>

<transition>
<source ref = "id5"/>
<target ref = "id2"/>
<label kind="guard">aa1 == 1</label>
```

```
</transition>

<transition>
<source ref = "id2"/>
<target ref = "id25"/>
<label kind="guard">aa2 == 1 and First30 == 1</label>
</transition>

<transition>
<source ref = "id2"/>
<target ref = "id6"/>
<label kind="guard">aa2 == 0</label>
<label kind="action">label := 3 , sample_uses_autologous := 1</label>
</transition>

<transition>
<source ref = "id25"/>
<target ref = "id14"/>
<label kind="guard">ScreeningOutcome == 2</label>
</transition>

<transition>
<source ref = "id25"/>
<target ref = "id7"/>
<label kind="guard">ScreeningOutcome == 1</label>
</transition>

<transition>
```

```
<source ref = "id7"/>
<target ref = "id25"/>
<label kind="guard">Reduced == 0</label>
</transition>

<transition>
<source ref = "id14"/>
<target ref = "id7"/>
<label kind="guard">Previous == 1 and PrevSuppOutcome == 1</label>
</transition>

<transition>
<source ref = "id14"/>
<target ref = "id15"/>
<label kind="guard">Previous == 0</label>
</transition>

<transition>
<source ref = "id15"/>
<target ref = "id24"/>
<label kind="guard">Supplemental == 1</label>
</transition>

<transition>
<source ref = "id24"/>
<target ref = "id18"/>
<label kind="guard">SuppOutcome == 1</label>
<label kind="action">sample_uses_all := 1</label>
```

```
</transition>

<transition>
<source ref = "id7"/>
<target ref = "id18"/>
<label kind="guard">Reduced == 1</label>
<label kind="action">sample_uses_all := 1</label>
</transition>

<transition>
<source ref = "id15"/>
<target ref = "id26"/>
<label kind="guard">Supplemental == 0</label>
</transition>

<transition>
<source ref = "id24"/>
<target ref = "id8"/>
<label kind="guard">SuppOutcome == 2</label>
<label kind="action">label := 4</label>
</transition>

<transition>
<source ref = "id8"/>
<target ref = "id19"/>
<label kind="guard">Research == 1</label>
<label kind="action">label := 5 , sample_uses_research := 1</label>
</transition>
```

```
<transition>
<source ref = "id8"/>
<target ref = "id12"/>
<label kind="guard">Research == 0</label>
<label kind="action">donor_uses := 0</label>
</transition>

</template>

<instantiation>
</instantiation>

<system>
        system CFR; </system>
</nta>
```

Bibliography

[1] D. Alonso, J. Plourde, S. Weininger, and J. M. Goldman. Web-based clinical scenario repository (CSR). In *Poster Presentation at the Society for Technology in Anesthesia Annual Meeting*, 2014.

[2] R. Alur, D. Arney, E. L. Gunter, I. Lee, J. Lee, W. Nam, F. Pearce, S. V. Albert, and J. Zhou. Formal specifications and analysis of the computer-assisted resuscitation algorithm (cara) infusion pump control system. *Software tools for technology transfer*, 5(4):308–319, 2004.

[3] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*, pages 322–335, Berlin, Heidelberg, 1990. Springer-Verlag.

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

[5] D. Arney, J. Goldman, I. Lee, E. Llukacej, and S. Whitehead. Use case demonstration: X-ray/ventilator. In *High Confidence Medical Devices, Software, and Systems*

and Medical Device Plug-and-Play Interoperability, 2007, page 160, June 2007.

[6] D. Arney, J. M. Goldman, S. F. Whitehead, and I. Lee. Synchronizing an x-ray and anesthesia machine ventilator: A medical device interoperability case study. In *BIODEVICES 2009*, pages 52 – 60, January 2009.

[7] D. Arney, J. M. Goldman, S. F. Whitehead, and I. Lee. *Improving Patient Safety with X-Ray and Anesthesia Machine Ventilator Synchronization: A Medical Device Interoperability Case Study*, pages 96–109. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

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

[9] D. Arney, M. Pajic, J. M. Goldman, I. Lee, R. Mangharam, and O. Sokolsky. Toward patient safety in closed-loop medical device systems. In *ACM/IEEE 1st International Conference on Cyber-Physical Systems, ICCPS '10, Stockholm, Sweden, April 12-15, 2010*, pages 139–148, 2010.

[10] D. Arney, J. Plourde, and J. Goldman. Openice medical device interoperability platform overview and requirement analysis. 63, 01 2017.

[11] D. Arney, J. Plourde, R. Schrenker, P. Mattegunta, S. F. Whitehead, and J. M. Goldman. Design pillars for medical cyber-physical system middleware. In *5th Workshop on Medical Cyber-Physical Systems, MCPS 2014, Berlin, Germany, April 14, 2014*, pages 124–132, 2014.

[12] D. Arney, K. Venkatasubramanian, O. Sokolsky, and I. Lee. Biomedical devices and systems security. In *Proc. of 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC '11)*, September 2011.

[13] D. Arney, S. Weininger, S. F. Whitehead, and J. M. Goldman. Supporting medical device adverse event analysis in an interoperable clinical environment: Design of a data logging and playback system. In *Proceedings of the 2nd International Conference on Biomedical Ontology, Buffalo, NY, USA, July 26-30, 2011*, 2011.

[14] D. E. Arney, R. Jetley, P. Jones, I. Lee, A. Ray, O. Sokolsky, and Y. Zhang. Generic infusion pump hazard analysis and safety requirements version 1.0. Technical report, University of Pennsylvania, February 2009. Department of Computer and Information Science Technical Report No. MS-CIS-08-31.

[15] P. Asare, D. Cong, S. G. Vattam, B. Kim, A. L. King, O. Sokolsky, I. Lee, S. Lin, and M. Mullen-Fortino. The medical device dongle: an open-source standards-based platform for interoperable medical device connectivity. In *ACM International Health Informatics Symposium, IHI '12, Miami, FL, USA, January 28-30, 2012*, pages 667–672, 2012.

[16] ASTM F2761-09(2013). 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. http://www.astm.org/Standards/F2761.htm.

[17] G. Behrmann, A. David, and K. G. Larsen. A tutorial on uppaal. pages 200–236. Springer, 2004.

[18] C. Callan. *Patient-controlled analgesia*, chapter An analysis of complaints and complications with patient-controlled analgesia, pages 139–50. Blackwell Scientific Publications, 1990.

[19] T. Carpenter, J. Hatcliff, and E. Y. Vasserman. A reference separation architecture for mixed-criticality medical and iot devices. In *Proceedings of the 1st ACM Workshop on the Internet of Safe Things, SafeThings@SenSys 2017, Delft, The Netherlands, November 5, 2017*, pages 14–19, 2017.

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

[21] M. R. Cohen, R. J. Weber, and J. Moss. Patient-controlled analgesia: Making it safer for patients. Technical report, Institute for Safe Medicine Practices.

[22] J. Commission. Sentinel event alert issue 33: Patient controlled analgesia by proxy. http://www.jointcommission.org/sentinelevents/sentineleventalert/sea_33.htm, December 2004.

[23] J. Commission. Preventing patient-controlled analgesia overdose. *Joint Commission Perspectives on Patient Safety*, page 11, October 2005.

[24] A. H. A. S. Committee and S. S. Subcommittee. Heart disease and stroke statistics 2007 update. *Circulation*, 115(5), February 2007.

[25] J. D. Day and H. Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71(12):1334–1340, Dec 1983.

[26] L. Feng, A. L. King, S. Chen, A. Ayoub, J. Park, N. Bezzo, O. Sokolsky, and I. Lee. A safety argument strategy for PCA closed-loop systems: A preliminary proposal. In *5th Workshop on Medical Cyber-Physical Systems, MCPS 2014, Berlin, Germany, April 14, 2014*, pages 94–99, 2014.

[27] J. M. Goldman, M. Jaffe, D. Osborn, and S. Weininger. The integrated clinical environment (ICE) standard (ASTM f2761-09) - the first ten years. In *Poster Presentation at the Society for Technology in Anesthesia Annual Meeting*, 2014.

[28] J. M. Goldman, S. F. Whitehead, and S. Weininger. Eliciting clinical requirements for the medical device plug-and-play (MD PnP) interoperability program. In *Anesthesia & Analgesia: Abstracts of Posters Presented at the International Anesthesia Research Society 80th Clinical and Scientific Congress*, March 2006.

[29] M. Grissinger. Misprogram a PCA pump? it's easy! *P&T*, 33(10):567–568, October 2008.

[30] O. M. Group. Data distribution service (DDS). http://portals.omg.org/dds/, March 2014.

[31] C. S. Hankin, J. Schein, J. A. Clark, and S. Panchal. Adverse events involving intravenous patient-controlled analgesia. *American Journal of Health-System Pharmacy*, 64:1492 – 1499, July 2007.

[32] R. W. Hicks, V. Sikirica, W. Nelson, J. R. Schein, and D. D. Cousins. Medication errors involving patient-controlled analgesia. *American Journal of Health-System Pharmacy*, 65(5):429–440, March 2008.

[33] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[34] R. M. Hofmann. Modeling medical devices for plug-and-play interoperability. Master's thesis, Massachusetts Institute of Technology, June 2007.

[35] ISO. ISO 14971 medical devices - application of risk management to medical devices. page 82, 2007.

[36] ISO/IEEE. ISO/IEEE 11073-10101:2004 health informatics – point-of-care medical device communication – part 10101: Nomenclature. 2004.

[37] R. Ivanov, H. Nguyen, J. Weimer, O. Sokolsky, and I. Lee. Openice-lite: Towards a connectivity platform for the internet of medical things. In *21st IEEE International Symposium on Real-Time Distributed Computing, ISORC 2018, Singapore, Singapore, May 29-31, 2018*, pages 103–106, 2018.

[38] R. Ivanov, J. Weimer, and I. Lee. Context-aware detection in medical cyber-physical systems. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-*

Physical Systems, ICCPS 2018, Porto, Portugal, April 11-13, 2018, pages 232–241, 2018.

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

[40] M. Kasparick, M. Rockstroh, S. Schlichting, F. Golatowski, and D. Timmermann. Mechanism for safe remote activation of networked surgical and poc devices using dynamic assignable controls. In *38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2016, Orlando, FL, USA, August 16-20, 2016*, pages 2390–2394, 2016.

[41] M. Kasparick, S. Schlichting, F. Golatowski, and D. Timmermann. New ieee 11073 standards for interoperable, networked point-of-care medical devices. volume 2015, 08 2015.

[42] M. Kasparick, M. Schmitz, B. Andersen, M. Rockstroh, S. Franke, S. Schlichting, F. Golatowski, and D. Timmermann. Or.net: A service-oriented architecture for safe and dynamic medical device interoperability. *Biomedical Engineering / Biomedizinische Technik*, 01 2017.

[43] M. Kasparick, M. Schmitz, F. Golatowski, and D. Timmermann. Dynamic remote control through service orchestration of point-of-care and surgical devices based on ieee 11073 sdc. In *IEEE-NIH 2016 Special Topics Conference on Healthcare Innovations and Point-of-Care Technologies, Cancun, Mexico.* IEEE, IEEE, 2016/11 2016.

[44] Y. J. Kim, S. Procter, J. Hatcliff, V. Ranganath, and Robby. Ecosphere principles for medical application platforms. In *2015 International Conference on Healthcare Informatics, ICHI 2015, Dallas, TX, USA, October 21-23, 2015*, pages 193–198, 2015.

[45] A. King, D. Arney, I. Lee, O. Sokolsky, J. Hatcliff, and S. Procter. Prototyping closed loop physiologic control with the medical device coordination framework. In *2nd Workshop on Software Engineering in Health Care SEHC 2010*, May 2010.

[46] A. King, A. Roederer, D. Arney, S. Chen, M. Fortino-Mullen, A. Giannareas, C. W. H. III, V. Kern, N. Stevens, J. Tannen, A. V. Trevino, S. Park, O. Sokolsky, and I. Lee. Gsa: A framework for rapid prototyping of smart alarm systems. In *Proceedings of the 1st ACM International Health Informatics Symposium (IHI '10)*, November 2010.

[47] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell, 1997.

[48] B. R. Larson, J. Hatcliff, S. Procter, and P. Chalin. Requirements specification for apps in medical application platforms. In *Proceedings of the 4th International Workshop on Software Engineering in Health Care, SEHC 2012, Zurich, Switzerland, June 4-5, 2012*, pages 26–32, 2012.

[49] B. R. Larson, Y. Zhang, S. C. Barrett, J. Hatcliff, and P. L. Jones. Enabling safe interoperation by medical device virtual integration. *IEEE Design & Test*, 32(5):74–88, 2015.

[50] L. L. Leape. Reporting of adverse events. *New England Journal of Medicine*, 347(20):1633–8, November 2002.

[51] I. Lee and O. Sokolsky. Medical cyber physical systems. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 743–748, New York, NY, USA, 2010. ACM.

[52] K. Li, S. Warren, and J. Hatcliff. Component-based app design for platform-oriented devices in a medical device coordination framework. In *ACM International Health Informatics Symposium, IHI '12, Miami, FL, USA, January 28-30, 2012*, pages 343–352, 2012.

[53] L. Lin, R. Isla, K. Doniz, H. Harkness, K. Vincente, and D. Doyle. Applying human factors to the design of medical equipment: patient-controlled analgesia. *Journal of Clinical Monitoring and Computing*, 14:253–63, 1998.

[54] L. Lin, K. Vincente, and D. Doyle. Patient safety, potential adverse drug events, and medical device design: a human factors engineering approach. *Journal of Biomedical Informatics*, 34:274–84, 2001.

[55] A. S. Lofsky. Turn Your Alarms On. *APSF Newsletter*, 19(4):43, 2004.

[56] P. E. Macintyre. Safety and efficacy of patient-controlled analgesia. *British Journal of Anaesthesia*, 87(1):36–46, 2001.

[57] P. Masci, A. Ayoub, P. Curzon, M. D. Harrison, I. Lee, and H. W. Thimbleby. Verification of interactive software for medical devices: PCA infusion pumps and FDA regulation as an example. In *ACM SIGCHI Symposium on Engineering Interactive*

Computing Systems, EICS'13, London, United Kingdom - June 24 - 27, 2013, pages 81–90, 2013.

[58] P. Masci, A. Ayoub, P. Curzon, I. Lee, O. Sokolsky, and H. W. Thimbleby. Model-based development of the generic PCA infusion pump user interface prototype in PVS. In *Computer Safety, Reliability, and Security - 32nd International Conference, SAFECOMP 2013, Toulouse, France, September 24-27, 2013. Proceedings*, pages 228–240, 2013.

[59] MD PnP Prototype Regulatory Submission Working Group. PRS level 1 hazard analysis. Accessed from MD PnP Basecamp Web Site, November 2010.

[60] R. Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[61] M. Mullen-Fontino, N. O'Brien, and M. Jones. Critical care of a patient after CABG surgery. *Nursing Critical Care*, 4(4):46 – 53, July 2009.

[62] M. Mullen-Fortino and N. O'Brien. Caring for a patient after coronary artery bypass graft surgery. *Nursing*, 38(3):46–52, March 2008.

[63] A. Murugesan, M. P. E. Heimdahl, M. W. Whalen, S. Rayadurgam, J. Komp, L. Duan, B. Kim, O. Sokolsky, and I. Lee. From requirements to code: Model based development of a medical cyber physical system. In *Software Engineering in Health Care - 4th International Symposium, FHIES 2014, and 6th International Workshop, SEHC 2014, Washington, DC, USA, July 17-18, 2014, Revised Selected Papers*, pages 96–112, 2014.

[64] H. Nguyen, B. Acharya, R. Ivanov, A. Haeberlen, L. T. X. Phan, O. Sokolsky, J. Walker, J. Weimer, W. H. III, and I. Lee. Cloud-based secure logger for medical devices. In *Proceedings of the First IEEE International Conference on Connected Health: Applications, Systems and Engineering Technologies, CHASE 2016, Washington, DC, USA, June 27-29, 2016*, pages 89–94, 2016.

[65] M. Pajic, R. Mangharam, O. Sokolsky, D. Arney, J. M. Goldman, and I. Lee. Model-driven safety analysis of closed-loop medical systems. *IEEE Trans. Industrial Informatics*, 10(1):3–16, 2014.

[66] J. Paul, M. Sawhney, W. Beattie, and R. McLean. Critical incidents amongst 10033 acute pain patients. *Canadian Journal of Anesthesiology*, 51:A22, 2004.

[67] J. Plourde, D. Arney, and J. M. Goldman. Openice: An open, interoperable platform for medical cyber-physical systems. In *ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS, Berlin, Germany, April 14-17, 2014*, page 221, 2014.

[68] S. Procter and J. Hatcliff. An architecturally-integrated, systems-based hazard analysis for medical applications. In *Twelfth ACM/IEEE International Conference on Formal Methods and Models for Codesign, MEMOCODE 2014, Lausanne, Switzerland, October 19-21, 2014*, pages 124–133, 2014.

[69] M. I. Program. OpenICE software repository. http://mdpnp.org/MD_PnP_Program___OpenICE.html, March 2014.

[70] V. Ranganath, Y. J. Kim, J. Hatcliff, and Robby. Communication patterns for interconnecting and composing medical systems. In *37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBC 2015, Milan, Italy, August 25-29, 2015*, pages 1711–1716, 2015.

[71] R. Schrenker. Ensuring sufficient breadth in use case development: How should non-functional requirements be elicited and represented? In *Proceedings of the 2007 Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, HCMDSS-MDPNP '07, pages 135–136, Washington, DC, USA, 2007. IEEE Computer Society.

[72] O. Sokolsky, I. Lee, and M. Heimdahl. Challenges in the regulatory approval of medical cyber-physical systems. In *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, pages 227–232, New York, NY, USA, 2011. ACM.

[73] S. Syed, J. E. Paul, M. Hueftlein, and M. Kampf. Morphine overdose from error propagation on an acute pain service. *Canadian Journal of Anesthesiology*, 53(6):586–90, June 2006.

[74] U.S. Department of Health and Human Services Food and Drug Administration Center for Drug Evaluation and Research (CDER) Center for Biologics Evaluation and Research (CBER). Design control guidance for medical device manufacturers. 1997.

[75] U.S. Department of Health and Human Services Food and Drug Administration Center for Drug Evaluation and Research (CDER) Center for Biologics Evaluation and Research (CBER). Guidance for industry development and use of risk minimization action plans. Technical report, Office of Training and Communication Division of Drug Information, HFD-240 Center for Drug Evaluation and Research Food and Drug Administration, 2005.

[76] U.S. Department of Health and Human Services Food and Drug Administration Center for Drug Evaluation and Research (CDER) Center for Biologics Evaluation and Research (CBER). Guidance for industry: Q9 quality risk management. 2006.

[77] U.S. Department of Health and Human Services Food and Drug Administration Center for Drug Evaluation and Research (CDER) Center for Biologics Evaluation and Research (CBER). Design considerations and pre-market submission recommendations for interoperable medical devices. 2017.

[78] K. K. Venkatasubramanian, E. Y. Vasserman, V. Sfyrla, O. Sokolsky, and I. Lee. Requirement engineering for functional alarm system for interoperable medical devices. In *Computer Safety, Reliability, and Security - 34th International Conference, SAFECOMP 2015 Delft, The Netherlands, September 23-25, 2015. Proceedings*, pages 252–266, 2015.

[79] K. J. Vicente, K. Kada-Bekhaled, G. Hillel, A. Cassano, and B. A. Orser. Programming errors contribute to death from patient-controlled analgesia: case report and estimate of probability. *Canadian Journal of Anesthesiology*, 50(4):328–32, 2003.

[80] C. F. Wallroth, J. M. Goldman, J. Manigel, D. Osborn, T. Roellike, S. Weininger, and D. Westenskow. Development of a standard for physiologic closed loop controllers in medical devices. In *Poster Presentation at the World Congress of Anesthesiology*, 2008.

[81] P. Welch, N. Brown, J. Moores, K. Chalmers, and B. Sputh. Integrating and extending jcsp. volume 65, pages 349–370, 01 2007.