A Software Architecture for Medical Device Plug and Play

Maurice Rabb, Charlie Meyer, Ralph Johnson
University of Illinois – Urbana-Champaign
Computer Science Dept, MDPnP Project Group
201 N Goodwin, Urbana IL 61801, USA
{m3rabb, cmeyer2, rjohnson} @illinois.edu

Abstract

Ralph please write.

1. Introduction

This paper describes our work on a new architecture for medical devices.* The architecture enables any device that conforms to our interoperability protocol to be plugged into an integrated clinical environment (ICE) [1], and communicate with other medical devices and other clinical computer systems [2]. The architecture directly addresses the challenging and competing requirements of flexibility and safety. The architecture must be flexible enough to enable a wide variety of devices to be connected on the fly. New devices are always being developed, and the behavior of existing devices can change as new features are added. But an architecture for medical devices must also ensure safety, and it cannot rely on tests or proofs of a static configuration. Our architecture is designed to meet safety requirements in a rapidly changing environment.

2. Motivation

Medical devices such as vital-sign monitors and infusion pumps have become ubiquitous in emergency and operating rooms, patient hospital rooms, and other modern clinical environments. It is common for patients to have an array of devices connected to them in the course of their stay at a hospital. Setting up and monitoring a single device can be challenging; managing more than one device can quickly become overwhelming. So, mistakes are common, and some are lethal.

The goal of our architecture is to enable the clinical staff to change the configuration of devices connected to a patient, as well while maintaining safety requirements. When the system cannot ensure a safety requirement, it should alert the clinical staff so that they can correct the problem or manage the requirement manually.

For more than a decade, the consumer computing world has enjoyed the convenience of plug-and-play (PnP) connectivity between devices. Communications technologies such as USB [3], Bluetooth [4], and WiFi [5] make it easy to connect multiple devices such as computers, printers, storage drives, and cameras over a common network. The standards are resilient enough to enable new types of devices to be added seamlessly. For example, the ubiquitous USB keys are a recent innovation.

The FDA currently certifies the safety of all individual medical devices [6]. However, there are no safety standard for interoperability between devices. Such a standard is

* Supported in part by NSF 0720482, by NSF 0834709 and by ONR N000140810896

needed since medical devices are so frequently used in concert with each other. That being said, direct interoperability between devices is uncommon in the medical world. When it does occur, it is usually via proprietary protocols between devices of the same manufacturer.

The vision of the Medical Device Plug-and-Play (MDPnP) project [7] is an open standard protocol that enables a wide variety of devices to be able to dynamically connect themselves to a system's network [8]. Not only would such a system be more convenient, and safer than the current approach, but it also has tremendous potential to provide improved and more reliable patient healthcare. Every year medical accidents occur that could possibly have been prevented if the devices themselves were connected and "smart" enough to communicate their own state, and the patient's condition with each other (and the clinical staff), to affect each other's behavior. [9] For example, each year dozens of patients overdose or are otherwise injured while using patient controlled analgesia (PCA) machines. Our motivating use cases come from such scenarios.

3. MDPnP architecture overview

Our experimental MDPnP software architecture is designed to simultaneously satisfy the challenging and competing requirements of *flexibility* and *safety*. The architecture encompasses medical devices providing treatment and monitoring, as well as a controller. A controller is a computer that is responsible for managing its ICE.

3.1. Integrated clinical environment

Each ICE represents a single patient and all of the medical devices used on that patient. While there may be more than one patient in an location (e.g. a hospital room), each patient has an individual ICE.

An ICE's controller keeps track of all of the devices that have been connected to it; maintains a model of the patient; references its own database of procedural and device rules aka rule system; mediates communication [10] between the devices and external systems such as the clinical staff notification, and patient electronic medical records systems; keeps a record of the changing state of the devices, all events, and all commands issued [10]; and lastly, provides a GUI for clinicians to monitor the system, add and update rules, and otherwise control the system. The relationships of the ICE entities are shown in figure 1.

The controller uses incoming vital sign messages to update its model of the patient [11]. Using the rules along with the devices' and patient's current state, the controller sends command messages to the treatment devices. When a device or patient enters an abnormal state, the controller sends warning or alarm messages to the clinical staff via the notification system. For the sake of simplicity, we assume a "wired" system where devices are physically connected and disconnected using USB.

3.2. Architecture requirements

A medical PnP system should be generic enough to support new devices without requiring a reengineering of the system architecture. New devices might require new rules, new medical procedures, or raise new hazards, but this should not require a change to the basic system.

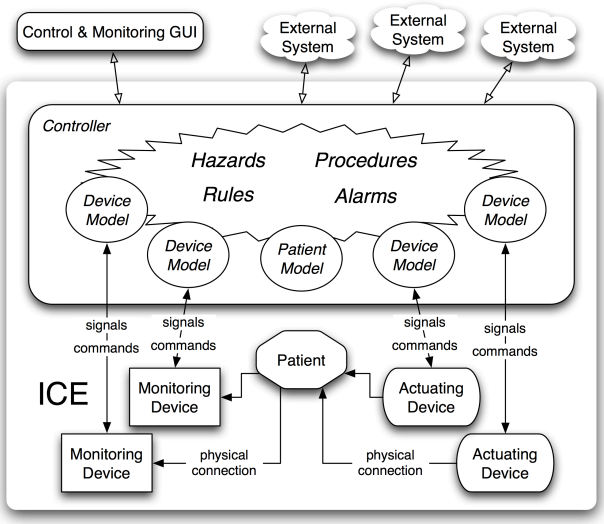

Figure 1. Architecture overview

A medical PnP system differs from consumer PnP systems, in that safety is much more important; with such devices human lives are at stake. Safety in medical devices has many aspects. One is to monitor for various hazardous conditions and to raise alarms when they occur. . This can be easily done with rules. Alarms are so common in most hospitals that medical personnel start to ignore them. A rule-based approach to monitoring can filter out useless alarms and let people focus on the ones

that remain.

A second aspect of safety is being able to ensure that some things never happen. For example, a laser should not be turned on when oxygen is being administered, and morphine should not be administered to a patient whose respiration and pulse are low. This requires some sort of interlock. To make things more difficult, this interlock must work even if the network or controller fails. A safety property is *open-loop* safe [12] if it can be ensured even if the controller and/or network fail. It is *closed-loop* safe [12] if it can be ensured as long as the controller and network do not fail.

The ICE's rule-based approach enables safety properties to be added dynamically. Our approach is flexible enough to enable clinicians to dynamically add rules for new medical procedures and devices. However, every time a new rule or device is added to the system, the system must check to ensure that that addition honors the safety requirements. It will raise an alarm if it can no longer guarantee a safety property or if an open-loop safety property becomes only closed-loop safe.

4. Use cases

We used two medical use cases as a basis for developing our architecture and system: 1) patient controled pain management, and 2) glucose management. Both of these use cases were derived, respectively, from the original "Safety Interlock" and "Physiological Closed Loop Control" (PCLC) use cases from the MDPnP reference proposal [1]. Our first step was to transform the medical prose into software engineering use cases [13], each of which identified the preconditions and post-conditions (including success, alarm and failure conditions), as well as the key operational, hazard and failure scenarios. After accomplishing this, we were able to compose the rules and required state information for each use case.

4.1. Patient controlled pain management

A Patient Controlled Analgesia (PCA) machine allows patients to manage their pain while staying in the hospital. The machine is used to administer a patient's prescription for an intravenous delivery of a drug such as morphine. The machine provides a continuous delivery of the drug, and also, by pushing a button, the patient is able to temporarily augment the drug delivery with an additional *bolus* of medication.

Historically, the conventional wisdom was that the machines were inherently fail-safe from a patient self-overdosing, because the patient would pass out before they could injure themselves. Unfortunately, this assumption has proven to be false.

The level of medication that a patient needs for pain relief, as well as the level at which it may injure them are always in flux. However, by monitoring a patient's respiration rate and blood oxygenation (SpO2) level, it is easy to check their status and detect when they are receiving too much medication.

The MDPnP Safety Interlock use case entails designing a system where a PCA machine is automatically controlled depending upon the input of a respiration monitor and SpO2 monitor. If either the respiration rate or SpO2 level fall below certain (patient specific) values, the PCA machine is turned off and the clinical staff is alerted. Furthermore, the number and frequency of button presses by the patient can help in recalculating the drug delivery rate, or in alerting the clinical staff that the prescription needs to be adjusted upward or downward. Finally, if there is some critical equipment failure, the PCA machine must be turned off.

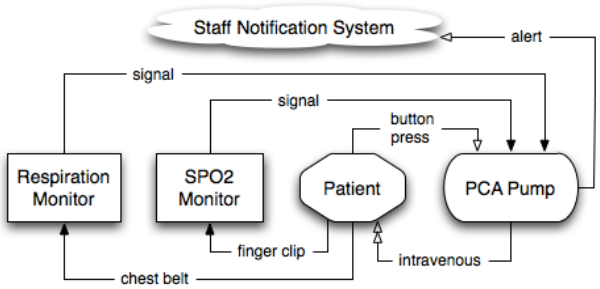

Figure 2. PCA abstract requirements model

These requirements are modeled in figure 2. The devices and patient are represented by the boxes. The arrows below the boxes represent the physical connections between the entities. The arrows above the boxes represent the input signals to and from the pump. Double arrows represent the flow of drugs. Black arrows represent the flow of safety critical information, whereas white arrows represent important yet non-safety critical relationships.

4.2. Glucose management

In the clinical environment, diabetic, sepsistic, post-surgical and otherwise critically ill patients frequently must have their glucose levels managed. This can be accomplished by connecting a patient intravenously to both insulin and a supplemental glucose supply via two separate infusion pumps. Unfortunately, managing these two medications is complex, because beyond the underlying reason(s) for the patient's abnormal glucose levels (e.g. diabetes), there are a wide-range of patient responses to glucose and insulin, as well as a natural daily variance of a patient's glucose level (due in part to the eating and sleeping cycles), not to mention a number of types of insulin drugs. While diabetes is a debilitating condition where one's glucose level remains chronically high, there is usually little risk in having one's sugar level temporarily elevated. Conversely, having one's glucose fall below a certain level, even for a short period can cause great harm, and can be potentially life threatening. Treatment is as much an art as a science, and requires a nuanced approach [14].

As with the PCA use case, integrated monitoring feedback can assist with glucose management. The MDPnP PCLC use case entails designing a system where the rates of the insulin and glucose infusion pumps are automatically controlled depending upon the measurements of a continuous or intermittent glucose monitoring device. If, in spite of the dynamic adjustments, the patient's glucose level still rises or drops too much, or simply fails to respond to the medications, the clinical staff is alerted. These requirements are modeled in figure 3.

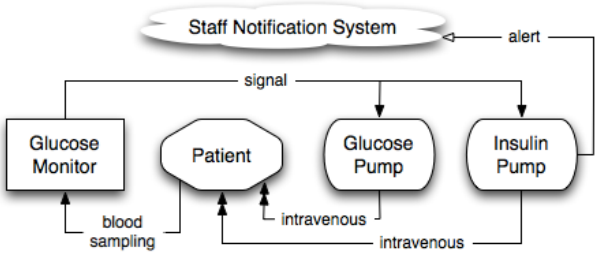

Figure 3. Glucose management abstract requirements model

Unlike the PCA, in this use case, there is no fail-safety state. In the case of a patient hazard or equipment failure, there is no definitive choice on whether or not to turn off either infusion pump.

5. Architecture specifics

The architecture for our system was designed with several requirements influencing the final implementation. Flexibility was of primary importance. The system needs to be able to handle a variety of devices, each with differing operational constraints and safety requirements. The architecture also needs to have a common communications protocol so that each of the devices in the system can reliably communicate with the central controller and each other. Modularity was also taken into consideration when designing our architecture, so that no one component of the system is responsible for more than it was designed for and so that each component can be easily tested. To this end, testing was also a requirement in our design. Each component of the system needs to be able to be tested both individually and in concert with the other components of the system.

5.1. Controller model

The controller is the central component of our architecture. All devices attach themselves to it and communicate with each other through it as well. The controller does not know the exact implementation of each device, but rather only the interface that each specific device exposes .

To attach devices to the controller, we have a set of device controllers, with one device controller responsible for each specific type of device that the system accepts. The device controllers monitor for new hardware attached to the network and when a new device is detected, they instantiate a software model for that specific device and attach it to the main controller. To add new types of hardware to our architecture, only a new device controller and software model need to be written for that device, since all other components of our architecture are agnostic of the implementation of each device.

The controller is also responsible for maintaining the patient model and providing indirect access to it from each device in the network. By exposing a limited interface to the patient, each device need only know about the functionality that the controller provides rather than the internal implementation of the patient model.

5.2. Device model

The primary goal of our architecture was to allow a variety of devices to interact with each other and the patient model regardless of their internal implementation, operating constraints, or safety levels. For each hardware device that exists in the network, there exists a software model that exposes its functionality to the system. Devices are represented as state machines with a set of states S and operations O. Operations are supplied parameters, and when an operation $o \in O$ is applied to the current state of a device $s \in S$, a transition is performed changing the current state of the device to $s' \in S$. All devices also have a set of error states $E \subseteq S$. When a device is in an error state $e \in E$, it actively notifies the network of the error, which usually will trigger a rule to be evaluated and further action to be taken either by the network or the clinical staff.

All devices in our architecture share a limited set of common functionality that is initially known to the architecture before any device is added to the network, such as retrieving the device's name, type, and extended functionality. Furthermore, our architecture classifies devices as either actuators or sensors. When an operation is applied to an actuator, the device physically alters the patient that it is attached to whereas sensors passively monitor the patient and send signals back to the network containing the data collected.

Each software model of each device necessarily contains the required logic to manipulate its hardware counterpart. These implementation specific details are abstracted away such that the controller and every other device in the network only views its interface, which consists of its operations and parameters that they require and the signals that it sends back to the network. This implementation independence allows us to be able to support any number of any type of devices with minimal modification to our architecture.

Currently, our architecture supports several hardware devices, including pulse oximeters, infusion pumps, and blood glucose meters to allow us to model several use cases. Our architecture was designed such that adding a new type or model of hardware only requires writing a new software model that conforms to the basic device specifications that our architecture requires.

5.3. Patient model

An ICE corresponds to all of the devices connected to a single patient in a clinical setting. An ICE models the connected patient as a repository of information. The patient model contains basic information about the connected patient such as name, age, sex, and date of birth along with any information generated by signals sent from devices that relate to the patient. The model is the only component of our architecture whose interface and implementation are explicitly known to the controller when the network is instantiated. This information is used by the controller to flow information from devices to the model and by the rules engine to retrieve information and take action on it. An instance of a patient model is the only prerequisite for bringing up the network and allowing devices to be attached to it. Each payload that is delivered to the patient model from a device conforms to the requirements of our architecture, which demands that it includes information such as the device that generated the data, the sampling time of the data, and its origin. Since all data stored in the patient conforms to these requirements, the rules can be written to dynamically modify the network.

5.4 Events

The event subsystem is a critical component of the architecture because it is what powers the rules framework and the user interface to be interactive. A set of event identifiers is stored in a central repository. Components in the network can attach handlers to each of the event identifiers known to the system. When an event is fired, the subsystem accesses which handlers are attached to the identifier of the fired event and safely executes them asynchronously, passing each of the parameters attached to the fired event to each of the handlers. This is done to ensure reliability and safety of the network. If one handler locks up or causes an error to be thrown, it does not cause an impact on the guarantee that any other handler does not begin execution within a reasonable amount of time from when the event was fired. Events can be fired due to a new device being added or removed from the network, input from a hardware device, input from a user, or from a device entering an error state. Any errors that handlers might throw are captured, logged, and cause another event to fire representing the error.

5.5 Rules

Rules govern how the system behaves when different events are fired. Rules are implemented as dynamic objects that can be evaluated. Each rule has a set of binary conditions and an action. When all the conditions are true, the action is executed. Rules are attached as handlers to each event that they mention, and when an event is fired, all rules that are attached to it are evaluated. This ensures that rules are evaluated whenever their conditions are true.

Though there has been substantial prior work on rule engines and rule systems [15, 16], for our simple proof-of-concept architecture, our rudimentary approach was adequate. However, as the complexity of our work increases we will consider adopting a more advanced existing rule system.

5.6 User interface

We built a simple user interface to wrap around our network using the Yahoo User Interface (YUI) library [17]. The interface is dynamically generated when it is loaded to accurately reflect the current state of the network. The interface is broken down into several main components: controller, rules, devices, and patient.

The controller component allows the user to initialize the network and once initialized, displays the current state of the controller. In the rules component, there is a wizard-like interface for creating, editing, and removing rules from the network, as shown in figure 5. Since our implementation of rules requires functions to be passed in to the constructor of each rule when they are created, the interface provides a set of predefined functions that can be executed on successful evaluation, failed evaluation, and on evaluation of each binary condition. Once rules are constructed, they can be edited, deleted, attached to the network, and detached from the network, as shown in figure 6. When attaching rules from the network, the interface queries the events system, enumerates all possible events, and displays the events to the user so that the user can pick which event the rule should be evaluated on.

The devices component of the interface contains a tab for each of the devices that are attached to the network and offers information about and control of each device.

Lastly, the patient component has several modules. The first module displays basic information about the patient such as name, date of birth, and sex. There is then one module for each type of vital sign that the patient has stored, such as blood oxygenation and heart rate. Each of the vital signs modules displays an interactive chart plotting the data as it streams in as well as detailed information about each reading in a table that also dynamically updates.

This proof of concept illustrates the features of our architecture and allows for a user with no knowledge of the inner workings of the software to have total control of the network.

6. PCA application of the architecture

We used our new architecture to implement both the PCA and glucose monitoring medical use cases. Working with our medical clinicians, we developed detailed descriptions of the pre and post conditions for these two medical procedures. With their help, we decomposed the initial use cases into detailed scenarios. From these scenarios we extracted the device and procedural rules. For the sake of brevity, of the two use cases, we only discuss our implementation of the PCA.

6.1. PCA pre and post conditions

Every medical procedure has a list of pre and post conditions. The pre-conditions for the PCA medical procedure largely consist of ensuring that the PCA pump, and SpO2 and respiration monitors are properly setup and connected. From the perspective of our architecture, the post-conditions are far more interesting.

The post conditions are shown in listing 1. They are classified as either: success, alarm or failure, and considered either critical or non-critical. The meaning of success and failure conditions is self-explanatory. Alarm conditions specifically identify the circumstances where the system must alert entities outside of the immediate device system, e.g. the clinical nursing staff.

Post Conditions
Success conditions
- ***Critical:*** The PCA pump administers medication at the proper dosage and frequency.
- ***Non-Critical:*** The patient is able to self administer pain relief as needed.

Alarm conditions
- ***Critical:*** The PCA pump fails.
- ***Critical:*** The patient's SpO2 or respiration rate are outside the defined acceptable range.
- ***Non-Critical:*** The patient or family members pushes delivery button too often.

Failure conditions

- ***Critical:*** The patient overdoses.
- ***Non-Critical:*** The patient doesn't receive adequate pain relief.
- ***Non-Critical:*** The nurse call system receives too many spurious alarms.

Listing 1. PCA pre & post conditions

6.2. PCA scenarios

We divided the PCA scenarios into several subsets: operational, hazard and failure, shown in listing 2. Each scenario describes a specific circumstance in performing the PCA procedure using the ICE. The operational scenarios describe the standard expected circumstances that must be handled; i.e. the patient has stable vital signs, or the vital signs are falling out of the save range. The hazard scenarios describe less common or unexpected circumstances that must be handled; i.e. equipment disconnections or patient interference. Finally, the failure scenarios describe circumstances where the failures occur because of incorrect assumptions or calculations, or due to equipment failure.

Operational Scenarios

- Normal patient
- Patient at risk

Hazard Scenarios

- PCA Pump detects internal error
- ICE detects PCA pump disconnect
- PCA detects ICE disconnect
- Patient pushes the button for pain relief
- Patient goes for a walk
- Patient or family member pushes the delivery button too often

Failure Scenarios

- Patient is incorrectly considered at too low a risk
- Patient is incorrectly considered at too high a risk
- Algorithm improperly calculates the safety range too optimistically
- Algorithm improperly calculates the safety range too conservatively
- SpO2 and Respiration Rate detected incorrectly

Listing 2. PCA scenarios

6.3. PCA rules

Having identified the pre and post conditions and scenarios, we were ultimately able to express the PCA procedure using 13 rules, shown in listing 3. (We may yet add more rules as we continue to develop the PCA use case.)

R1. Pump ready
R2. Start pump
R3. Normal pumping operation
R4. Vital signs out of range
R5. Pump failure
R6. Pump low battery warning
R7. Pump near empty warning
R8. Pump empty alarm
R9. Patient pushes delivery button
R10. Patient pushes delivery button too many times
R11. Pump out of contact
R12. Pump Can't "hear" any instructions
R13. Re-evaluate patient risk assessment

Listing 3. PCA rules

Each rule is composed of a list of conditions, and list of success or failure actions respectively. If all of the conditions are true then the success actions are executed, otherwise the failure actions are executed instead. The two rules that cover the standard operation scenarios are representative examples, and are shown in listing 4.

R3: Normal pumping operation

Conditions:

- The system status is running.
- The current oxygen saturations is within the normal range.
- The respiration rate is within the normal range.

Actions:

- Enable the PCA pump to pump X many doses over Y minutes.

R4: Vital signs out of range

Conditions:

- The current SpO2 or respiration rate is outside the defined acceptable range.

Actions:

- Turn off the pump.
- Send an alert to the call system.

Listing 4. Expanded PCA rules for standard operation

Most of the remaining rules are quite simple. For example: ***R7****: Pump near empty warning* has only one condition: "*The pump sends a near-empty signal*"; and only one action, "*Alert the nurse calling system*".

6.4. PCA patient and device models

We then specify the attributes of the patient model and the signals and commands of the devices. The patient model is assigned attributes to keep track of the patient's respiration rate and SpO2 level vital signs, as well as the patient's identification information.

The respiration and SpO2 monitor models handle the vital sign signals and error events produced by their respective devices. The two monitoring device models log their devices' current vital signs, and represent their devices' current operational and/or error status.

The pump device model handles communications with the pump. It handles the pump's status and error messages, representing its current operational and/or error status. It logs the pump's drug volume, battery level, and all patient button drug delivery requests. Finally, it responds to messages from the controller, relaying commands to the pump; most specifically, commanding the pump to deliver a bolus of drug to the patient.

Additionally, each of the device models determines when communication to its respective devices has timed out.

6.5. PCA ICE configuration

To run the PCA medical procedure on a patient, we first build the rules using the GUI, and then connect the PCA pump and monitoring devices to the ICE. As the rules are generated, they are added to a set maintained by the controller. As the devices are connected to the ICE, the controller instantiates a model for each of them, as well as a patient model. While running, the controller manages signals from the devices; constantly iterating through and evaluating the rules; and as necessary, sending commands to the PCA pump.

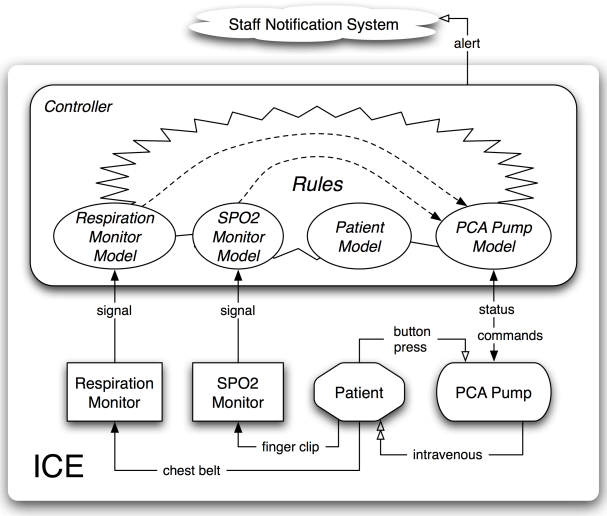

Figure 4. ICE configured for PCA

The ICE PCA configuration is shown in figure 4. The controller mediates the signals from the monitors and pump. The dashed lines represent the fulfillment of the signals paths shown in figure 2. These two safety critical signals are now effected *indirectly* via the rules. Since the bolus request button is connected directly to the pump, the controller is not able to directly mediate the button signal. Instead, the controller learns about this signal indirectly through is polling of the PCA pump's status.

For our PCA proof-of-concept demo, we used an actual infusion pump and SpO2 monitor, and mocked the respiration monitor. Our demo was capable of working with either the SpO2 monitor connected to an actual person, or by simply mocking a patient SpO2 data-stream. Though our pump was fully operational, it was never actually connected to a person.

7. Future work

Currently, none of the pumps or monitoring devices provide direct support for our architecture. Therefore, we were forced to wrap the actual medical devices to simulate the interface for our architecture. For simplicity, we used a single computer to act as the wrapper, communicating with each device, and also to act as the ICE controller. Ideally, we would have used separate computers to act as controllers and wrap each device. Not only would such a system be more realistic, but it would be more reliable because upon initialization, the necessary rules would be distributed to each machine, ensuring that safety was

preserved even if the ICE controller were to fail. Though our design fundamentally supports this distributed approach, we have yet to implement it.

Our system uses rule for several purposes, *foremost*, to implement the safety properties for a specific configuration of devices. It uses rules to implement the effective properties, e.g. in the PCA use case, enables the patient to push the PCA button to delivery additional pain relief. While this is important for the expected behavior of the PCA, its fulfillment has nothing directly to do with the safety of the system. The system also uses rules to gather data from the devices and build a patient model. These rules are essentially defining the value of variables. They might be used to state a safety property, but they are not themselves safety properties.

In our simply system, there are no automatic checks to ensure that the rules satisfy the safety requirements of a configuration. As device and rules were added, it would be useful if the system could calculate the dependency graph dynamically, and through a GUI display any safety conflicts or hazards as they arise. We recognize that there has already been substantial research into the design and use of rule systems. Additional, we are looking into leveraging an existing and more sophisticated rule system instead.

Our system doesn't yet have a graceful way to end the use of a particular configuration. While our system checks for device disconnections, it does have a specific protocol for definitively detecting the end of a procedure. For example, in the clinical environment, when they are done with a suite of monitors and pumps, they just disconnect them and put them away. There is no official step that says "all done".

While we are pleased with our model for the PCA use case, in following up with doctors, we realize that our glucose monitoring model needs to be improved. Our solution still does not address the realities of the needs for glucose management for the specific medical reasons that a patient's glucose is being managed.

8. Conclusion

Ralph please write.

9. Acknowledgement

We would like to thank Steven Moser, Yun Young Lee, Nicholas Chen, Audrey Petty, and the MDPnP project group – with special thanks to Mu Sun, Cheolgi Kim, and Mary Flesner.

References

1. Goldman, J. M., "Medical Devices and Medical Systems—Essential safety requirements for equipment comprising the patient centric integrated clinical environment (ICE) — Part 1," draft ASTM TC F29.21 N 21, Sep. 2008. http://mdpnp.org/uploads/ICE_Part_I_draft_21Dec2008_N30_web.pdf
2. Mu Sun, Qixin Wang, Lui Sha. Building Reliable MD PnP Systems. Joint Workshop On High Confidence Medical Devices, Software, and Systems (HCMDSS) and Medical Device Plug-and-Play (MD PnP) Interoperability. June, 2007.
3. Universal Serial Bus. http://www.usb.org/
4. Bluetooth Sig. http://www.bluetooth.com/
5. Wi-Fi Alliance. http://www.wi-fi.org/
6. Federal Food, Drug, and Cosmetic Act, Title 21 United States Code [321] (h). http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm?CFRPart=860
7. Medical device "plug-and-play" interoperability program. http://mdpnp.org/, 2008.
8. C. Kim and et. al, "A Framework for Wireless Integration in Interoperable Real-Time Medical Systems," Tech. Rep. http://agora.cs.illinois.edu/display/mdpnp, 2009.
9. K. Grifantini. "plug and play" hospitals: Medical devices that exchange data could make hospitals safer. MIT Technology Review, July 9, 2008.
10. Gamma, Erich; Richard Helm, Ralph Johnson, and John Vlissides (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN 0-201-63361-2.
11. Fowler, Martin (1997). Analysis Patterns: Reusable Object Models. Addison-Wesley. ISBN 978-0201895421.
12. Storey, Neil (1996). Safety Critical Computer Systems. Addison-Wesley. ISBN 978-0201427875.
13. Cockburn, Alistair (2000). Writing Effective Use Cases. Addison-Wesley. ISBN 978-0201702255.
14. Interview with Evan Lyon MD. Partners in Health. October 15, 2009.
15. Hayes-Roth, F., Rule-based systems. Commun. ACM, 1985. 28(9): p. 921-932.
16. Zacharias, V., Development and Verification of Rule Based Systems -- A Survey of Developers, in Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web. 2008, Springer-Verlag: Orlando, Florida. p. 6-16.
17. Yahoo! User Interface Library. http://developer.yahoo.com/yui/

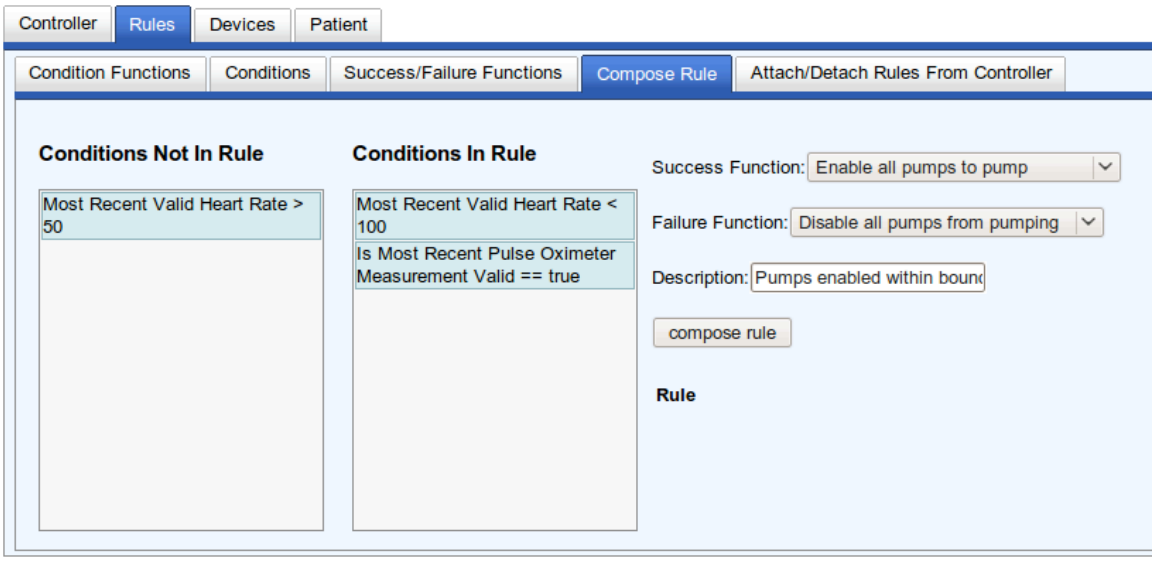

Figure 5. Rules wizard

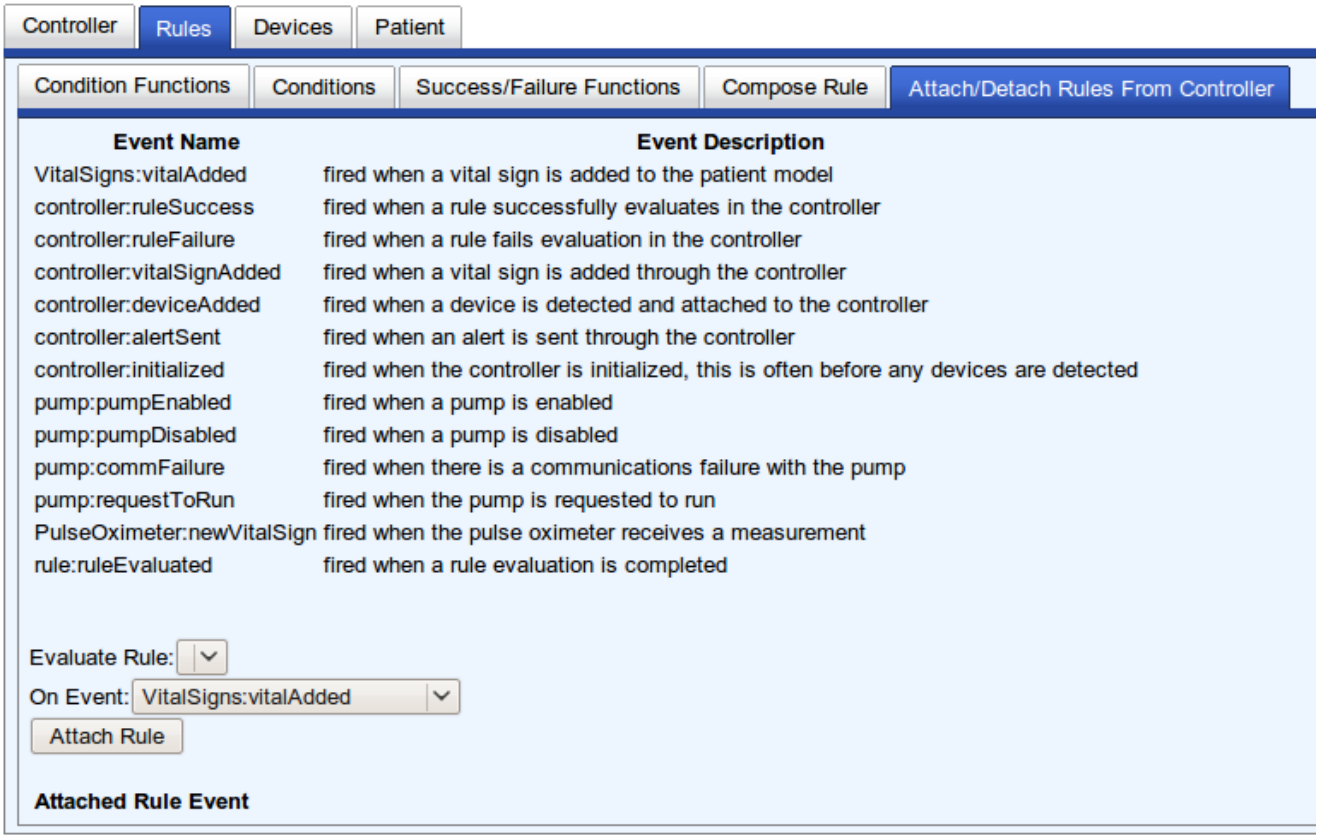

Figure 6. Rules editor