Formal Methods Based Development of a PCA Infusion Pump Reference Model: Generic Infusion Pump (GIP) Project

David Arney [*] Raoul Jetley [†] Paul Jones [‡]
Insup Lee [**] Oleg Sokolsky [††]

[*] University of Pennsylvania, arney@cis.upenn.edu
[†] FDA OSEL,
[‡] FDA OSEL,
[**] University of Pennsylvania, lee@cis.upenn.edu
[††] University of Pennsylvania, sokolsky@cis.upenn.edu

Postprint version. Presented at *Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, June 2007.

Formal Methods Based Development of a PCA Infusion Pump Reference Model: Generic Infusion Pump (GIP) Project

David Arney
University of Pennsylvania
arney@cis.upenn.edu

Raoul Jetley
FDA OSEL
raoul.jetley@fda.hhs.gov

Paul Jones
FDA OSEL
PaulL.Jones@fda.hhs.gov

Insup Lee
University of Pennsylvania
lee@cis.upenn.edu

Oleg Sokolsky
University of Pennsylvania
sokolsky@cis.upenn.edu

Abstract

As software becomes ever more ubiquitous and complex in medical devices, it becomes increasingly important to assure that it performs safely and effectively. The critical nature of medical devices necessitates that the software used therein be reliable and free of errors. It becomes imperative, therefore, to have a conformance review process in place to ascertain the correctness of the software and to ensure that it meets all requirements and standards.

Formal methods have long been suggested as a means to design and develop medical device software. However, most manufacturers shy from using these techniques, citing them as too complex and time consuming. As a result, (potentially life-threatening) errors are often not discovered until a device is already on the market.

In this paper we present a reference model based approach to software conformance checking. Reference models enable the application of formal methods to software conformance checking, and provide a framework for rigorous testing. To illustrate the approach, we develop the reference model for a Generic Patient Controlled Analgesic Infusion Pump, and explain how it can be used to aid software conformance checking in a regulatory environment.

1 Introduction

Software has been used in medical devices for several decades now. During this time it has grown in complexity and society has come to rely heavily on it. Because of the often safety-critical nature of medical devices, even a small software error can have undesired consequences that can result in serious injury or even death.

There are currently no widely accepted assurance techniques in use for development or verification of software in medical devices. Most software for these devices are developed using conventional (or ad hoc) strategies, which do not serve to rigorously assure system properties, such as safety.

Regulatory agencies, like the FDA's Center for Devices and Radiological Health (CDRH), assess device software to ensure that it conforms to standards for safety and reliability. Existing standards, however, concentrate only on quality system development processes, not the integrity of the software itself [9]. Such an approach, though not entirely without its benefits, depends mainly on crafted test case construction and the results of test case executions for validation, with little regard for properties of the actual code [8].

The use of formal methods based verification has often been suggested as an alternative to process-based conformance [15]. Formal methods have been used extensively in the analysis of real-time embedded systems [12], and in the automotive and aviation industry [19, 10]. However, unlike these applications, there are as of yet no established guidelines for formal methods based development and testing of medical devices.

To begin exploring this issue, we introduce in this paper, a formal methods based approach to specifying the design requirements for a patient controlled analgesia, or PCA, infusion pump.

2 Overview of the Model Based Process

The goal of our work is to develop a methodology to facilitate checking that medical device software is safe and reliable. Figure 1 shows the overall structure of the proposed methodology, which is based on formal models and consists of the following steps:

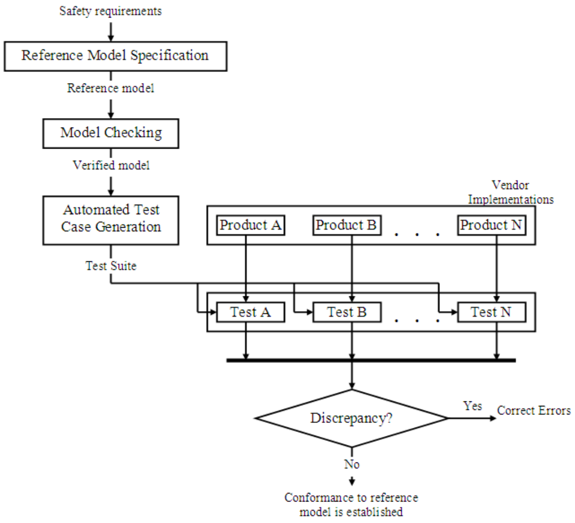

Figure 1. Methodology

- Capture functional and safety requirements of a specific type of device.
- Build a model which captures 'safety' and 'reliability' properties for a specific type of device.
- Use formal methods to verify that the model is correct with resepect to the requirements.
- Generate tests from the model and use the tests to check implementations.

This section describes each of these steps. In this paper, we present our infusion pump work as an example of applying our broader design and verification methodology. The next several sections describe a case study applying this methodology to Patient Controlled Analgesia (PCA) infusion pumps.

The model based conformance review process is explained in Algorithm 1. The scheme, as depicted in the Algorithm, involves deriving a test case suite from a formally verified reference model, essentially a one-time effort, and using this test suite to assess individual device software.

The process, as definied in the algorithm, consists of four separate activities or phases. The first phase deals primarily with establishing a reference model based on the requirements and delineating the system specifications used in the reference model. This reference model consists of a set of communicating state machines which capture the behaviors described in the requirements. To have the reference model serve as a benchmark for all software implementations, it is imperative to ensure that the reference model is sound and correctly incorporates the major functionalities of the device captured in the requirements. To ensure this, the requirements are collected in consultation with a set of experts (both in-house at the FDA, and from the industry) during Phase 0. With the help of these experts, common characteristics of the device are identified, and used to define the specifications for the reference model.

Algorithm 1 Premarket Conformance Review

```
Input: A software family $S$ characterized by system requirements $R$, a set of safety properties $P$ for $S$, a set of software implementations $I$ based on $R$.
Output: boolean array conform[I], indicating whether $i \in I$ conforms to safety properties given by $P$.
Phase 1: Define a reference model $M$ based on the requirements $R$.
Phase 2: Verify the reference model $M$ using formal verification techniques (i.e., model checking) to check whether $M \vDash P$.
if $M \nvDash P$ then
  Revise the reference model $M$. Go to Phase 1.
end if
Phase 3: Derive a suite of test case sequences $T_s$ from reference model $M$.
for all $i$ in $I$ do
  Validate $i$ against derived test suite $T_s$ (i.e., check whether $i \vDash T_s$).
  if $i \vDash T_s$ then
    conform[i] $\leftarrow$ true.
  else if $i \nvDash T_s$ then
    conform[i] $\leftarrow$ false.
  end if
end for
```

While it may not be possible to capture all functionalities and behavior for each and every PCA pump implementation in the PCA pump reference model, the reference model must consist of a set of minimum safety behaviors that all implementations of the device must possess. The behaviors of the reference model are shown to guarantee the minimum safety properties during Phase 2. The implementations, thus, will be evaluated against these behaviors during testing. A failed test case would indicate a violation of a safety condition and may lead to a potential hazardous malfunction.

In Phase 2, the reference model itself is defined using formal methods based specification techniques and is (usually) represented as an extended state machine or finite state automaton. The advantage of using a formal model is that it can be rigorously verified using model checking based on exhaustive state explorations to ensure its correctness and adherence to the safety requirements. In this phase, the correctness and adequacy of the reference model is evaluated with respect to safety properties. If needed, the reference model is revised and refined until it meets the requirements.

In phase 3, the reference model is verified and used to automatically derive test case sequences. Each test case generated represents an execution path in the reference model and corresponds to a set of user inputs or events for the system. All such distinct paths (test sequences) and their corresponding results are recorded and maintained as a test suite for products in the device family. As a manufacturer develops a implementation of a modeled device, it is checked against the test suite for conformance. Any discrepancy between the expected results (as obtained through the reference model) and observed outputs indicates a failure of compliance for the product. However, if the results of the tests are in agreement, then the product is deemed compliant with the established requirements. Such evidence can then be used when making a case for the device in a pre-market submission.

3 Generic PCA Pump Case Study

An infusion pump is a typical safety-critical medical device that uses software to control its underlying hardware and events. Not all infusion pumps are the same, however. There are a number of different classes of pumps providing varying treatment options to patients. Even within a particular class, pumps can be further categorized based on their use environment, delivery mechanism, intended function etc.

PCA infusion pumps are used to administer pain medication. The pumps are programmed with a basal rate, which is the background rate used most of the time, and can be triggered by the patient to deliver a bolus dose, which is a preset volume of drug delivered at a higher rate. The patient triggers a bolus dose by pressing a button when they feel a high level of pain and decide that they need more medication. The pump must be programmed with upper limits on the number and frequency of bolus doses which may be delivered in order to prevent overdoses.

PCA infusion pumps have been involved in numerous incidents resulting in injury or death. Thus, techniques which may improve their safety are of interest from a national health care system viewpoint. Improving the safety of these pumps involves work in numerous engineering disciplines including human factors, mechanical, electrical and software design. Our interest is primarily in the fourth area, software design and development. In this paper, we present our infusion pump work as an example of applying our broader design and verification methodology.

Our goal is to develop a set of requirements and a reference model of a generic PCA infusion pump. This reference model can be used to help verify the correct functioning of software in real world PCA pump implementations submitted to FDA for market approval. Using a recognized reference model could allow device manufacturers to concentrate on the specialized functionality of their particular pump devices and simplify the verification process.

3.1 Hazard Analysis Process.

A hazard analysis for a device is a systematic way of enumerating the hazards and their contributing factors. We compiled such a list for PCA infusion pumps in several stages. First we listed generic hazards common to all infusion pumps. Then we added the hazards for small volume pumps, and finally those associated specifically with PCA pumps.

We began by creating a system diagram, shown in Figure 2, and listed hazards for each element of this diagram. For instance, the infusion set has certain hazards associated with it, such as blockages (occlusions) and leaks. We also consulted publications on infusion pump hazards including ECRI's evaluation of PCA pumps [7] and risk management and safety standards such as ISO 14971 [14] and IEC 60601 [13].

3.2 Examples of Hazards and Common Mitigations.

This section contains several examples of hazards we gathered during the hazards analysis process. These hazards will be used as examples in the requirements and model analysis sections of this paper.

Output side occlusion. An output side occlusion is a blockage of the infusion set after the pump. This can be caused by kinked tubing, closed valves, clot formation, or numerous other problems. This condition is most commonly detected using a pressure sensor, but could also be detected with a flow sensor or by monitoring the pump motor's current draw.

Air bubble. Air bubbles can be introduced into the pump system in a variety of ways, for instance when the fluid reservoir is changed or damaged. Air bubbles delivered to the patient can result in serious injury or death, so the bubbles must be detected and the pump stopped quickly enough to prevent them reaching the patient. This is accomplished using an air sensor to detect bubbles. The sensor may measure the impedance of the fluid or use an optical sensor or some other method.

Misuse. In a home care environment, children and pets may damage cords, pull tubing, press buttons, and so on. This hazard is on a different level than the others we describe in this section, as it can not be directly detected by the pump's sensors. Instead, the sensors detect the effects of the child or pet's actions. For instance, if the pet pulls on the tubing between the fluid reservoir and the pump and causes a leak, the pump may only detect that it is not getting fluid. The pump can only determine the effects of this hazard and not the cause. That is enough, however, to allow the pump controller to take action by sounding an alarm and stopping the pump. This hazard can be mitigated by locking down the pump programming and arranging the pump so that tubing and cables are not readily accessible.

Overdose. Patients on a PCA pump are given a button they can use to request bolus doses of medication. There are no limits on how often or how frequently the patient can press the button, but the pump must not give the patient too much drug, as this would cause an overdose. The pump should be programmed with proper drug concentration and limits (The pump should check the limits and respond appropriately- if the total dose and frequency are below the preset limits, then the patient is given a bolus dose. Otherwise, the pump does not deliver a dose, and it may signal the patient so they can see that their inputs are not being ignored).

4 Generic PCA Requirements and Hazard Analysis

The first step in building the reference model was to come up with a set of requirements. These requirements were collected from various sources and were added to throughout the development process. The requirements are structured hierarchically following the structure of the pump classification. There are requirements that apply to all infusion pumps, additional requirements for small volume pumps, and another set which are specific to PCA pumps. Each set of requirements builds upon the higher level requirements so a PCA pump implementation should meet all three of these sets.

The top level requirements, which should apply to all infusion pumps, are quite generic. They include statements like "The pump should transfer fluid at the commanded rate" and "The pump must not pump air bubbles into the patient". Infusion pumps are heterogeneous enough that it is quite difficult to find requirements common to all of them.

We generated our requirements by examining literature about numerous pumps on the market, by reading infusion pump patents, and through discussions with our collaborators at the Hospital of the University of Pennsylvania, Duke University Hospital, ECRI, and elsewhere.

The question of completeness of requirements is often raised. We believe that the best way to show that a set of requirements is complete is to use them to build a working system. It is usually the case that missing requirements will be found and filled in during the process of building the system.

The complete list of design requirements is too long to present here. Instead, we will show some of the requirements most closely associated with the example hazards listed above.

Output side occlusion. Our design assumes a pump with pressure sensors on the line from the fluid reservoir to the pump (input side) and the line from the pump to the patient (output side).

An occlusion alarm sounds when a pressure sensor reading goes above some threshold and clears when the reading drops below it and the alarm is reset by the user.

Air bubble. The timing from detection of an air alarm condition to stopping pump must be such that bubbles can not be pumped to the patient. This should hold even if the smallest (shortest and smallest diameter) possible tubing is used.

When an alarm occurs, the pump should immediately stop pumping. A time should be specified which makes the pump stop before it could push an air bubble through the tube to the patient.

$$T_{stop} < V_{tubing}/(S_{pump} + T_{pump} * S_{pump})$$

Where T_{stop} is the amount of time the pump has to stop transferring fluid, V_{tubing} is the volume of the tubing set, S_{pump} is the maximum speed of the pump, and T_{pump} is the tolerance for the pump speed (e.g., $\pm 10\%$).

The air alarm sounds when the conductivity sensor reading goes above some threshold and clears when the reading drops below it. Pumps with an air alarm measure the conductivity of the fluid they are pumping. If the reading goes out of bounds, this means that there is air in the line. (These alarms may also be tripped by using the wrong fluid, for instance pure water, another potentially hazardous situation).

Misuse. There are no specific design requirements for pump misuse because the pump sensors can not distinguish between a pump misuse and normal use hazardous situation.

For example, leaks and blockages of the tubing set, which are covered by the air or occlusion alarm. Children may also turn knobs or play with the pump controls. This is one reason that the controls of PCA pumps should be designed such that only an authorized caregiver can introduce changes. This can be accomplished with a mechanical lock or through software.

Overdose. The user interface for PCA pumps adds a means of setting the bolus dose volume, a means of setting the maximum dosage per unit time (max dose), a means of setting the time delay between boluses (delay time), and a button which the patient can push.

When the patient button is pushed, the pump will deliver a bolus of the volume specified by the bolus dose volume setting if and only if these conditions are met:

- The bolus dose would not cause the total infused volume to exceed the max dose.
- It has been at least the established delay time since the last bolus event.

Once the pump is started, all pump settings are locked in and it should not be possible to change them while the pump is running.

5 The Generic PCA Reference Model

The generic PCA infusion pump reference model contains components modeling the caregiver, pump system, infusion set, and patient, as shown in Figure 2. The purpose of this model is to provide a formal reference implementation of a pump system which can be used to generate tests for other pump implementations.

We modeled the pump system as a set of extended finite state machines (EFSMs) which communicate using synchronized channels. This EFSM model is then automatically translated into an UPPAAL model [1] so we can check it for properties derived from the hazards analysis.

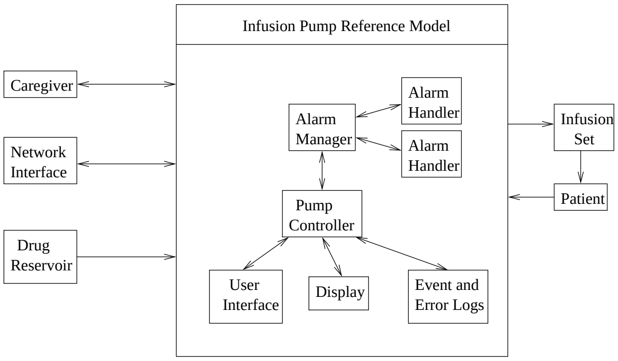

Figure 2. System Overview

5.1 User Interface

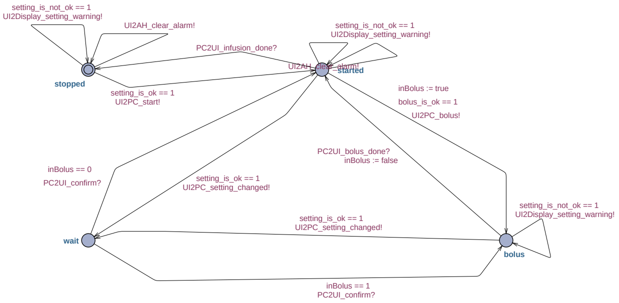

Figure 3. User Interface

The user interface component takes inputs from the user and synchronizes with the rest of the system, particularly the pump controller. This component controls when the user may trigger different inputs. For instance, the pump must be started before a bolus dose may be triggered. This is accomplished, as shown in Figure 3, by only allowing UI2PC_Bolus signals when the UI EFSM is in its 'started' state.

The UI EFSM has four states: stopped, started, bolus, and wait. The first three match the conditions when the pump is stopped, started, or delivering a bolus dose. The wait state is used when the user interface has signaled the pump controller that the settings have

[1] There is nothing specific about UPPAAL for our methodology. We could have easily used other model checkers such as NuSMV, SPIN, VERSA, etc.

changed and is waiting for a confirmation before allowing further input.

5.2 Pump Controller

The pump controller handles much of the core functionality of the reference model. Its five states - stopped, started, bolus, setting, and alarming - match the possible modes of the pump. Critical functions such as stopping the pump motor when an alarm occurs are performed by this component.

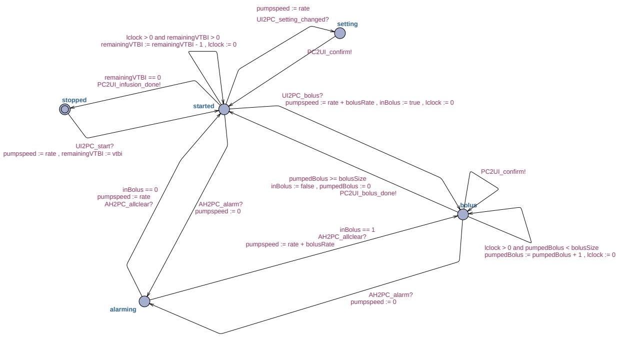
pumpspeed := rate
UI2PC_setting_changed?
setting
lclock > 0 and remainingVTBI > 0
remainingVTBI := remainingVTBI - 1 , lclock := 0
PC2UI_confirm!
remainingVTBI == 0
PC2UI_infusion_done!
stopped
started
UI2PC_bolus?
pumpspeed := rate + bolusRate , inBolus := true , lclock := 0
UI2PC_start?
pumpspeed := rate , remainingVTBI := vtbi
PC2UI_confirm!
pumpedBolus >= bolusSize
inBolus := false , pumpedBolus := 0
PC2UI_bolus_done!
inBolus == 0
pumpspeed := rate
AH2PC_allclear?
AH2PC_alarm?
pumpspeed := 0
bolus
inBolus == 1
AH2PC_allclear?
pumpspeed := rate + bolusRate
lclock > 0 and pumpedBolus < bolusSize
pumpedBolus := pumpedBolus + 1 , lclock := 0
AH2PC_alarm?
pumpspeed := 0
alarming

Figure 4. Pump Controller

5.3 Alarm Manager

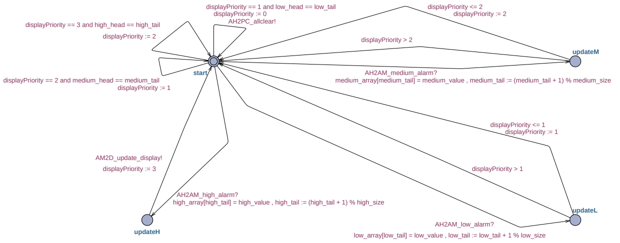
displayPriority == 1 and low_head == low_tail
displayPriority := 0
AH2PC_allclear!
displayPriority <= 2
displayPriority := 2
displayPriority == 3 and high_head == high_tail
displayPriority := 2
displayPriority > 2
updateM
start
AH2AM_medium_alarm?
medium_array[medium_tail] = medium_value , medium_tail := (medium_tail + 1) % medium_size
displayPriority == 2 and medium_head == medium_tail
displayPriority := 1
displayPriority <= 1
displayPriority := 1
AM2D_update_display!
displayPriority := 3
displayPriority > 1
AH2AM_high_alarm?
high_array[high_tail] = high_value , high_tail := (high_tail + 1) % high_size
updateL
updateH
AH2AM_low_alarm?
low_array[low_tail] = low_value , low_tail := low_tail + 1 % low_size

Figure 5. Alarm Manager

The alarm manager is responsible for keeping track of which alarm conditions are active at any time. When more than one alarm goes off at one time, the alarm manager must also decide which active alarm should be shown.

Alarms can be given priorities to force more important alarms to be handled before less important alarms. For instance, the designers may decide that the air alarm is more important (i.e., higher priority) than the occlusion alarm since air bubbles may harm the patient while a blockage in the line may be only an inconvenience. This system has three levels of alarm priorities: high, medium, and low. High level priority alarms override lower level priority alarms. There are three alarm queues, one for each priority level.

The alarm manager state machine uses three arrays to keep track of the three alarm priorities. Each of these arrays is initialized to be larger than the number of alarms in the corresponding priority group. Each array has a head and tail pointer, which is an integer pointing to the position in the array holding the first and last active alarm in the group. The array starts out empty with the head and tail variables pointing to the first position. When an alarm is activated, it is added to the array corresponding to its priority level at the position indicated by the tail variable and the tail is incremented. Alarms are removed by incrementing the head. All arithmetic is done modulo the size of the array so that the positions wrap around correctly. Another variable, displayPriority, is used to keep track of which array has the highest priority active alarm. This alarm is the one which should be displayed to the user, and it is always found at the head position of the alarm array named by displayPriority.

5.4 Alarm Handlers

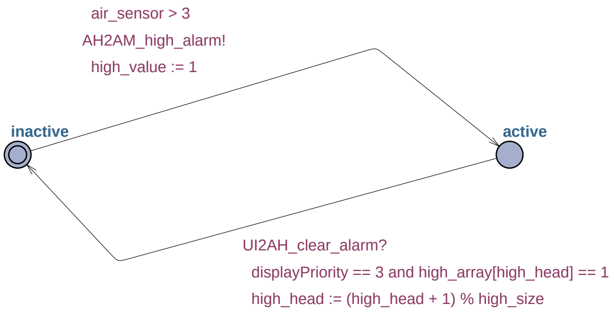
air_sensor > 3
AH2AM_high_alarm!
high_value := 1
inactive
active
UI2AH_clear_alarm?
displayPriority == 3 and high_array[high_head] == 1
high_head := (high_head + 1) % high_size

Figure 6. Air Alarm Handler

Each alarm has its own handler. When an alarm occurs, its handler catches that event and sets an internal variable representing the alarm to high. When the alarm is cleared, the internal variable is reset, showing that the alarm condition is no longer true. This variable does not directly alter the behavior of the pump-resetting it will not stop the alarm annunciation or restart the pump if it has stopped. The alarm handler only keeps track of whether the alarm condition is true or false at a particular instant and makes that information available to the higher level control algorithm.

5.5 Display

The display component is the interface from the pump system to the caretaker. It consists of a text display, a light, and an audible alarm. The text display is used to show short messages such as the current pump speed, active alarms, or other information about the status of the pump.

In this model, the display is abstracted and simplified. The display component synchronizes with other components when a new display message is set, but does not do anything else with it.

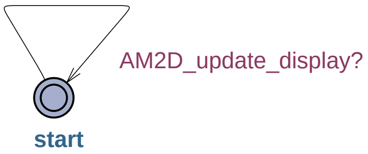

Figure 7. Display

5.6 PCA Patient Model

We model the patient as a black box whose only response to treatment is periodically pressing a bolus button. The patient model contains a variable representing the level of medication absorbed by the patient. As the patient receives periodic doses of medication from the pump, it is absorbed in the patients bloodstream at a rate generally unique to the patient. Our model assumes that the patient absorbs medication at a rate higher than the basal rate of the pump. This means that over time, the level of medication in the patients bloodstream will decrease.

We also define a pain threshold variable, which represents a level of absorbed medication below which the patient feels enough pain to want to press the bolus button for more medication. Pressing the bolus button will increase the level of medication in the bloodstream reducing the level of pain below the pain threshold level for some period of time.

The behavior of the patient model is shown in Figure 8, which plots the patients pain level over time. Points 1, 2, and 3 represent points in time when the patients pain threshold level "t" is exceeded and patient presses the bolus button to request more medication.

This simple model could be improved by recognizing that the pain threshold is different for every patient and changes over time even for the same patient.

6 Verifying the Generic PCA Reference Model

One of the key reasons for building a formal reference model is that we can check it for desirable properties. This allows us to prove that the requirements are met, that hazards are avoided, and that certain structural problems such as deadlocks are avoided.

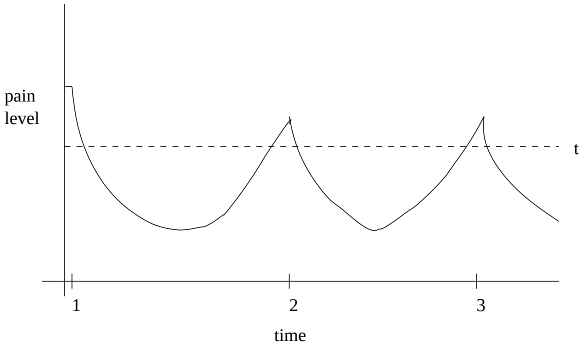

Figure 8. Graph of the patient model behavior

6.1 Structural Properties

Structural properties are properties of the model which can be checked without any domain knowledge. These properties include nondeterminism, completeness, and the presence of deadlocks.

6.1.1 Checking for nondeterminism

An EFSM is nondeterministic if there exists some state with two or more outgoing transitions which may be enabled at the same time. In order to check for this condition, we check that it is not possible for more than one transition from any state to be enabled. This is done by finding the set of guard conditions $g_1, g_2, \ldots, g_n$ for the transitions from each state then checking that the expression $g_i \wedge g_j$ is not satisfiable for any $i < j <= n$. The check is performed by feeding the expression into a boolean satisfiability (SAT) solver. This process is illustrated in Figure 9. The input format for most SAT solvers is DIMACS, which is a syntax for expressions in conjunctive normal form (CNF). In order to use a SAT solver to check these expressions, we must first convert the guards into CNF.

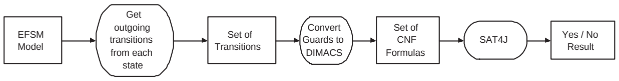

Figure 9. Using a SAT solver to check the model

6.1.2 Checking for Completeness

Another property we can test for is completeness, or totality. This is a way of saying that the system doesn't get stuck. That is, at each state S with outgoing transitions $t_1, t_2, \ldots, t_n$ and corresponding guards $g_1, g_2, \ldots, g_n$, $(g_1 \vee g_2 \vee \ldots \vee g_n) \rightarrow True$.

6.1.3 Checking for deadlocks

UPPAAL allows us to easily check that the reference model is deadlock-free, since this test is built into the tool.

6.2 Safety Properties

The category of safety properties encompasses a broad range of properties related to system safety. These include properties related to many of the hazards and the requirements. As space does not permit the complete list of properties to be included here, we have again limited the list here to the properties associated with the hazards and requirements discussed above.

In order to check properties with UPPAAL, they must be expressed in a dialect of temporal logic. For readability, they are also written here as English sentences.

Output side occlusion. If the output side pressure is over the permissible limit, then the output occlusion alarm shall be active and the pump shall be stopped.

$\Box P_{out} > L_{out}$ imply AH_{Occ} ==active

Air bubble. If the air sensor reading is over the threshold, then the air alarm should be active and the pump should be stopped.

$\Box S_{air} > L_{air}$ imply AH_{air} == active and pumpspeed == 0

The time from when the air alarm is activated until the pump is stopped must be less than the user defined maximum.

Misuse. This hazard expresses itself as a wide variety of effects. Each of these effects is handled by a sensor, alarm, and associated set of properties. For example, a pet may chew the input side tubing, admitting air. This is handled by the air alarm and is checked using the air alarm property above.

Overdose. There are several cases:

1. If a bolus is requested and the total infused + bolus size > max dose, then the bolus must not be delivered.

2. If a bolus is requested and the current time - time of last bolus < delay time, then bolus must not be delivered.

3. If a bolus is requested and neither 1 nor 2 is applicable, then the bolus may be delivered.

6.3 Test Generation

Test case sequences are derived from the reference model by performing a walk through or guided simulation through the state space of the model. Test sequences are generated by providing the reference model with an extensible application interface (AI). The application interface for the model maps the abstract state space of the model to concrete structures in the code. Every time, during simulation, a target state (or transition) is encountered, the component generates stimuli that the corresponding event would produce in the device implementation. The various stimuli generated during the traversal are recorded to produce a (unique) test sequence. An exhaustive collection of all such scenarios for an implementation is referred to as a *test suite.*

The criteria for recording the stimuli are defined with respect to the syntax of the reference model. Each model, defined in terms of an automaton, contains states and transitions. Each transition in turn may have a label that includes all or some of the following: an event, a condition, a condition action, and a transition action. A transition's condition action is executed whenever it is deemed ready to fire. The transition's action is executed only when it is included in such a firing transition.

The criteria for generating test case sequences can thus be based on the following coverages:

1. State coverage. This criterion is met when each state in the model has been entered at least once.

2. Condition action coverage. A condition action is deemed to be covered if it has been executed at least once, or, if the segment has no condition action, if its condition has evaluated to `true` at least once. This criterion is satisfied when all target conditions have been covered.

3. Transition action coverage. This criterion is met when all target transition actions have been executed at least once. If a transition has no associated action, it is ignored by this criterion.

The test cases generated via reference models help the regulator validate individual implementations using a black-box testing approach. This is ideally suited to a conformance review process, as the regulator is not concerned about details of the implementation during this process. Our future work is to develop other test converages and evaluate their effectivenesses.

7 Related Work

There have been several studies involving the use of formal methods-based analysis of safety-critical systems, mostly for embedded systems [3]. Sreemani and Atlee [20] used SMV to analyze the A-7E aircraft software requirements. Bharadwaj and Heitmeyer [2] continued this line of work, using SMV and Spin for other SCR requirements. Pugliese and Tronci [16] developed a process-algebra specification from an informal specification of a hydroelectric power plant, which was then verified with an in-house BDD-based model checker. Crow and Di Vito [5] verified invariants of the requirements for a software subsystem on the Space Shuttle of NASA with the explicit model checker $Mur\phi$ [6]. Using Spin, Havelund et al. [11] found errors in a spacecraft controller, while Schneider et al. [18] validated a model of a fault-tolerant system specified as a hierarchical state machine.

Efforts involving formal verification of medical devices, and infusion pumps in particular, have dealt mainly with model checking of the CARA system. Alur et al. [1] made use of Extended Finite State Machines (EFSMs) to formally model CARA, and verified the specifications using tools supporting formal analysis, such as SCR and Hermes. Ray and Cleaveland, on the other hand, used the CWB-NC to analyze the CARA system, using a technique called unit verification [17], which entails verifying individual small units of a system and repeatedly combining them to form more tractable units. The GPCA project builds upon these efforts, employing many of the same techniques to define a formal approach for conformance review in medical device software.

This process, viz. reference model based conformance review, itself is similar to the Family-oriented Abstraction, Specification and Translation (FAST) process [4]. FAST is a development process for producing software in a family-oriented way that applies software product-line architecture principles [21] into software engineering process. The FAST process separates product-line engineering process into two main parts. One step concentrates on providing the core assets including the environment for implementing each product. The other step utilizes the environment in the production of different software products belonging to the family. Similarly, our approach has a core (GPCA) model defining generic safety properties, allowing specific implementations to specify the architecture for individual devices.

8 Discussion

We have presented a model-based approach to conformance testing of medical devices. While no individual step of this methodology is new, their application in this domain raises interesting issues. FDA staff at CDRH currently use a quality process documentation based approach to reviewing submitted devices. In this approach, little is revealed about the properties of the implemented software. The challenge is to establish a methodology whereby a significant portion of the some fifteen thousand medical device manufacturers can present substantive arguments about certain properties of the software, such as safety and reliability, to the regulators.

There are many issues to overcome before this vision can be realized. First and foremost is overcoming complacency. We must demonstrate that the methods we offer will permit manufacturers to build their software faster, better, and cheaper. This will involve comparing current process based methods to formal based methods in terms of measures that can readily be converted to financial gain. We must provide appropriate tools that facilitate principled engineering based software development manufacturers can trust. Such tools should serve to minimize the regulatory burden of manufacturers, perhaps by automatically managing properties of regulatory interest, while at the same time assuring product integrity.

This effort represents the first step of many to support an argument that formal methods based software development is a practicable means of building software faster, better, cheaper, and with a positive regulatory impact. We have begun the journey by establishing a reference device of moderate complexity. By going through the development process ourselves we have the opportunity to learn about the kinds of things our tools and manufacturers need to do. For example, we have learned that some available academic (open source) tools are better at some things than others. But virtually none of them are seamlessly interoperable. This presents a significant hurdle for manufacturers to deal with. We have also learned that modeling even simple devices requires that the tools need to manage a large number of states. There are further challenges in providing tools that present information in a form that device domain experts can identify with.

Once we have established a reasonable level of design maturity it will be necessary to compare our model with real implementations. This will entail collaboration of some form with a manufacturer of this type of device- a not too insignificant challenge in itself.

While our work may serve to benefit one particular type of device, it is clear that we can not do this for each device coming to market. Rather, this work should ultimately serve to demonstrate what is needed for manufacturers to produce similar results themselves.

A potential benefit of this research is that we may establish a reference standard for this type of device. By enhancing our approach, we provide the capability to (reference) test implementations for a set of important safety properties that all implementations of this device type should meet.

We plan to publish our design work as an open source resource for manufacturers to use as a reference for implementations and for academics to experiment with.

9 Conclusion and Future Work

We have developed a methodology for conformance testing device implementations and begun a case study applying it to PCA infusion pumps. We gathered lists of hazards and requirements, built a reference model for this class of pumps, and tested the model for structural and safety properties.

Our immediate future work is to generate tests from the model and use the tests to check a pump implementation. We will develop the notion of a generic infusion pump as a hierarchy of pump models. Requirements for various classes of pumps overlap a great deal. We can exploit this overlap to build a family of pump models with shared requirements and components.

10 Acknowledgements

We would like to thank Victoria Rich and Poppy Bass of the Hospital of the University of Pennsylvania and Erin Sparnon from ECRI for their help. Without the assistance of domain experts and clinical practitioners this work would not have been possible.

References

[1] Rajeev Alur, David Arney, Elsa L. Gunter, Insup Lee, Jaime Lee, Wonhong Nam, Frederick Pearce, Steve Van Albert, and Jiaxiang Zhou. Formal specifications and analysis of the computer-assisted resuscitation algorithm (CARA) infusion pump control system. *Software Tools for Technology Transfer*, 5(4):308–319, 2004.

[2] Ramesh Bharadwaj and Constance L. Heitmeyer. Model checking complete requirements specifications using abstraction. *Autom. Softw. Eng.*, 6(1):37–68, 1999.

[3] Edmund M. Clarke and Jeannette M. Wing. Formal methods: state of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, 1996.

[4] James Coplien, Daniel Hoffman, and David Weiss. Commonality and variability in software engineering. *IEEE Software*, 15(6):37–45, 1998.

[5] Judith Crow and Ben L. Di Vito. Formalizing space shuttle software requirements: Four case studies. *ACM Trans. Softw. Eng. Methodologies*, 7(3):296–332, 1998.

[6] D. L. Dill. The Murϕ verification system. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 390–393, New Brunswick, NJ, USA, 1996. Springer Verlag.

[7] ECRI. Patient-controlled analgesic infusion pumps. *Health Devices*, 2006.

[8] Food and Drug Administration. General principles of software validation; final guidance for industry and FDA staff. Technical report, Food and Drug Administration, 2002.

[9] Food and Drug Administration. Guidance for the content of premarket submissions for software contained in medical devices - guidance for industry and FDA staff. Technical report, Food and Drug Administration, 2005.

[10] Klaus Havelund, Michael R. Lowry, and John Penix. Formal analysis of a space-craft controller using SPIN. *Software Engineering*, 27(8), 2001.

[11] Klaus Havelund, Mike Lowry, and John Penix. Formal analysis of a space-craft controller using SPIN. *IEEE Trans. Softw. Eng.*, 27(8):749–765, 2001.

[12] Thomas A. Henzinger and Howard Wong-Toi. Using hytech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications*, pages 265–282, 1995.

[13] IEC International Electrotechnical Commission. IEC 60601-2-24 medical electrical equipment- part 2-24: Particular requirements for the safety of infusion pumps and controllers. Technical report, IEC, 1998.

[14] IEC International Electrotechnical Commission. ANSI/AAMI/ISO 14971, medical devices – application of risk management to medical devices. Technical report, 2000.

[15] Insup Lee, George J. Pappas, Rance Cleaveland, John Hatcliff, Bruce H. Krogh, Peter Lee, Harvey Rubin, and Lui Sha. High-confidence medical device software and systems. *Computer*, 39(4):33–38, 2006.

[16] Rosario Pugliese and Enrico Tronci. Automatic verification of a hydroelectric power plant. In *FME*, pages 425–444, 1996.

[17] Arnab Ray and Rance Cleaveland. Unit verification: the CARA experience. *Software Tools for Technology Transfer*, 5(4):351–369, 2004.

[18] Francis Schneider, Steve Easterbrook, John R. Callahan, and Gerard J. Holzmann. Validating requirements for fault tolerant systems using model checking. In *ICRE '98: Proceedings of the 3rd International Conference on Requirements Engineering*, pages 4–13, Washington, DC, USA, 1998. IEEE Computer Society.

[19] Tirumale Sreemani and Joanne M. Atlee. Feasibility of model checking software requirements. In *Compass'96: Eleventh Annual Conference on Computer Assurance*, page 77, Gaithersburg, Maryland, 1996. National Institute of Standards and Technology.

[20] Tirumale Sreemani and Joanne M. Atlee. Feasibility of model checking software requirements. In *COMPASS'96, Proceedings of the 11th Annual Conference on Computer Assurance*, pages 77–88, Gaithersburg, Maryland, 1996. National Institute of Standards and Technology.

[21] David M. Weiss and Chi Tau Robert Lai. *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.