Design Safety Verification of Medical Device Models using Automata Theory

A Thesis Presented to
The Faculty of Computer Science Program
California State University Channel Islands

In (Partial) Fulfillment
of the Requirements for the Degree
Masters of Science in Computer Science

Hita Gambheer

Supervised by
Dr. Michael Soltys

08/01/2016

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

12/04/16

Advisor: Dr. Michael Soltys — Date

12/01/16

Dr. Brian Thoms — Date

12/01/2016

Dr. David Claveau — Date

APPROVED FOR THE UNIVERSITY

12-1-16

Dr. Gary Berg — Date

Non-Exclusive Distribution License

In order for California State University Channel Islands (CSUCI) to reproduce, translate and distribute your submission worldwide through the CSUCI Institutional Repository, your agreement to the following terms is necessary. The author(s) retain any copyright currently on the item as well as the ability to submit the item to publishers or other repositories.

By signing and submitting this license, you (the author(s) or copyright owner) grants to CSUCI the nonexclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video.

You agree that CSUCI may, without changing the content, translate the submission to any medium or format for the purpose of preservation.

You also agree that CSUCI may keep more than one copy of this submission for purposes of security, backup and preservation.

You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. You also represent and warrant that the submission contains no libelous or other unlawful matter and makes no improper invasion of the privacy of any other person.

If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant CSUCI the rights required by this license, and that such third party owned material is clearly identified and acknowledged within the text or content of the submission. You take full responsibility to obtain permission to use any material that is not your own. This permission must be granted to you before you sign this form.

IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN CSUCI, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT.

The CSUCI Institutional Repository will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

Design Safety Verification of Medical Device Models using Automata Theory

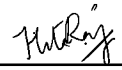
Title of Item

Model Safety Verification using Timed Perti Nets and Automata Theory

3 to 5 keywords or phrases to describe the item

Hita Gambheer

Author(s) Name (Print)



12-01-2016

Author(s) Signature Date

Abstract

The medical device industry is solely dependent on technology to aid continuous monitoring of the human body in difficult medical conditions. These monitoring and responsive devices are life savers of many patients and at the same time cause severe bodily harm if they do not function as expected. Today, there are several design techniques used to design safe devices with minimum risks but there are not many techniques used to isolate edge case scenarios and disruptive states within the model. The objective of this thesis is to model a device into a state machine and a Petri-Net to observe all possible state flows of the device. The flows depict *logic* of the device and reachability analysis on the *logic* helps us identify the *safety rules* required to maintain the integrity of the system.

The methodology adopted to validate this theory starts by choosing a known faulty design of an Artificial Pancreas and performing reachability analysis on the design to isolate the state causing the failure. A simulation of the system is performed on a diabetes data set from a known source to support the results from the reachability analysis; the results show that the system fails. Therefore, this thesis convincingly shows that an automata based approach to minimize design risk's can be adopted to validate the safety of a device and the conclusion describes possible solutions with future work.

Acknowledgment

I would like to thank Dr.Michael Soltys for his guidance and support to finish my thesis in time. I would also like to thank Dr. David Claveau and Dr. Brian Thoms for evaluating my thesis. Finally, I would like to thank my family supporting and encouraging me to do my best.

Contents

List of Figures

Chapter 1

Introduction

1.1 Need for Safety Verification of Medical Device Designs

The physiological interaction of a human being with an application is safety critical and therefore it is necessary for constant evolution of verification standards within the medical device domain. The biggest challenge is to combine the complexities of the physiological systems with the possible states of the application. Through this thesis research, an approach to identify possible risks within an unsecured medical application's design using automata theory is described. The theory is proven by choosing a faulty design and exploring methods to isolate the unsafe states. The main objective is to make the application's design as secure as possible through exhaustive verification.

In recent times *"Model- Based- Design"*[1] is considered a core entity of the software development process. Model's reveal clarity in state transfer and possible leaks in the application. This is a good technique to build bug free applications from the beginning of the development, especially in the field of medical devices where rigorous verification is needed because the cost of bugs are very high.

With safety as a high concern an exhaustive development process from ground up is becoming widespread. The device used in this thesis research is the Artificial Pancreas which comes under the domain of "Embedded Software". The term "Embedded Software" means software that runs on a computer system and strongly sometimes solely interacts with a specific environment.

The important factor in the device of our choice is that the timing and physical characteristics of the environment are essential for the correctness of the system as well as for performance. The control logic is dependent on various states that are continuously activated and provide decision factors for other states to respond. The models of timed and hybrid automata can capture all these factors

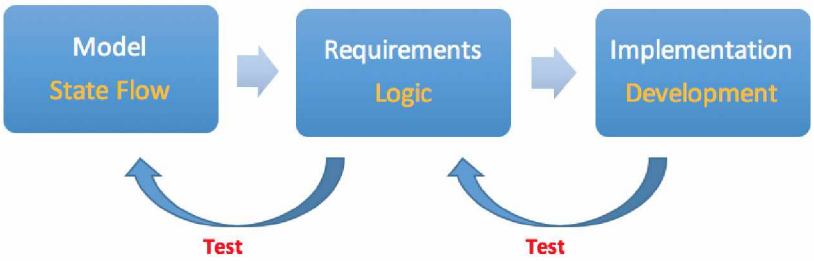


Figure 1.1: *Model Based Design* represents the core entity of software development process

and analyze the potential bug causing states to prevent faulty designs.

The following tasks are considered to justify the verification capability of timed and hybrid automata models:

- Translation: Translating the scope of the problem into variables.
- Modeling: Implementing the variables and their evolution into states
- Verification: Applying "model checking", an exhaustive verification technique.
- Reachability Analysis: Finding possible "safe" and "unsafe" states.
- Testing : Applying general testing techniques for hybrid and timed automata.

The approach of this research deals with some of possible verification techniques closely suited to small size medical devices that can be designed on timed and hybrid automata; however, there are several more techniques and the field is very wide. The main idea is to build on the problem posed in the paper "Using formal Methods to Improve Safety of Home Use Medical Devices" where a fault in a design is identified but there was no formal verification techniques used to isolate the cause of the fault. We try to identify the faulty states by using reachability analysis and confirm our findings with simulation.

1.2 Related Work

This research is conducted on the basis of a previous study that applied the Hybrid Automata (henceforth HA) technique to a simple Artificial Pancreas design. A challenge limiting the applicability of Hybrid Automata in medical devices is

the *state space explosion* phenomenon. Predicting Blood Glucose (henceforth BG) levels and analyzing previous BG values is computed as a linear function. Modeling this predictor requires a large set of variables. Consequently the state space of the model can be too large to handle since HA models are memory-less. By memory less we mean, it depends only on the last node to move forward - does not remember the past (history).

The author in the paper "Using Formal Methods to Improve Safety of Home Use Medical Devices"[2] tries to overcome this problem by designing a closed loop system logic with the variables being generated on necessity to solve a situation. The data manager, which in this case is the sensor is separated from the controller logic.

1.3 Motivation and Problem Approached

The paper "Using Formal Methods to Improve Safety of Home Use Medical Devices"[2] was chosen to understand how to design system models minimizing the chances of faults. Localizing the faults in a system we already know that is defective is easier to analyze; therefore, this thesis research studies the design of the Artificial Pancreas device described by the author and tries to validate it through Automata Theory.

Also the model checking of the design is performed by using additional verification techniques and simulation that enable a more comprehensive assessment of home use device safety. The author in the related work[2] states that this system leads to Hypoglycemia or Hyperglycemia in patients. Hypoglycemia is low BG level and Hyperglycemia is a high BG level. This defeats the main safety property that the device must maintain the patient's body in normal BG level range.

There are many devices in the market and also in research in this area. This research thesis concentrates mainly on designing a small medical device using Automata Theory with a strict check to evade faults. It also includes fining areas the automata could have missed by using Timed Petri - Nets.

The approach to build a verification method in this research is by performing reachability analysis and simulating the closed loop automata using a reliable patient database from a known source(UC Irvine Machine Learning Repository, Diabetes Data Set)[12]. The results of the data integrated analysis on the logic is used to isolate the state(s) where the system fails.

1.4 Contribution

The research paper "Using Formal Methods to Improve Safety of Home Use Medical Devices"[2] defines how a time based automated device can be modeled

into a state machine to improve and streamline its functionality. My contribution to this research is to validate the theory. I have used Timed Petri-Nets to validate the logic derived from the automaton. This involves defining the device in terms of a Timed Petri-Net and deducing properties using the Place Invariants method. Some of the properties include time constraints and risk constraints of the device like when a dose is injected and how much of the dosage is safe. The properties define the control flows in all possible states. This helps identify all the loose ends of the model which can lead to an unsafe state.

After removing the unsafe states I verify the final Temporal states for performance and potential deadlocks. Reachability analysis using predicate logic on the device properties defines the reachable states of the model. I validate that these reachable states are on terms with the working rules of the device. Once all the safe states are discovered I finally define the safety rules of the device.

To prove that the model fails without the safety rules I simulated the model to show the failures and analyze the results. As a conclusion I finally propose Monitors to solidify the verification.

1.5 Summary

This research is to combine reachability analysis with safety properties that define the main purpose of the device. Safety properties validate the functioning through all the states of the automata and reachability analysis deals with computing all the possible states from the initial state. When there are safety guards limiting the possibility of reaching the bad and/or dangerous states reachability analysis can reveal all the safe states of the device.

We exploit a faulty design with known bad states to verify our method to identify if the list of good states is complete. The main questions aimed to answer are as follows:

- How robust is the combination of safety properties with reachability analysis?

  Safety properties are derived in Chapter 4 and when combined with reachability analysis the potential “bad” states can be outlined. The verification of this analysis is carried out in Chapter 5 and the ”bad” states which were previously predicted are confirmed. Thus, this strategy is validated to work well the chosen design.

- How well does this design verification strategy work with high risk models?

  This strategy defines a methodology to identify potential design flaws and/or bugs in the design stages of the device before it is implemented.

Isolating these flaws helps to build robust designs that do not need to be implemented to be tested. In chapter 5 we look at how the design can be simulated to check its performance.

- How can this strategy improve the test case coverage?

  When we have all possible states of the device covered within the design, full coverage for test cases can be easily outlined. Algorithms like Rapidly exploring Random trees can help generate test cases within the states.

- How can this strategy reduce cost of the bugs with early identification?

  With majority of the possible flaws identified in the design of the system, this strategy reduces major costs that occur with bugs after implementation. Adding test cases to designs provides a strong foundation for high risk devices that are mandated to be safe.
  Also if identified early new solutions to designs can be explored. In our case, we proposed to use "monitors" to improve the system architecture (of the chosen model) in Chapter 5.

Overall the main idea is to propose a technique for high risk device design checking.

Chapter 2

Preliminaries

2.1 Use of Timed and Hybrid automata in Model Designs

Automata are used to translate applications or devices as states that are based on mathematical formalism. It is useful to make variations in the machine without the need of building it. A finite automaton consists of:

- finite set S of N states
- special $START$ state
- set of $FINAL$ (or accepting) states
- set of transitions T from one state to another.

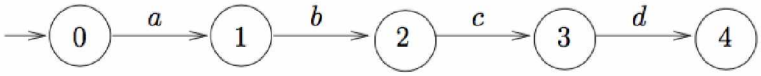
Timed automata is finite automata with time constraints and hybrid automata is timed automata with complex mathematical calculations. The device Artificial Pancreas that has been explored has deterministic and finite states. The following sections will briefly describe the structure of the automata.

2.1.1 Timed automata

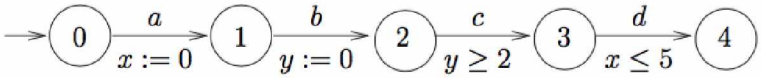
When a finite automaton is extended with real time clocks it is called timed automata. The clocks start with zero and progress at a predefined rate. The clocks mimic variables and are initialized with zero when the system starts, and increase synchronously with the same rate. Clock constraints called guards are used to restrict the behavior of the automaton. A transition represented by an edge can be taken when the clock values satisfy the guard labeled on the edge. Clocks may be reset to zero when a transition is taken.[1]

The Clock constrains contain "location invariants" which define accepting conditions in the automata. An automata may remain in a location as long as the

clock constraints satisfy the invariant condition of the location. For example , Consider the figure 2.2 in which the **end** is marked as an accepting node. The timing behavior is controlled by two clocks x and y. The clock x is used to control the self-loop in the location **loop**. The single transition of the loop may occur when $x = 1$. The clock y controls the execution of the entire automaton. The invariant specifies a local condition that **start** and **end** must be left when y is at most 20 and **loop** must be left when y is at most 50. This gives a local view of the timing behavior of the automaton in each location.



(a) A finite state automaton



(b) A timed automaton

Figure 2.1: The above figure represents a Timed Automata[17] and a regular finite state automata. The difference between the two are the timed variables represented by [x,y]

Definition 1 (Timed Automaton) Assume a finite set of real valued variables C ranged over by x, y, etc standing for clocks and a finite alphabet Σ ranged over by a, b etc. standing for actions. We use $B(C)$ to denote the set of clock constraints.[1]
A timed automaton A is represented as a tuple $\{N, l_o, E, I\}$ where :
- N is a finite set of locations(or nodes),
- $l_o \in N$ is the initial location,
- $E \subseteq N$ is the set of edges
- $I : N \to B(C)$ assigns invariants to locations.

Clock Constraints A clock constraint is a conjunctive formula of atomic constraints of the form $x \sim n$ or $x - y \sim n$ for $x, y \in C$, $\sim \in (\leq, <, =, >, \geq)$and $n \in N$

2.1.2 Hybrid automata

A hybrid automaton is a generalized finite-state automaton that is equipped with continuous variables and discrete changes that work on the lines of the timed automaton. The continuous variables are described by a set of ordinary differential equations and the discrete changes of the hybrid system are modeled

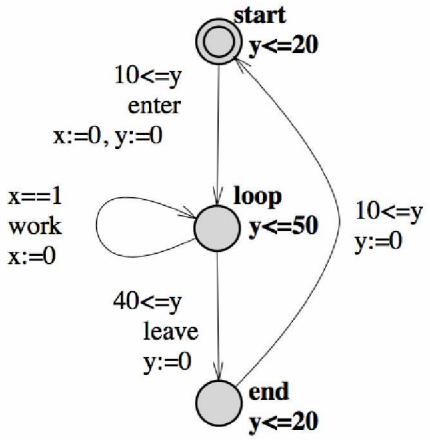


Figure 2.2: Timed Automata Example[6] with a start node, end node and a loop governed by timed constraints.

by edges of the automaton[1]. For example, an automobile engine whose fuel injection, which represents continuous variable is regulated by a microprocessor, which represents discrete variable; by combining them together they represent a hybrid system. Reliability becomes a primary concern as embedded computing becomes ubiquitous and hybrid systems are increasingly employed in safety critical applications. Reliability requires rigorous formal modeling which is achievable through hybrid automata. Consider the example of figure 2.3 which models a thermostat. The variable x represents the temperature. In control mode *off*, the heater is off, and the temperature falls according to the flow condition $\dot{x} = -0.1x$. In control mode *On*, the heater is on, and the temperature rises according to the flow condition $\dot{x} = 5 - 0.1x$. Initially, the heater is off and the temperature is set at 20 degrees Celsius. According to the condition $x < 19$, the heater gets turned on as the temperature falls below 19 degrees Celsius. According to the invariant condition $x \geq 18$, the lowest temperature that the heater will get turned on is at 18 degrees Celsius. The syntax of hybrid automata is defined as follows.

Definition 2 (Hybrid Automata) The timed transition system $\langle S, S_0, \Sigma, \rightarrow \rangle$ of the hybrid automaton H is a tuple of a set of states (S) containing a start state (S_0), set of edges (Σ) and transitions $(\rightarrow)$ where $\{N, l_o, E, X, Jump, Pre, I\}$ is defined as follows [1][6]:

- S is a set of pairs(l, v) where l is the location and v are the variables such that $v \epsilon [lnv(l)]$, this set is called the state space of H.

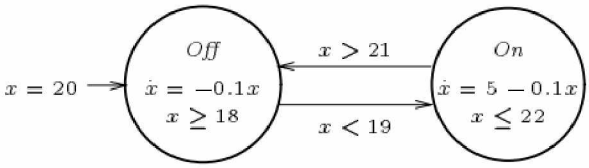


Figure 2.3: Hybrid Automata Example[6] of a room heater system.

- S_0 is the subset of pairs $(l, v)\epsilon S$ such that $v\epsilon[lnv(l)]$, this set is called the initial state space of H.
- N is a finite set of locations(or nodes),
- $l_o \in N$ is the initial location,
- $E \subseteq N$ is the set of edges
- X is the finite set of real valued variables.
- pre is the function that assigns predicates to each location.
- $Jump$ is the function that assigns each edge a predicate.
- $I : N \rightarrow B(C)$ assigns invariants to locations.

In this transition system we abstract continuous flows by transitions retaining only the information about the source, target and duration of each flow. The paths contained in the timed transition system of a hybrid automaton H are formal representations of the possible trajectories of the hybrid system modeled by H, i.e., the evolution of the state of the hybrid system along time. Formally, a *finite path*, noted λ, in the timed transition system $T = \langle S, S_0, \Sigma, \rightarrow \rangle$ is a finite sequence alternating between states and transition labels $s_0\tau_0 s_1\tau_1 ...\tau_{n-1}s_n$ such that at any $i, 0 \leq i \leq n, s_i\epsilon S$ and for any $i, 0 \leq i \leq n, (s_i\tau_i s_{i+1})\epsilon \rightarrow$ .We call n+1 the *length of the path* and it is denoted by $|\lambda|$.[1]

2.1.3 Properties of Hybrid Systems

Safety properties assign values to trajectories of hybrid systems and are the most important class of properties when considering safety critical systems. This thesis research defines properties in terms of the chosen model. In the case of Artificial Pancreas the automata model must enforce the following properties on the trajectory of the entire system.
- No two states must be activated at the same time.
- The states must be activated after a specific period of time has elapsed.
- Overall the system must try to maintain the body in normal Blood Glucose levels.

Further chapters define safety properties in the device model.

Chapter 3

The Artificial Pancreas

3.1 Overview of the Device

The artificial pancreas (AP), known as closed-loop control of blood glucose in diabetes, is a system combining a **glucose sensor**, a **controller algorithm**, and an **insulin infusion device**. AP developments can be traced back 50 years [3] to when the possibility for external blood glucose regulation was established by studies in individuals with type 1 diabetes using intravenous glucose measurement and infusion of insulin and glucose.

The illustration and explanation below describe the parts of an artificial pancreas device system and depicts how they work together.

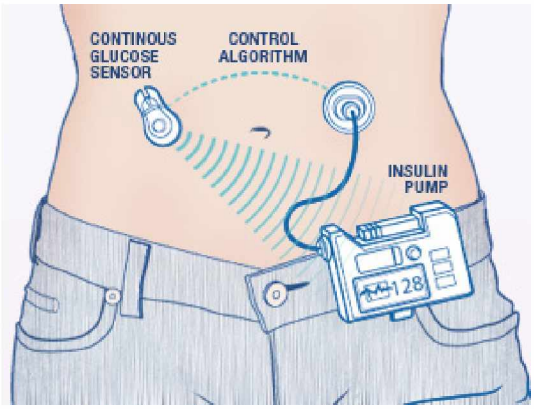


Figure 3.1: Artificial Pancreas [19] device prototype

1. An insulin pump provides continuous delivery of insulin. The types of insulin are :

   - Basal Insulin : Insulin delivered throughout the day to meet metabolic needs and also to balance low blood sugar levels.
   - Bolus Insulin : Insulin delivered to address meals and also to balance high blood sugar levels.

2. A sensor which continuously monitors patient's blood glucose levels.

3. A remote controlled based controller that instructs the insulin pump to adjust the insulin delivery rate accordingly.

3.2 Control Flow of the Device

The control flow of the device begins at the *sensor*. It periodically checks the levels of blood glucose in the body of the patient. The *controller* uses the *sensor* data to adjust the insulin administration and keep the patient's blood glucose level within prespecified safe range. The control of the algorithm divides the operation of the *controller* into three discrete modes.

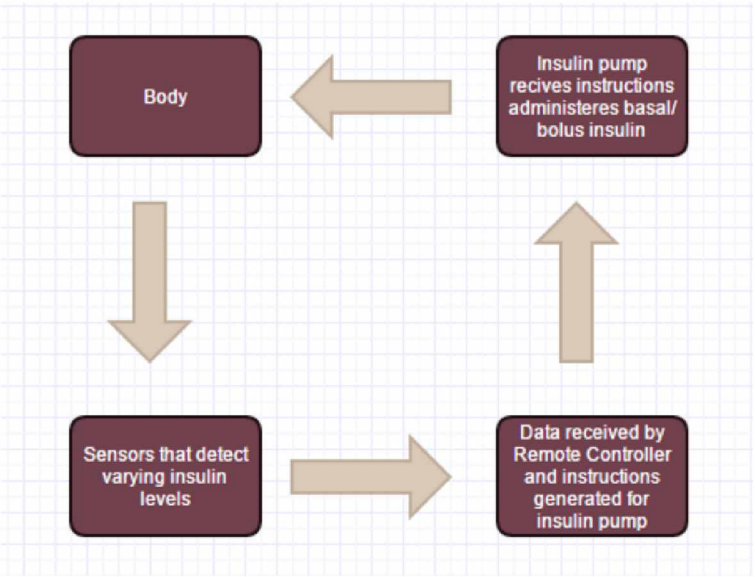


Figure 3.2: Control Flow of the Artificial Pancreas starting with the sensor data to insulin administration into the body.

- **Breaking (BR)**

  This mode continuously adjusts the basal rate (insulin delivered throughout the day) based on the projected blood glucose level. It uses Risk level R(t) to determine basal rate.[2] Let Time = t, The Risk level after 1 hour would be $R(t+60)$. If $R(t+60) >= 180mg/dl$ $R(t)$ is set to 0. If $R(t+60) <= 20mg/dl$ $R(t)$ is set to 100.

- **Meal Supervision (MS)**

  This mode is activated when a patient is about to have a meal. With given meal size(entered by patient), the system calculates the amount of insulin to be administered.

- **Correction Bolus (CB)**

  This mode is activated when all of the following conditions are true:[2]

  1. It has been two hours since last meal bolus insulin.
  2. System is not in breaking mode currently.
  3. At least one hour has passed since last correction bolus.
  4. Blood Glucose is greater than 180 mg/dl.

  The operation always starts with the **Basal Infusion State** and transits to other states based on corresponding 3 transition modes described above.

3.3 System Architecture

The simple finite controller of the Artificial pancreas has 3 possible states, "*Initial State (or) Basal State*", "*Glucose Level Correction State*" and "*Meal Supervision*"[2]. So initially the system checks if the current glucose level is abnormal or influenced by a meal. If abnormal it applies the *Correction Bolus* or the *Breaking state*. If a meal has been noted it applies the *Meal Supervision* state. The final state has to always be the *Basal State*.
To keep the idea closely related to the algorithm of the controller and avoid the complication of biological definitions, the physiological computations of body interaction with medicine's chemical composition have been omitted within each state of the automata.

3.3.1 State Transitions

A new state is reached by using a transition when a condition is *true*. Every state is reached by satisfying the Time and Risk constraints attached to it. The resultant transitions back to the crux of the loop i.e the *Basal State*. It is the state the human body is expected to be in majority of the time. The following diagram is the control logic with arcs which explain the conditions.

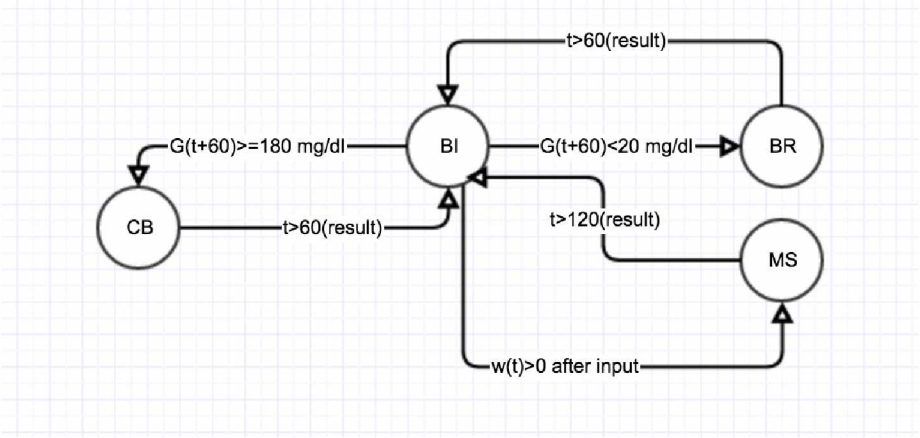


Figure 3.3: System Architecture of the Artificial Pancreas

From the above diagram, we can deduce the following information,

Guard	**State**	**Next State**
t≥60mins	MS/CB/BR	BI
G(t+60mins)≥180	BI	CB
G(t+60mins)<20	BI	BR
w(t) 0	BI	MS

An important observation to make here is, *every state has to transition back to the BI state.*

3.3.2 Building the Hybrid Automata

To design the automata [5] with clocks as guards , it has to be defined as a tuple $\{\Sigma, Q, q_o, F, \delta, I, pre\}$ such that

$L(M) = \{x \in \{0,1\}^* \text{ and } x \text{ ends in } 0\ \}$
$\Sigma = \{0, 1\}$
$Q = \{\ BA, MS, CB/BR\ \}$
$q_o = \{\text{BA}\}$
$F = \{BA\}$
$\delta = \{\delta(BA, 0, MS), \delta(BA, 1, CB/BR), \delta(CB/BR, 0, BA), \delta(CB/BR, 1, CB/BR),$
$\delta(MS, 0, BA), \delta(MS, 1, BA)\}$

I is the invariant that defines every time there is a transition $[x_i = 1$

$\rightarrow S(M, CB/BR, 1) \wedge (x_i = 0 \rightarrow S(M, BA, 0) \vee (x_i = 0 \rightarrow S(M, MS, 0))]$

The I/O behavior is specified in a Boolean format and the following is generated. It is optimized to avoid multiple unnecessary states.

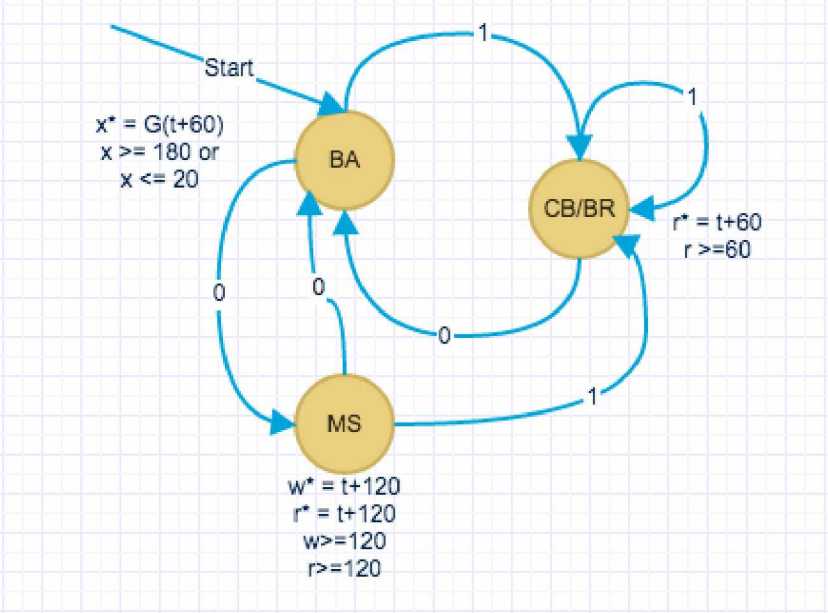


Figure 3.4: Finite State Machine of the Artificial Pancreas

3.3.3 Logic derived from the Automata

Based on the invariant and knowledge available the following conditions are derived,

1. BA is active only when CB/BR and MS are inactive.($\sim CB/BR \wedge \sim MS \rightarrow BA$)
2. MS is active when CB/BR and BA are inactive.($\sim CB/BR \wedge \sim BA \rightarrow MS$)
3. CB/BR is active when BA and MS are inactive. ($\sim BA \wedge \sim MS \rightarrow CB/BR$)

3.4 Software and Human Body

Software behavior has always been a matter of concern in the medical field. The modern computer is dependent on Boolean logic; however, the working of the human body is not. This unpredictable and erratic behavior must be also covered by the medical device model and the device must respond correctly to these situations. Model's encompassing such unpredictable behaviors have been addressed less.

This part of the research aims at covering areas that have not been addressed by automata. Automata are excellent at modeling synchronous state flows; however, for instances where there are concurrent scenarios automata will not be able to produce correct results. To cover these missing spots Timed Petri-Nets are very helpful.

3.5 Timed Petri-Nets

A Timed Petri-Net provides a visual formal modeling method to study the dynamic behavior of systems in terms of the system states and where the states change. They are used to model systems with concurrent activities. Each activity can be represented by a "token" which move within a static graph-like structure of the net. More formally, a marked place/transition Petri net M is defined as $M = (N, m_o)$ where the structure N is a bipartite graph and $N = (P, T, A)$ with a set of places P, set of transitions T and a set of directed arcs A. Places can be connected with transitions and transitions with places, $A \subseteq TXP \cup PXT$. The initial marking function m_o assigns non negative numbers of tokens to places of the net, $m_o : P \rightarrow \{0, 1, ...\}$. Marked nets can be equivalently defined as $M = (P, T, A, m_o)$

A place is shared if it is connected to more than one transition. A shared place p is free-choice if the sets of places connected by directed arcs to all transitions sharing p are identical. A net is structurally conflict free if it does not contain shared places. The model of the artificial pancreas discussed in this research are conflict-free nets.

More formally, a conflict free timed petri net is a pair $T = (M, f)$ where M is a marked net and F is a timing function which assigns an average occurrence time to each transition of the net, $f : T \rightarrow R$, where R is a set of non negative real numbers.

3.6 Modeling Control System using Timed Petri-Nets

As formal models, a Petri net[25] is a directed bipartite graph in which the nodes represent transitions (i.e. events that may occur, represented by bars) and places (i.e. conditions, represented by circles). The directed arcs describe which places are pre- and/or postconditions for which transitions (signified by arrows). Timed petri-nets(TPN) are an extension of petri-nets with additional model of timing. The TPN allows the setup of temporally coherent actions where transition firing dynamics can be specified intuitively in terms of the relevant space (time) when they are derived statistically from data.Also deadlock detection and deadlock prevention seem to be the dominant aspects of application of TPN to embedded software.

The most important part to model in the device is its temporal behavior that relates to its state change. The Timed Petri-nets makes it convenient to interpret properties of the artificial pancreas in the time space. The main idea is to assign each edge with a time stamp. This time stamp makes the decision on which the state can be changed by a binding element[23]. The binding element in this case is the blood glucose value obtained from the sensor. The present state will change when the binding element triggers. And the next trigger of the binding element will occur when the clock is greater than or equal to the timestamps of the edge. After the new binding element trigger occurs the new timestamp of the edge is obtained, which is the time of the current clock added to a time delay. The time delay, defines that the machine should remain in the current state and move only when the delay is completed. The timestamp in this case is considered in minutes.

In the proposed TPN model the following fundamental components are considered: blood glucose values, meal times and trigger clocks. More precisely, each insulin administration represents the blood glucose value and the appropriate time that triggered the sensor to check the state of the blood glucose. Hence a generic set of blood glucose values(L) pertaining to specific triggers can be represented as a set $L = \{L_i | i = 1, ...I\}$ of I insulin administrations where L are a set of *places* and insulin administration the *transition.* In addition, insulin administrations can be classified as : 1) input insulin $L_i \in L_{in}$, that are controlled by a sensor, 2)intermediate insulin $L_i \in L_{int}$ associated to a meal consumption, and 3) output insulin $L_i \in L_{out}$, associated to an elapse of a recorded amount of time.

A generic insulin administration L_i has a finite capacity $T_i > 0$ denoting the amount of time T_i the insulin can control the blood glucose level in the body. Hence each L_i is divided into T_i units of capacity. It must also be taken into consideration that the blood glucose levels may change due to multiple unrelated reasons (not mentioned here) and the machine should accommodate such

unexpected events also. Following section is the derived model as a TPN.

3.6.1 Model

The logic from section 3.3.3 defines each state within the system and the factors for them to be activated. At any given time the following condition should hold good. With reference to section 3.3.3 and ease of notation, let the states be represented with *BA as* ***B****, MS as* ***M*** *and CB/BR as* ***C*** respectively.

$$(\sim B \wedge \sim M \wedge C) \vee (\sim M \wedge B \wedge \sim C) \vee (M \wedge \sim B \wedge \sim C)$$

Essentially since this model has only 3 states each state can be broken down into a model of its own. A very simple model of each state can be represented with two places, p_1 for place 1 that is the abnormal state and p_2 for place 2 that represents a normal state. The transition between p_1 and p_2 is denoted as t_1 and its average time stamp is $f(t_1)$. Place p_2 indicated (by a single token) that the state is in the normal state(waiting) but ready to move to another state in case there is an abnormal BG level. Whenever there is a request in p_1 and p_2 is idle, t_1 is activated and starts its firing by removing single tokens from p_1 and p_2.

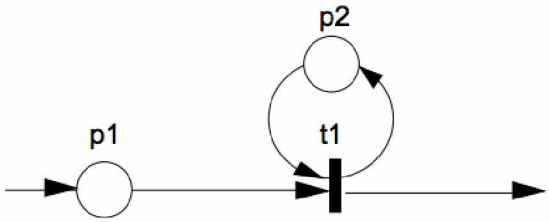


Figure 3.5: Petri net model of a single state with the AP

For a system of this type having distributed request arrivals and erratic service times, a state can be modeled as a single input single output transition with average firing time equal to response time of the system.

With the consolidation of all the three conditions from 3.3.3 an interactive dynamical system is designed. p_2 and t_1 model the system "returning" to the idle (normal) state. They represent the average time the body was normal. The initial marking of p_1 represents the "meal session". p_6 and t_2 represent the "low BG level" state of the body with a queue from p_2, while p_5 and t_5 model a "high BG level" with a queue(waiting time for next BG level test)on p_4. p_3 combines p_5 and p_6 to make the system closed loop.

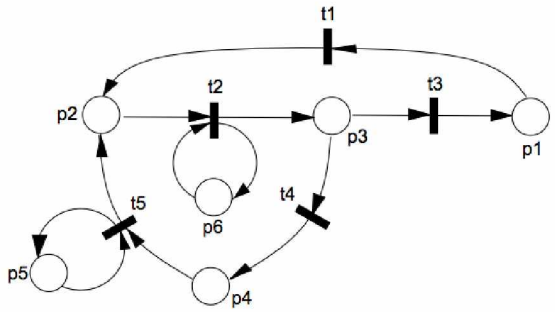


Figure 3.6: Petri net model of the Artificial Pancreas

3.6.2 Place Invariants Method

We can deduce a few properties from the model designed above. The basic idea behind these properties is to create equations that are satisfied in all reachable states. Based on the rules of Timed P-nets[28], a transition can be fired if the global time is great than or equal to the timestamp. It hints that the system's actions are in sync according to the timestamps of the binding elements. In fact the system models require only the timestamps to be small enough instead of requiring them to have some exact time values. This means that linearity of weight functions is insufficient to guarantee that each flow determines an invariant. However, the artificial pancreas system model is predetermining time. Therefore, it is certainly to use invariants in analysis of Timed P-nets models.

There are four equations obtained from Fig.3.6., and the performance of the system is verified them. All the equations are supposed to end on p_2 since it is the normal (idle) state.

1. $p_1 + p_2 = 1$ i.e a meal state should eventually return to idle state.
2. $p_6 + p_3 + p_4 + p_2 = 1$ i.e a low BG level should be addressed and eventually lead to the idle state.
3. $p_3 + p_1 + p_2 = 1$ i.e a sudden meal state activation should be addressed.
4. $p_3 + p_4 + p_5 + p_2 = 1$ i.e a high BG level should be addressed and eventually lead to the idle state.

The above equations show the key invariants in the model. It proves that the control logic modeled by Timed P-nets model is accurate and it does not allow any accident to happen.

3.6.3 Temporal Properties

For the need of making sure the invariants deduced hold good or not in a system specification, we have to be able to express the properties on a formal logic. The logic prove how the property holds good in every reachable state.

Given a system specification **T** if a state **S** of this system satisfies a state formula ϕ , then we denote **T** , **S** = ϕ . If a computation path π satisfies a path formula, then we write **T** , $\pi = \phi$ .Some of the main temporal operators used in temporal logic's are **X, G, F** and **E**. These operators describe properties which may hold or not in a path of states as follows:

- **T**, $\pi \models$ **X** ϕ (next)
- **T**, $\pi \models$ **G** ϕ (globally)
- **T**, $\pi \models$ **F** ϕ (eventually)
- **T**, $\pi \models$ **E** ϕ (path starting at **S**)

We can define the most important features of the system as temporal properties below by using various combinations from the above :

- *"When the body is in an abnormal state of blood glucose, and the patient consumes a meal, the meal state should be able to be activated":* $CB/BR \wedge \sim BA \Rightarrow \mathbf{EF}MS$
- *"When the body is in idle state for long repeated periods of time with no meals taken, the connecting state should be able to reach the correction state for future incidents":* $BA \wedge \sim MS \Rightarrow \mathbf{F}CB/BR$
- *"Every state must end with the state Basal State":* $CB/BR \wedge MS \Rightarrow \mathbf{X}BA$

Chapter 4

Model Checking through Reachability Analysis

Model Checking is a formal verification technique, which is based on the exhaustive exploration of a given state space trying to determine weather a given property is satisfied by the system[4]. The verification can be accomplished by several techniques. Reachability analysis is chosen to determine the set of states that a system can reach, starting from a set of initial states under the influence of a set of input trajectories and parameter values. Reachability analysis is a beautiful method of not only verifying the structure of the model but also validating it against the user requirements.

As already mentioned in Chapter 2, one of the main ideas of this research is to find possible faults in the system and reachability analysis is used to check whether a system can reach a set of unsafe states. A set of unsafe states might be a set of state transitions that lead to the patients BG levels to remain High or Low defeating the purpose of the device.

Besides safety verification there are other possible applications for reachability analysis[7]:

1. Performance assessment of control strategies: It can be checked if the system trajectories stay in a region around a reference trajectory, or reach a goal region around a set point.

2. Scheduling: Reachability analysis can verify if the optimal schedule of a system (typically a production system) is ensured under all conditions.

3. Controller synthesis: The safety verification capabilities of reachability analysis can be used to find parameter sets of controllers that satisfy safety con-

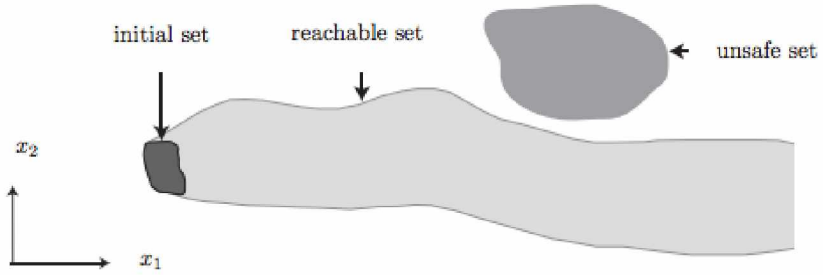


Figure 4.1: Finite State Machine of the Artificial Pancreas[7]

straints.

4. Deadlocks: Reachability analysis can determine whether a system might get stuck in a certain region of the continuous state space or an operation mode of a hybrid system.

4.1 Proof through predicate logic

A predicate[9] is a verb phrase template that describes a property of objects or a relationship among objects represented by the variables. Predicates are functions of zero or more variables that return *Boolean* values. Thus predicates can be true or false depending on the values of their arguments.

Proof through predicate logic[9] can be used to reason systems or "expert" systems such as automatic medical diagnosis programs and theorem-proving programs. In this scope predicate logic can be used to prove the properties of the Artificial Pancreas system.

Since safety verification can be reduced to reachability we can represent the working and safety properties of the Artificial Pancreas device in terms of reachability using predicate logic.

To formalize properties [1][8], following is the notation. Let $T = \langle S, S_0, \Sigma, \rightarrow \rangle$ be a Timed Transition System(TTS). Let $\lambda = s_0\tau_0 s_1\tau_1....s_n$ be a finite path in T. We denote State(λ) for the set of states that appear *along* the path λ. We say that a path λ *reaches* a state s if $s \in$ State(λ). We say that a state s is *reachable* in T if $s \in \cup_{\lambda \in PathF(T)}$ State (λ). The set of states that are reachable in T is noted Reach(T). The set of states $R \subseteq S$ is called a region. We note $\bar{R}$ for the complement of R in the state space of T, that is, $\bar{R} = S/R$. We say that T is *safe for* R iff Reach(T) $\subseteq R$. A region is *reachable* in T iff $R\cap$ Reach(T) $\neq \emptyset$

4.2 Reachability Algorithm

The reachability algorithm[11] is as follows. It can be used to find how many steps are needed to reach the final state in a state machine[8].
Input : Initial state s_o and transition relation δ for closed finite state system M, represented symbolically.
Output : Set R of reachable states of M, represented symbolically.

1. **Initialize:** Current set of reached states $R = \{s_0\}$
2. **SymbolicSearch()** {
3. $R_{new} = R$
4. **while** $R_{new} \neq \phi$ **do**
5. $R_{new} := \{\ s' | \exists s \in R \text{ s.t } s' \in \delta(s)\ \}/R$
6. $R \cup R_{new}$

   end
7. }

4.3 Working Rules of the Device

The following section elaborates on finding the reachable states and possible unsafe states in the Automata based on reachability analysis. We use *Explicit State Model Checking* by manually finding possible states using predicate logic. There are several transitions within this model and we explore just a few important ones to identify the unsafe state.

1. Within any trajectory, in any state, the system must return back to the *Basal State.*

To prove: BA is active when CB/BR and MS are not active.

This is important because we do not want the system to inform the patient that he is normal when his blood glucose is low or high.

False Positive Encountered: We shall begin with $\sim CB$. Since the system can be in only state at a time it implies that either BA or MS are active. The statement (3) from the logic when negated proves the above derivation. From statement (1) we see that CB/BR has be be inactive for the BA state to be active under all circumstances. But from statement (5) we see that if BA is inactive and if CB/BR is in any state (active or inactive) the MS state is activated. Therefore the system activates MS when CB/BR is not active and also

BA is not active leading to false information that the body is in meal supervision though it is not leading to high blood sugar in the patient.

Reachable state's : $Reach(T) = \{\ (\sim\ CB/BR \wedge MS)$,$((\sim\ BA \vee BA) \wedge\ \sim CB/BR)\ \}$

2. Within any trajectory and any state, the increase of *Risk* should increase the chances of injecting a correction insulin dose to the patient.

To prove: CB/BR is active when BA and MS are not active.

In the context of the device, *Time* is associated to *Risk*. When there is increase in elapse of time the Risk also increases as it is directly proportional.

False Positive Encountered: When we begin with statement (3), we understand that the system can only be in the correction state when both the other states are inactive. But when the system moves to meal supervision, and if in an edge case scenario the blood glucose is normal, the system might still transition to CB/BR based on statement (5).

Reachable state's : $Reach(T) = \{\ (\sim BA \wedge \sim MS)$, $(CB/BR \wedge\ (\sim CB/BR \wedge MS))$, $((\sim BA \vee BA) \wedge \sim CB/BR)\ \}$

3. Within any trajectory and any state, after an increase in *Wait* time the system will automatically move to Correction Bolus.

To Prove: In the context of the system, the "wait" time is a periodical mandatory check to manage high BG levels incurred due to forgotten meal consumption input by the patient.

False Positive Encountered: If in an edge case scenario the patient's body maintains normal blood glucose levels the system will still transition to the Correction Bolus/ Breaking Mode by which the patient may have dangerous levels of *Hypoglycemia* .

Reachable state's : $Reach(T) = \{\ (\sim CB/BR \wedge\ \sim BA)$, $((\sim BA \vee BA) \wedge\ \sim CB/BR)\ \}$

4.4 Safety Rules of the Device

A safety property (informally) states that "nothing bad will ever happen with the system"[1]. These properties search the state space defined by the equivalent FSM and identify the unsafe states. This helps build a robust and safe environment with minimum faults. Temporal logic can be used to express such queries:

- necessarily (Ω)
- possibly (Δ)
- next (Θ)

It is possible to express if a desired property is valid for a whole model or in a part of it. In the example mentioned in the working logic's #3 rule, we see that the system reaches a false positive after an edge case which is a kind of *deadlock* and performs an activity which could lead to dangerous levels of Hypoglycemia within the patient. Similarly in the #2 we see that *Risk* can indicate the forthcoming of a BG level spike /dip in the patients body which needs to be taken care of immediately. However, there is no guarantee that that state will occur after the increase of *Risk*. Finally in #1 we see that the next state should always be the Basal State. We can represent these properties as follows in terms of safety rules:

1.$\Omega(BG.level < 180 \&\& BG.level > 20)$, expresses that "the patient has normal blood glucose levels and no insulin in needed" which means that the device will never inject insulin into the body of the patient if there are normal blood glucose levels avoiding the chance to harm the patient.

The translation of the temporal formula is that the requirement expressed with that formula holds in the model if in every state of the state space the following is true; the Blood Glucose remains in $(BG.level < 180 \&\& BG.level > 20)$ until the sensor indicates a spike or dip in the latest value.
The property $p \Leftrightarrow (BG.level < 180 \&\& BG.level > 20)$, *is true unless sensor indicates otherwise.*

2.$\Delta((G + 60) > 180 || (G + 60) < 20)$, expresses that "the patient may have glucose levels that spike or dip occasionally in between regular dosages", which means the patient may suddenly have an abnormality in blood glucose levels in the body between regular dosages and these unexpected events have to be addressed.

The translation of the temporal formula is that the requirement expresses with that formula holds in the model if in every state of the state space the following is true; the states must be ready to transition to CB/BR state in the possibility of an unexpected abnormality.
The property $p \Leftrightarrow \Delta((G + 60) > 180 || (G + 60) < 20)$, *is true unless sensor indicates otherwise.*

3.$\Theta(\sim CB/BR \&\& \sim MS)$, expresses that "after any transition within all states the final state has to be the Basal State", which means the patients body's goal state is to be in Basal State for majority of the time.

The translation of the temporal formula is that the requirement expresses with

that formula holds in the model if in every state of the state space the following is true; every transition must end with the Basal State as the final state.
The property $p \Leftrightarrow \Theta(\sim CB/BR \&\& \sim MS)$, *is true at the end of every transition.*

4.5 Analysis

After taking into account the reachable states and the unsafe states within the system we can derive the following Computational Tree, which depicts the flow of the system. The tree is infinite since the system is closed loop.

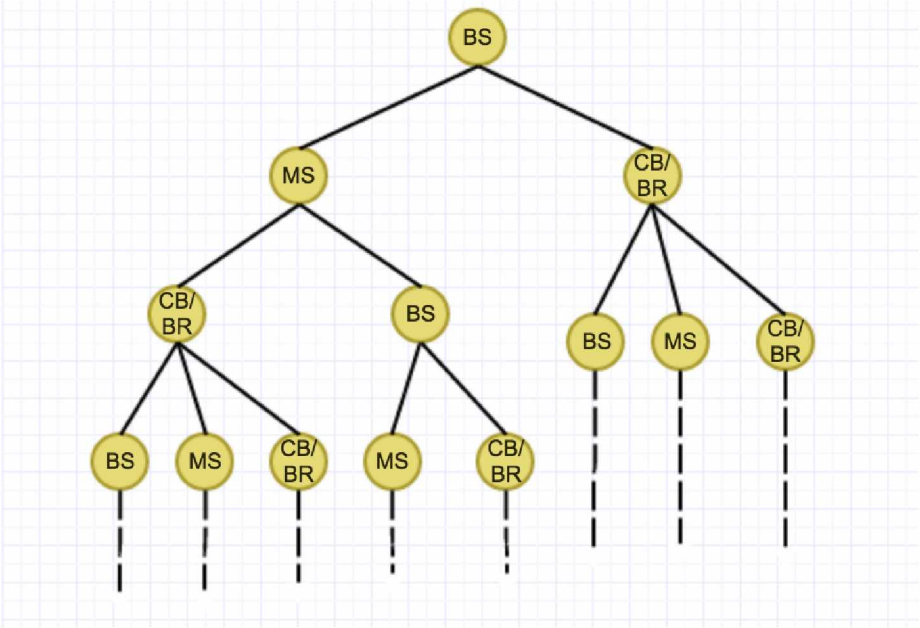


Figure 4.2: Computational Tree of the Artificial Pancreas that depicts the flow of states in an infinite loop.

Chapter 5

Performance Testing using Simulation

5.1 Software Testing

Software Testing[10] can be used if the system satisfies the requirements and performs functions for which it is intended and meets user needs. A good software development methodology will have *testing* incorporated from the beginning of the software development life cycle.

Software testing is a high level activity and ensures product integrates correctly into the environment. A product may pass with false positives in the environment through verification but it will fail when validated on the paper against *Proof of Logic.* This section deals with the conversion of the Predicate Logic and Safety rules into a simulated application for the performance testing of the system. This will confirm the loose ends identified in the reachability analysis that lead to incorrect behavior of the system.

5.2 System Simulation

The following section describes how the system was simulated in a step by step procedure.

Software Requirements:

Java, Eclipse-neon and Oracle MySQL-Workbench.

Architecture:

The overall structure of the system is designed as "Model Based Design"(from chapter.1). It simulates the hybrid automata as described by the author of the

previous paper. All the components are loosely coupled, which means they are independent entities by themselves. This structure helps in clearly understanding the role of each entity. The "controller" that has a "helper" package which is the crux of the system containing the logic. The "sensor" is connected to a database that simulates a human data input into the system's controller which helps to make decisions based on the data. The "pump" injects the insulin dosage based on the decision made by the controller.

Database:

The database used to input the system with values of patient BG levels is from the **University of California, Irvine "Diabetes Data Set"[12].**
Diabetes patient records were obtained from two sources: an automatic electronic recording device and paper records. The automatic device had an internal clock to timestamp events, whereas the paper records only provided "logical time" slots (breakfast, lunch, dinner, bedtime). For paper records, fixed times were assigned to breakfast (08:00), lunch (12:00), dinner (18:00), and bedtime (22:00). Thus paper records have fictitious uniform recording times whereas electronic records have more realistic time stamps.

Diabetes files consist of four fields per record. Each field is separated by a tab and each record is separated by a newline.

File Names and format:

1. Date in MM-DD-YYYY format
2. Time in XX:YY format
3. Code
4. Value

Code field is deciphered as follows:

- 33 = Regular insulin dose
- 48 = Unspecified blood glucose measurement
- 57 = Unspecified blood glucose measurement
- 58 = Pre-breakfast blood glucose measurement
- 59 = Post-breakfast blood glucose measurement
- 60 = Pre-lunch blood glucose measurement
- 61 = Post-lunch blood glucose measurement
- 62 = Pre-supper blood glucose measurement

- 63 = Post-supper blood glucose measurement
- 64 = Pre-snack blood glucose measurement
- 65 = Hypoglycemic symptoms
- 66 = Typical meal ingestion
- 67 = More-than-usual meal ingestion
- 68 = Less-than-usual meal ingestion
- 69 = Typical exercise activity
- 70 = More-than-usual exercise activity
- 71 = Less-than-usual exercise activity
- 72 = Unspecified special event

Analysis on the Database:

The database reflects the typical type 1 diabetes sugar level fluctuation in the patients. This data set fits perfectly with the system structure to produce desired results. The following diagram represents the data-sets random BG levels at different times, which makes it difficult to predict the patients BG levels behavior. Our Artificial Pancreas system goal is to overcome prediction difficulties and handle unexpected events in a robust way. The x-axis is the code field and the y-axis the insulin level at the code field.

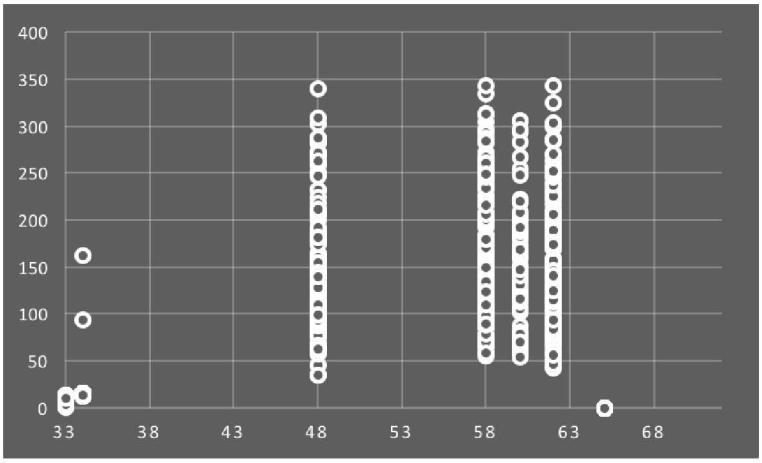


Figure 5.1: BG levels analysis of the database

From the graph we can see that the common discrepancy in BG levels is unspecified/sudden around meal times and very rarely during exercise activity.

```
public static void main(String[] args)
{
    Device_Start();
}

private static void Device_Start()
{
 data = new dataset_read();
 check = new check();
 data.start();
 check.start();
}
```

Figure 5.2: The logic of the system reads each data set along side with parallel checks which decide which state the body is in based on the data set entry.

Expected Results:

Since the system has a couple of unsafe states that lead to Hypo/Hyperglycemia in patients (stated by the previous paper) the data should result in BG levels never reaching normal values.

5.3 Running the system

The control logic uses threads to simulate time guards on the system and to simulate parallel processing of the controller with the sensor and pump. The database is connected to the controller through the sensor to generate results that simulate the patients body response to the insulin dosage to the body. The results try to mimic the real device by generating reports of controller decisions.

5.4 Analysis of Results

The states in the hybrid FSM transition according to the working and safety rules mentioned in the previous chapters generates a result set that shows the patients BG levels while the device is managing it. The results reveal that the patient receives the regular insulin dosage "33" (shown on the graph) but it transitions to wrong states unexpectedly "48" (shown in the graph) which is unspecified BG level and "58-63" which are the meal supervision states. These unexpected states are in accordance to unsafe states recorded in chapter 4.

```
if (hyper.result()== true)
{
    state = CurrentState.braking_correction_state;
    basal.Result();
}

if (hypo.result() ==true)
{
    state = CurrentState.braking_correction_state;
    bolus.Result();
}

if(meal.result() == true)
{
    state =  CurrentState.meal_supervision_state;

    bolus.getClass();
}

if(normal.result() == true)
{
    state = CurrentState.basal_state;
    System.out.println("Blood Glucose is Normal");
}
```

Figure 5.3: Based on the results from the system check, the logic decides the amount of insulin to be administered.

```
Blood Glucose is:  4  Code is:  33
Blood Glucose is:  129  Code is:  62
Meal Supervision Activated
Blood Glucose is Normal
Blood Glucose is:  340  Code is:  48
Blood Glucose High
Basal Insulin Infused
Blood Glucose is:  5  Code is:  33
Blood Glucose is Low
Bolus Insulin Infused
Blood Glucose is:  67  Code is:  58
Meal Supervision Activated
Blood Glucose is Normal
Blood Glucose is:  9  Code is:  33
Blood Glucose is Low
Bolus Insulin Infused
Blood Glucose is:  14  Code is:  34
Blood Glucose is Low
Bolus Insulin Infused
Blood Glucose is:  4  Code is:  33
Blood Glucose is:  206  Code is:  62
Blood Glucose High
Basal Insulin Infused
Meal Supervision Activated
```

Figure 5.4: The results of the application look as above.

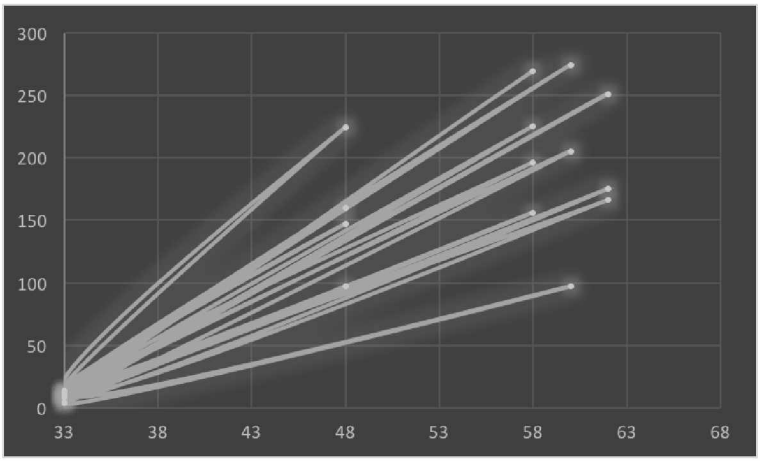


Figure 5.5: BG levels final analysis diverging from normal levels

5.5 System Under Test(SUT)

After we have all the requirements confirmed and aligned to the design of the system, SUT[8] checks whether the system satisfies the specification.

SUT acts a *blackbox* in the sense that the internals are not cared about. Instead, we interact with the SUT by means of inputs and outputs i.e., we provide certain inputs and observe the outputs. A test case implemented on the the SUT would be a description of the flow of control in various circumstances. If the behavior of the test case is expected the test is titled PASSED; however, if the behavior is not expected the test case is titled FAILS. If the verdict of the test is a FAIL the meaning would imply that the SUT does not meet its specifications.

In this context, the SUT would be the simulated Artificial Pancreas system implemented in the previous sections. The test case generation is preferred to be automatic because automatically synthesized tests would contain minimum human errors. The goal would be to generate minimum test cases with maximum coverage of the specifications.

5.5.1 Test Generation

Tests are preferably generated by restricting the behaviors of the environment to yield a deterministic testing automaton[8]. A test suite can therefore be a finite set of executions of the environment automaton. In the previous chapters, the main goal was to isolate a scenario which yields to a set of bad and/or unsafe states of the system. This section will help build test cases to steer towards

these bad states using the RRT (Rapidly Exploring Random Tree) algorithm and coverage guided strategy. These test cases will expect to FAIL since the SUT was built with know defects.

Rapidly Exploring Random Trees

The RRT algorithm[13] begins from a root node and incrementally grows a tree until the tree reaches the goal configuration. To grow the tree,

1. Pick a location (configuration), q_r, (with some sampling strategy)
2. Find the vertex in the search tree closest to that random point, q_n
3. Try to add an edge (path) in the tree between q_n and q_r, if you can link them without a collision occurring.

The process is repeated until the tree grows to within some user-defined threshold of the goal.

1. $state.init(q_o)$;
2. for $i = 1$ to k do
3. $state.add.vertex(\alpha(i))$;
4. $q_n \leftarrow NEAREST(S(state), \alpha(i))$;
5. $state.add.edge(q_n, \alpha(i))$;

The most important property of the RRT in this scenario is understanding that there are several paths to the goal state from any state and it can be infinitely iterate. This property aligns with the fact that the Artificial Pancreas should constantly monitor the human body. The following diagram describes the three states the body can be in , i.e., normal BG, high BG and low BG. When random

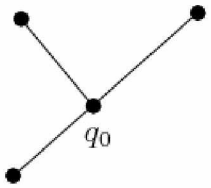


Figure 5.6: Initial state of the tree

states are added to the above tree the infinite loop of the system can be depicted by never ending expansions of the RRT as below.

A typical branch from the RRT of the artificial pancreas will look as follows:

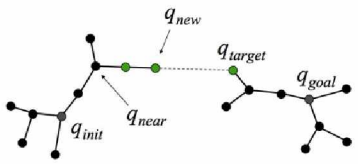


Figure 5.7: Graphical demonstration of RRT

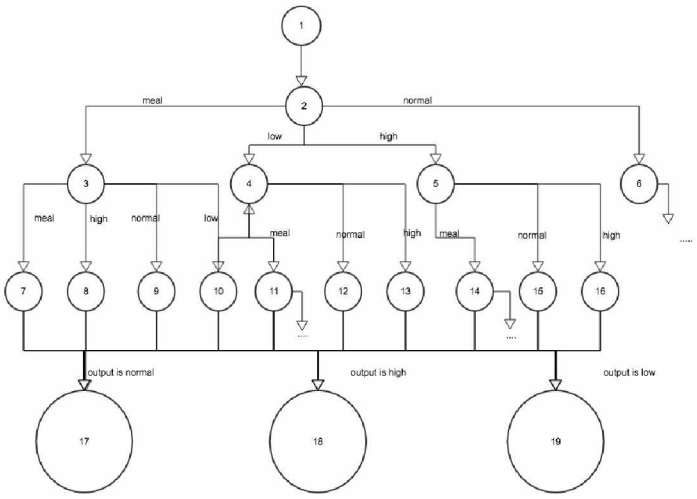


Figure 5.8: RRT an Artificial Pancreas in a single iteration

Pseudo Test Cases

The FAIL destination nodes are deemed as the bad states in the system. They can arise from any state. Since test coverage should be maximum the test cases should be deterministic. The coverage should include all possible inputs and what the SUT should generate depending on the input i.e., the output.

In the case of the artificial pancreas any state should lead to the PASSING state i.e., given any starting state the system should expect the body to return to normal state after the state completes. Therefore, even if a state is continuously occurring it is standard to continue rectification till the goal state is attained.

// test case pseudo-code: s := initialize state;
// this is the state of the tester
while(not some termination condition) do

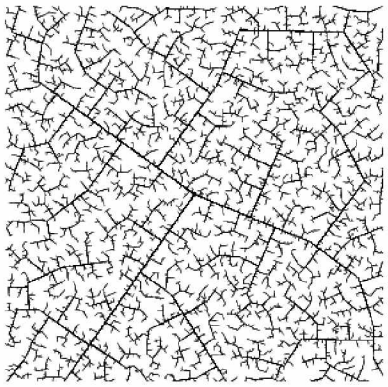


Figure 5.9: Dense RRT after 1000 iterations

x := select input in set of legal inputs given s;
issue x to the SUT;
set timer to TIMEOUT;
wait until timer expires or SUT produces an output;
if (timer expired) then
s := update state s given TIMEOUT;
end if;
if (SUT produced output y, T time units after x) then
s := update state s given T and y;
end if;
if (s is not a legal state) then
announce that the SUT failed the test and exit;
end if;
end while;
announce that the SUT passed the test and exit;

Ideally, test cases should include all possible failure scenarios from every state. They should also include the best case working scenario, where all the states work ideally. So we write a test case with expectations and requirements embedded in it initially. Initially it will fail as long as the system is under development. After everything is properly set the test will pass. The graphical representation of the test case for the Artificial Pancreas can be shown as below

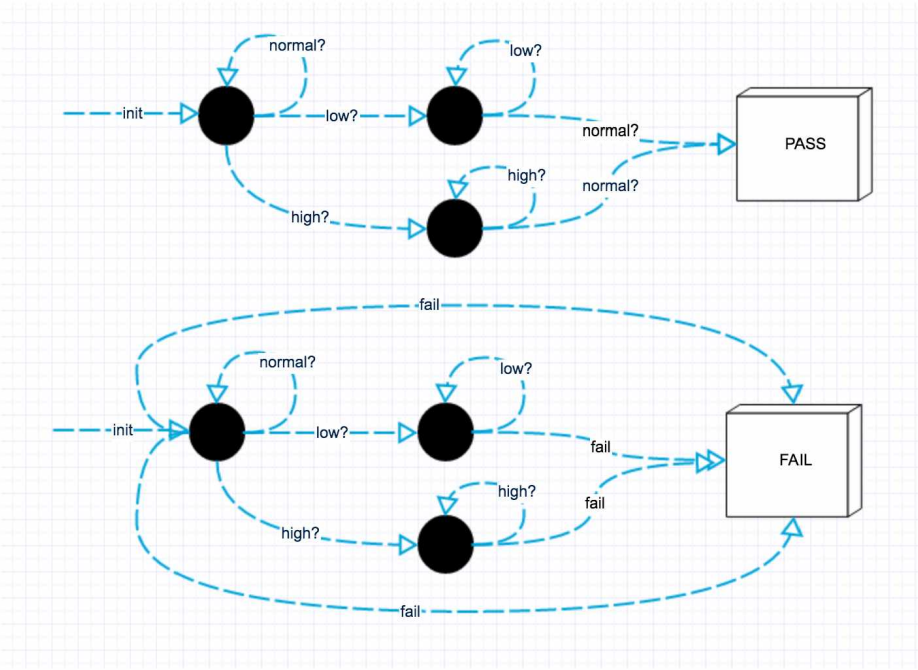


Figure 5.10: Graphical Pass/Fail test scenarios with ideal passing test cases and failure test cases.

Chapter 6

Future Work and Conclusion

6.1 Possible Solution to Failures

6.1.1 Monitors

In order to formalize safety requirements it is often very convenient to use a *monitor automaton*, also often called an *observer*, that watches the trajectories of the system and enters the "BAD" locations when one trajectory violates a given safety property. Safety verification is then reduced to deciding the reachability of a set of "Bad" locations.

A monitor automaton can be defined as temporal logic formula which is a finite automaton. It accepts exactly the behaviors that satisfy the safety properties.

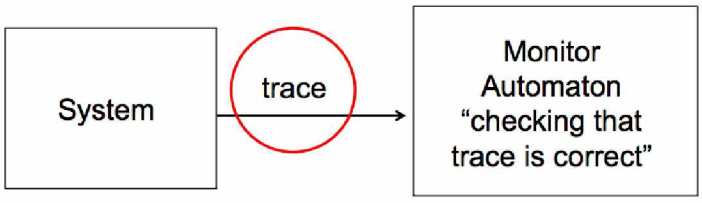


Figure 6.1: Monitor Automaton

A trace is a sequence of the observable parts of the state, in our case they can be called the reachable states path. Following figure shows the monitor for

the safety requirement (1), (2) and (3) respectively. These safety requirements are the working rules defined in chapter 4. The monitor observes the value of the risk and the patients BG levels whose dynamics is defined by the Artificial Pancreas automaton. BAD STATES will be defined as edge case scenarios that cause unexpected behavior deadlocks or wrong decisions. Similarly, from the reachability analysis we observed that sometimes the state transitions to CB/BR even though the patient has normal BG levels. Thus to verify, we must establish that no state in which the control of the monitor is in the BAD location is reachable in $[CONTROLLER \bigotimes PUMP \bigotimes SENSOR \bigotimes MONITOR]$. In that case, we know that the controller ensures the requirement.

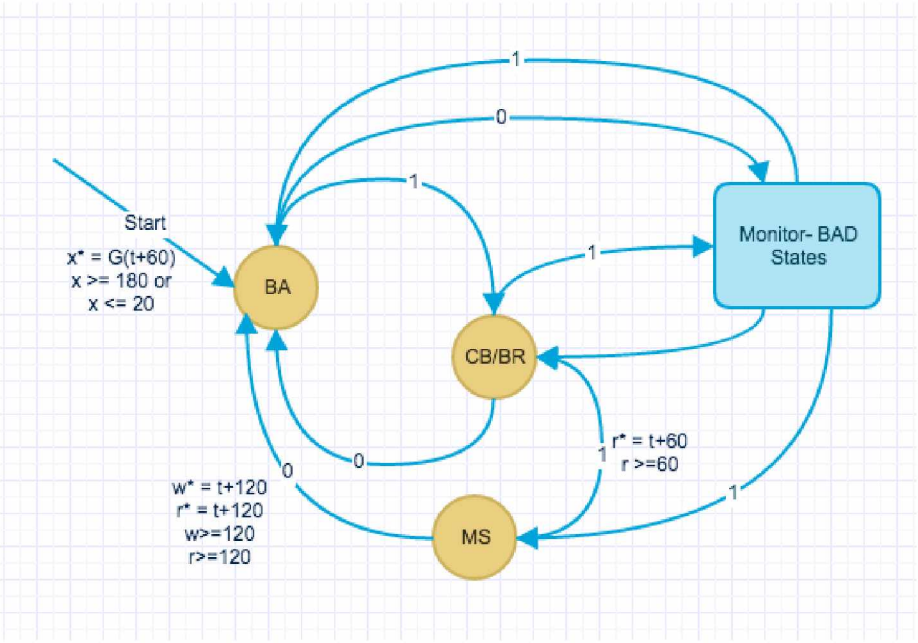


Figure 6.2: FSM with Monitor Automaton

Thus by adding a monitor state to the FSM we may catch the document and catch "unsafe" states and make sure our system is faultless.

6.2 Conclusion

We have studied and presented a methodology for design and model checking of medical devices structured as Hybrid Automaton and Timed Petri Nets. The methodology is based on isolating "unsafe states" in the automaton using reachability analysis. We confirmed that the states as "unsafe" by producing

live results while simulating the faulty device. We also proposed to include a "monitor" state that will handle the unsafe states in the edge case scenario. Future work may include implementing the monitor to observe results and using this strategy to build a new device with more complex requirements.

In such systems the obtained data from the human body are uncertain, and unpredictable values should be supported. This means that the model used for such purpose should have capability of reasoning based on uncertain information. This research introduced a new model based off not just rule-based hybrid automata but also fuzzy incorporated timed petri net. We used TPN's to model the system knowledge and rules, and we demonstrated that the system hazards can be derived easily by the model.

This method may be one of the strategies to address false positive faulty designs like the one mentioned by the related paper's author. Future work can include implementing the monitor or integrating multiple monitors to increase the robustness of the system.

Appendix A

Bibliography

[1] Eleftherakis, G., & Kefalas, P. (2001). Towards model checking of finite state machines extended with memory through refinement. *Advances in signal processing and compu ter technologies*, 321-326.

[2] Banerjee, A., Zhang, Y., Jones, P., & Gupta, S. K. Using Formal Methods to Improve Safety of Home-Use Medical Devices.

[3] Cobelli, C., Renard, E., & Kovatchev, B. (2011). Artificial pancreas: past, pre sent, future. *Diabetes*, *60*(11), 2672-2682.

[4] Eleftherakis, G., & Kefalas, P. (2001). Towards model checking of finite state machines extended with memory through refinement. *Advances in signal processing and computer technologies*, 321-326.

[5] Automata theory. (n.d.). Retrieved October 08, 2016, from https://en.wikipedia.org/wiki/Automata_theory

[6] Jiang, Z., Pajic, M., Alur, R., & Mangharam, R. (2014). Closed-loop verification of medical devices with model abstraction and refinement. *International Journal on Software Tools for Technology Transfer*, *16*(2), 191-213.

[7] Althoff, M. (2010). Reachability analysis and its application to the safety assessment of autonomous car s. *Technische Universitt Mnchen*.

[8] Tripakis, S., & Dang, T. (2009). Modeling, verification and testing using timed and hybrid automata. *Model-Bas ed Design for Embedded Systems*, 383-436.

[9] Predicate Logic. (n.d.). Retrieved from http://csitiub.pbworks.com/w/file/fetch/72718706/Notes

[10] Seshia, S. A. Introduction to Embedded Systems.

[11] ISTQB Exam Certification. (n.d.). Retrieved October 08, 2016, from http://istqbexamcertification.com/what-is-validation-in-software-testing-or-what-is-software-validation /

[12] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[13] Ferguson, D., Kalra, N., & Stentz, A. (2006, May). Replanning with rrts. In *Proceedings 2006 IEEE International Conf erence on Robotics and Automation, 2006. ICRA 2006.* (pp. 1243-1248). IEEE.

[14] Finite Automata. (n.d.). Retrieved October 08, 2016, from https://www.cs.rochester.edu/~nelson/courses/csc_173/fa/fa.html

[15] Bengtsson, J., & Yi, W. (2004). Timed automata: Semantics, algorithms and tools. In *Lectures on concurrency and petri nets* (pp. 87-124). Springer Berlin Heidelberg.

[16] First Order Logic. (n.d.). Retrieved from http://www.cs.cornell.edu/courses/cs4700/2011fa/lectures/16_firstorderlogic.pdf

[17] Raskin, J. (n.d.). An Introduction to Hybrid Automata. Retrieved from http://www.cmi.ac.in/~madhavan/courses/qath-2015/reading/Raskin_Intro_Hybrid_Automata.pdf

[18] What is the pancreas? What is an artificial pancreas device system? (n.d.). Retrieved October 08, 2016, from http://www.fda.gov/MedicalDevices/ProductsandMedicalProcedures/HomeHealthandConsumer/ConsumerProducts/ArtificialPancreas/ucm259548.htm

[19] Written by Allison Blass | Published on January 30, 2012. (2012). Tom Brobson: My Experience in the Artificial Pancreas Clinical Trials. Retrieved October 08, 2016, from http://www.healthline.com/diabetesmine/tom-brobson-my-experience-in-the-artificial-pancreas-clinic al-trials

[20] Software Verification and Validation. (2012). *Reliable Design of Medical Devices, Third Edition,* 401-410. doi:10.1201/b12511-35

[21] Introduction to predicate logic. (n.d.). doi:10.1075/ps.5.3.02chi.audio.2f

[22] Predicate Logic. (n.d.). Retrieved from http://infolab.stanford.edu/~ullman/focs/ch14.pdf

[23] UML 2 Use Case Diagrams: An Agile Introduction. (n.d.). Retrieved October 08, 2016, from http://agilemodeling.com/artifacts/useCaseDiagram.htm

[24] Dotoli, Mariagrazia, and Maria Pia Fanti. "An urban traffic network model via coloured timed Petri nets." *Control Engineering Practice* 14.10 (2006): 1213-1229.

[25] Petri Nets. (n.d). Retrieved October 30,2016 from https://en.wikipedia.org/wiki/Petri_net

[26] Chao, Crystal, and A. Thomaz. "Timed petri nets for multimodal interaction modeling." *ICMI 2012 Workshop on Speech and Gesture Production in Virtually and Physically Embodied Conversational Agents*. 2012.

[27] Zuberek, W. M. "Throughput analysis of simple closed timed Petri net models." *Circuits and Systems, 1993., Proceedings of the 36th Midwest Symposium on*. IEEE, 1993.

[28] Huang, Yi-Sheng, and Ta-Hsiang Chung. "Modeling and Analysis of Urban Traffic Lights Control Systems Using Timed CP-nets." *J. Inf. Sci. Eng.* 24.3 (2008): 875-890.