Department of Computer Science & Engineering
The University of Texas at Arlington

# Detailed Design Specification
# CSE 4317: Senior Design II
# Spring 2024


## Sim Breathe Repeat
## Breathe Easy Sim

David Gomez
Elisabeth Harris
Kyle Henry
Ethan Sprinkle
Maicol Zayas

## Revision History

| Revision | Date | Author(s) | Description |
|---|---|---|---|
| 1.0 | 2.12.2024 | DG, EH, KH, ES, MZ | document creation |
| 1.1 | 4.28.2024 | ES | updated document to reflect final project turn in |

# CONTENTS

## List of Figures



## List of Tables

# 1 Introduction

The Breathe Easy Sim is designed to provide a solution for the UTA SMART Hospital's need for a medical ventilator simulator that is compatible with the Hospital's smart manikins. Currently, the ventilators being used by the Hospital provide too much airflow to the smart manikins, causing the internal air bladders to rupture. The Breath Easy Sim will provide reduced and controlled airflow capability so as not to rupture the manikin's air bladder. Additionally, the Breathe Easy Sim will be controlled using customize simulation software in order to provide as close an experience as possible to a real medical environment. The simulation environment is further enhanced through the inclusion of other ventilator subsystems such as alarm tones, a touchscreen user interface with associated buttons, and simulated health diagnostics.

# 2 System Overview

The system consists of four layers: the Air Pump Layer, the Peripherals Layer, the User Software Layer, and the Processing Layer. The architecture of the system is designed to display a clear separation between the hardware and software processes interacting with each other. Two of the four layers, Air Pump and Peripherals, are composed entirely of hardware components. The Air Pump layer controls the airflow being sent to the manikin using a customized air pump capable of creating small amounts of air pressure that are unable to rupture the bladders. For the Peripherals Layer, it functions as the user's main input to the system, which includes: push buttons, a rotary encoder, and a touch screen.

The other half of the diagram illustrates the software and how it connects to the previous layers. As the name describes, the User Software Layer represents the software part of the system. This software consists of a Graphical User Interface (GUI) that allows the user to operate all the functionalities of the system. Lastly, the Processing Layer represents the interconnection between the hardware and software of the system. The main purpose of this layer is to translate and transform input and output signals between the physical components that the user utilizes, and the software program used to operate the system.

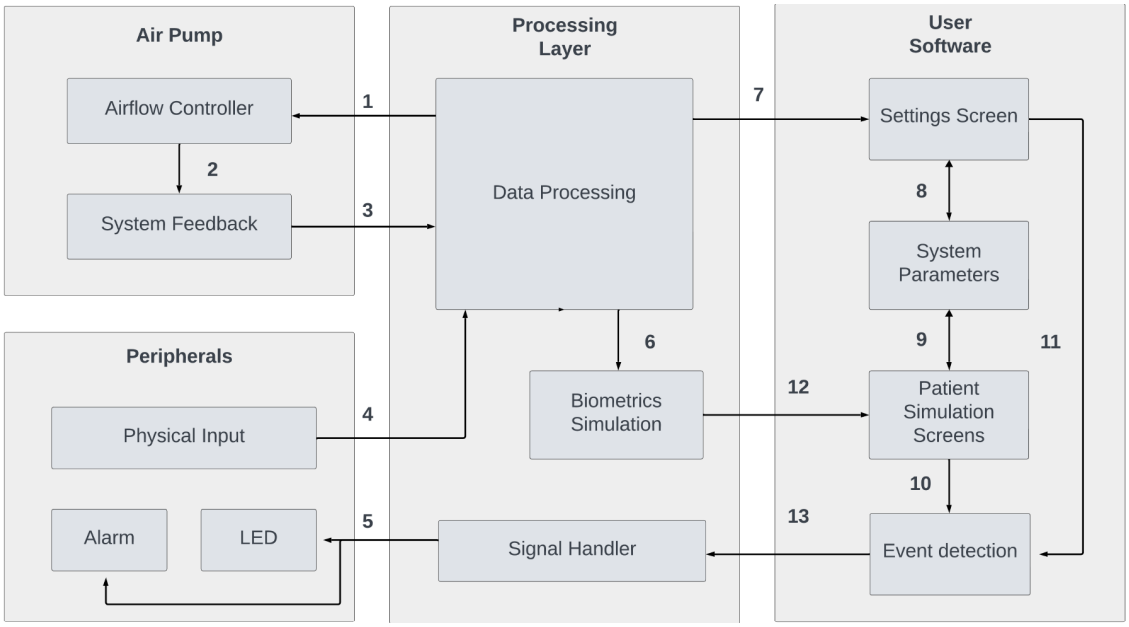


Figure 1: System architecture

# 3 Air Pump Layer Subsystems

The Air Pump Layer is responsible for providing air flow through the use of a singular bellows pump system controlled by stepper motors.

## 3.1 Layer Hardware

The 2 stepper motors that control the position of the bellows are two STEPPERONLINE brand Nema 17 Stepper Motor 2A 59Ncm driven by a DM542 CNC Digital Microstep Driver. A limit switch is used to calibrate the position of the air pump, accounting for sudden loss of power during operation.

## 3.2 Layer Software Dependencies

Inputs to the stepper motors will be generated by an embedded EK-TM4C123GXL redboard in the Data Processing Layer.

## 3.3 Airflow Controller Subsystem

The Airflow Controller subsystem is comprised of a 2-stepper motor controlled bellows system to provide positive and negative air pressure to the manikin. Compressing the bellows results in positive air pressure (breathing in) while decompressing the bellows results in negative air pressure (breathing out). A removable cap on the pump can be removed in the event that negative air pressure is no longer required.

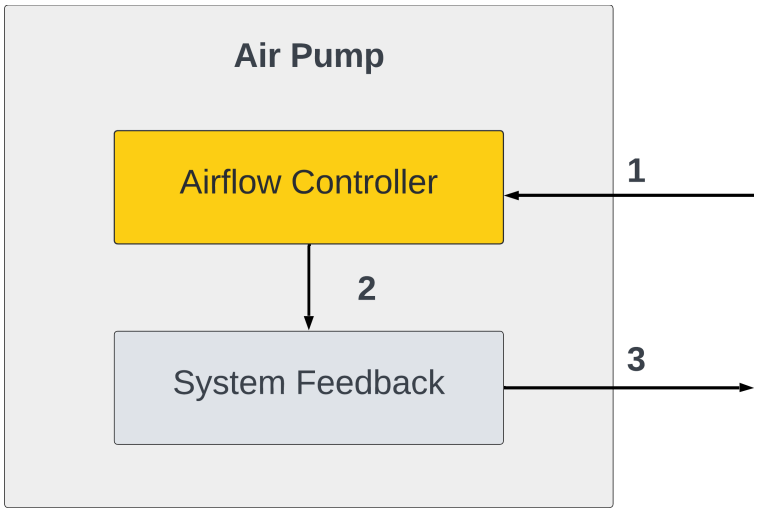


Figure 2: Subsystem description diagram of the Airflow Controller subsystem

### 3.3.1 Subsystem Hardware

The frame of the bellows, along with the gears and rail system that, are constructed from 3D printed PLA material while the "body" of the bellows are made of leather.

## 3.4 System Feedback Subsystem

A limit switch is used to prevent the bellows from over and under extending past its intended design and reports it to the Data Processing subsystem.

### 3.4.1 Subsystem Programming Languages

The embedded EK-TM4C123GXL redboard that interfaces with the motors will be programmed using the C programming language.

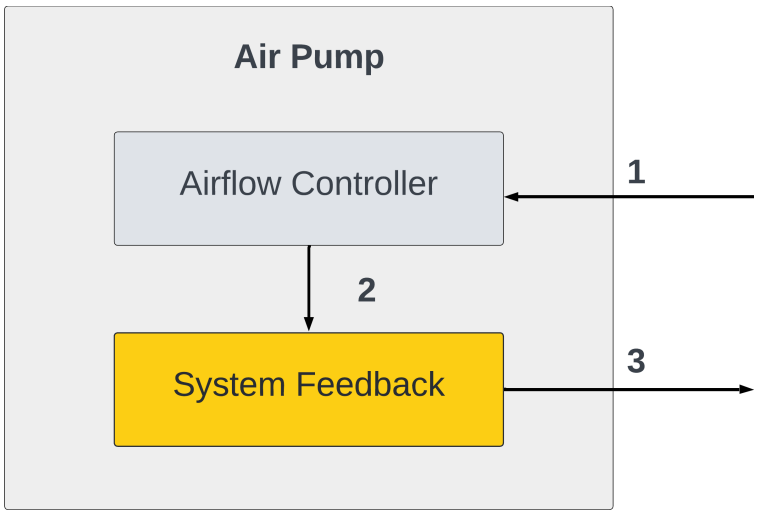


Figure 3: Subsystem description diagram of the System Feedback subsystem

### 3.4.2 Subsystem Data Structures

Data between the redboard and stepper motors and microcontroller will be handled using I2C communication protocol.

# 4 Peripherals Layer Subsystems

The Peripherals Layer is responsible for providing the user the means of interacting with the system through multiple hardware components. These interactions include taking input from the user and sending output back to the user as information.

## 4.1 Layer Hardware

All peripherals are controlled by the Raspberry Pi in the Data Processing Layer.

## 4.2 Layer Software Dependencies

All components will be interpreted by the Raspberry Pi in the Data Processing Layer.

## 4.3 Physical Input Subsystem

This subsystem is composed only of hardware input peripherals, including push buttons, an encoder, and a touchscreen. The Physical Input subsystem is in charge of interfacing with the user and obtaining all the desired input. This subsystem has great importance for the overall usage of the system since it is the only channel for receiving external input.

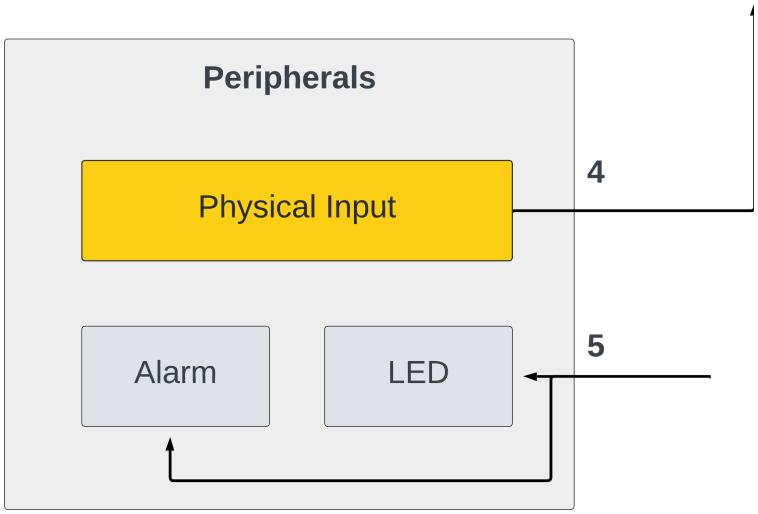


Figure 4: Subsystem description diagram of the System Feedback subsystem

### 4.3.1 Subsystem Hardware

This subsystem contains 6 push buttons that will be connected to the Raspberry Pi and allow the user to navigate through multiple different screens on the GUI. Another component found in this subsystem is a rotary encoder that grants the user the capacity to modify all parameters with precision and ease. The encoder is connected to the Raspberry Pi and interpreted using a Java library. The other peripheral associated with this subsystem is the touch screen of the system which is directly connected to the Pi.

### 4.3.2 Subsystem Programming Languages

The software is programmed using Python.

### 4.3.3 Subsystem Data Structures

All input data recorded by this subsystem is sent to the Data Processing module inside the Processing Layer. The Data Processing Layer acquires the data, processes it, and transforms it into information for the GUI.

## 4.4 Alarm Subsystem

The Alarm subsystem handles the specific case of an alert alarm. This alarm is called in situations when the system has a malfunction or the user is to be notified regarding an issue. This subsystem is purely output and is currently turned off.

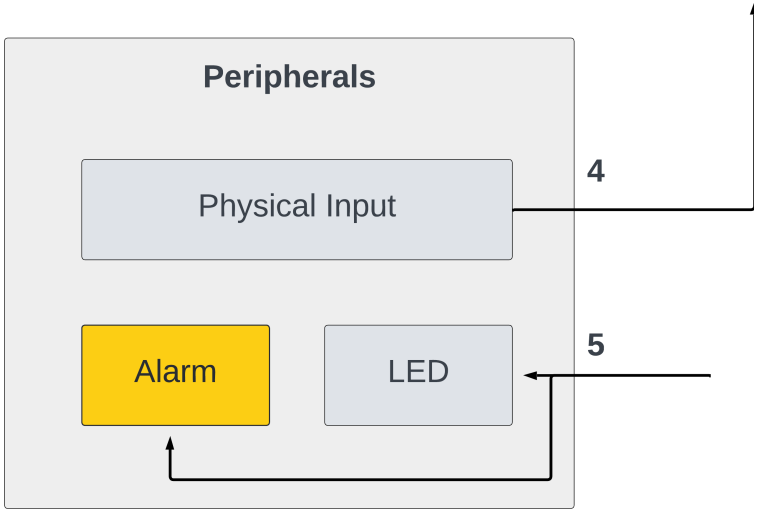


Figure 5: Subsystem description diagram of the Alarm subsystem

### 4.4.1 Subsystem Hardware

This subsystem contains two speakers attached at the back of the touchscreen and an LED in the front screen case. The speaker is the only component in this layer that is not interfaced with the Pi. Instead, it is directly connected to the touch screen as part of the audio output. The LED will be controlled through the Pi.

### 4.4.2 Subsystem Programming Languages

The software is programmed using Python.

## 4.5 LED Subsystem

The LED subsystem is used to notify the User when there is power or an active alarm. It is controlled directly through the Raspberry Pi.

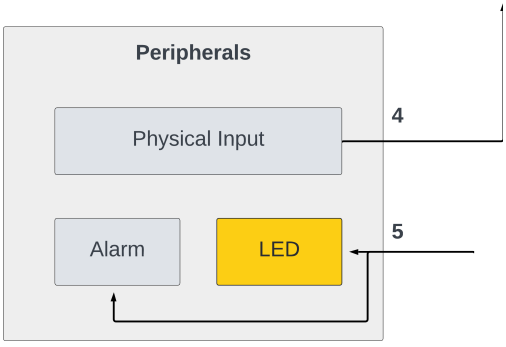


Figure 6: Subsystem description diagram of the LED subsystem

### 4.5.1 Subsystem Hardware

This subsystem contains a single LED light that can be utilized to alert the user of any problems with the system.

### 4.5.2 Subsystem Programming Languages

The Raspberry Pi software is programmed using Python.

# 5 Processing Layer Subsystems

The processing layer's main goal is to take in information from all the other layers and process the signals for interpretation.

## 5.1 Layer Hardware

This layer compromises of 2 main pieces of hardware: the Raspberry Pi 4 and the LauchPad Redboard. The Raspberry Pi 4 will process incoming and outgoing signals which communicates through the Redboard across the layers.

## 5.2 Layer Software Dependencies

To communicate between the Raspberry Pi 4 & the Redboard, we will be using the Pi4J module, which is a Java module specifically designed to communicate with devices attached to the General Purpose Input Output pins.

## 5.3 Data Processing Subsystem

The Data Processing Subsystem is the subsystem responsible for the receiving and translating the signals from the hardware subsystems.

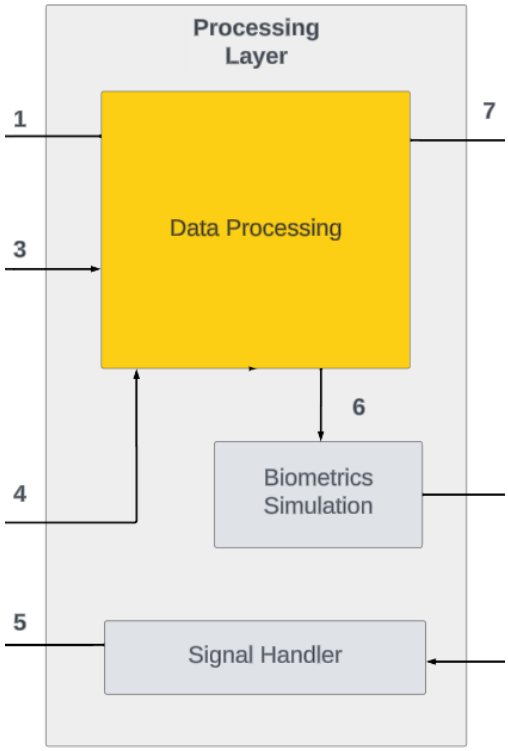


Figure 7: Subsystem description diagram of the Data Processing subsystem

### 5.3.1 Subsystem Hardware

This does not require any hardware apart from what is mentioned above.

### 5.3.2 Subsystem Operating System

This requires Raspberry Pi OS.

### 5.3.3 Subsystem Software Dependencies

This does not require any software dependencies apart from what is mentioned above.

### 5.3.4 Subsystem Programming Languages

This requires OpenJDK 11 SE.

## 5.4 Biometrics Simulation Subsystem

The Biometric Simulation subsystem is what is in charge of emulating the patient that is "connected" to the ventilator. These diagnostics are generated using math libraries and do not currently correlate to ventilator output.

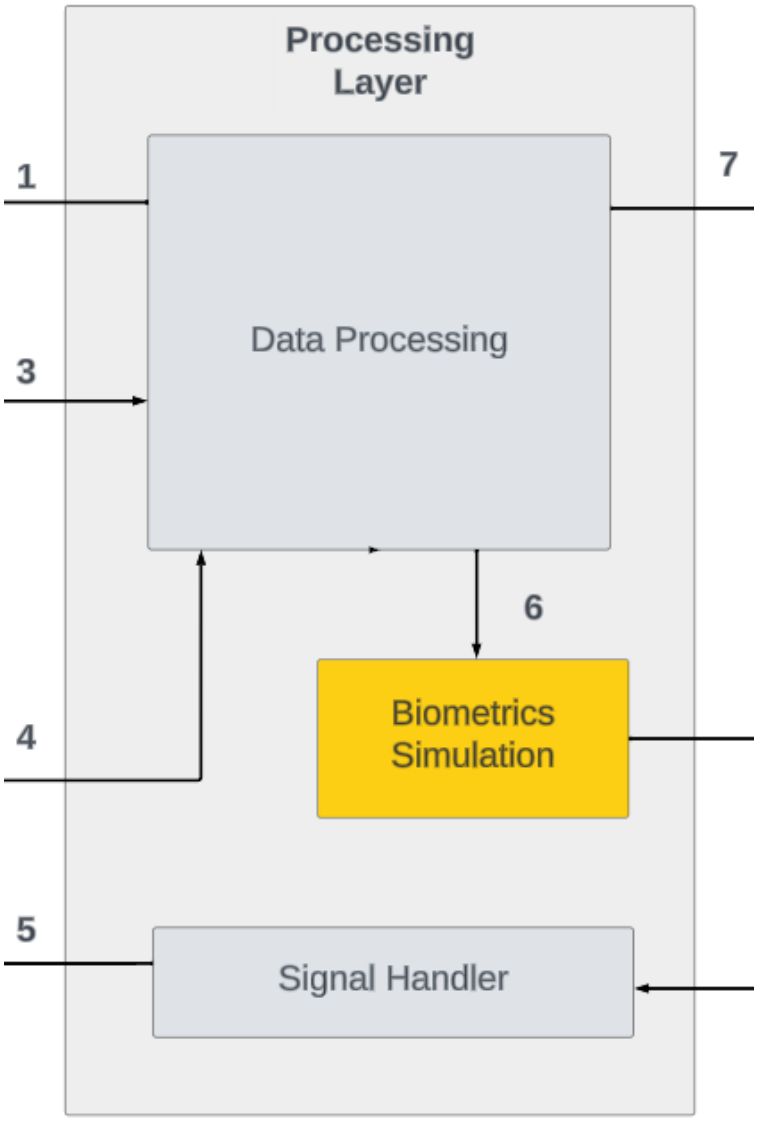


Figure 8: Subsystem description diagram of the Biometrics Simulation subsystem

### 5.4.1 Subsystem Hardware

This requires the Raspberry Pi 4.

### 5.4.2 Subsystem Operating System

This does not require any specific OS.

### 5.4.3 Subsystem Software Dependencies

This will require Java Math libraries to calculate accurate patient Biometrics.

### 5.4.4 Subsystem Programming Languages

This requires OpenJDK 11 SE.

## 5.5 Signal Handler Subsystem

The Signal Handler subsystem receives commands from the Event Detection sublayer on the Processing Layer. The signal handler transforms these commands into analog signals sent from the output peripherals including LEDs and the alarm. This sublayer represents the translation of commands from the software on the Raspberry Pi.

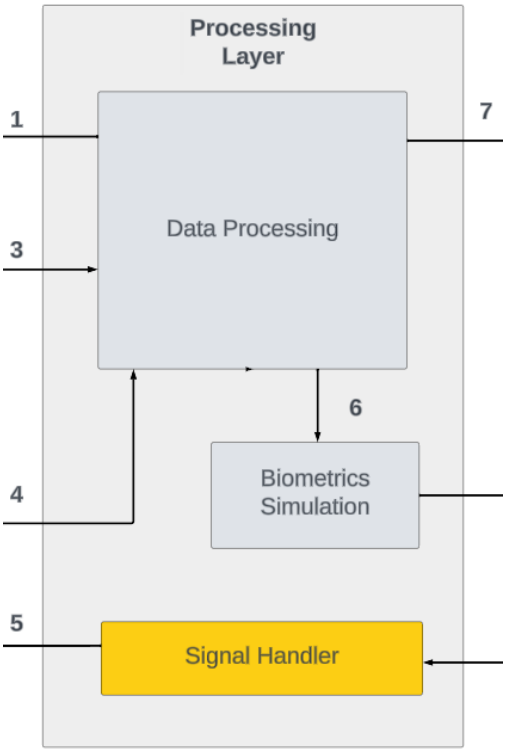


Figure 9: Subsystem description diagram of the Signal Handler subsystem

### 5.5.1 Subsystem Hardware

The processing takes place inside the Raspberry Pi.

### 5.5.2 Subsystem Operating System

Although this layer is independent of the GUI, the operating system used is the dedicated Raspberry Pi OS.

### 5.5.3 Subsystem Programming Languages

There are two programming languages in use. The GUI on the RPi uses Java and the Microcontroller uses C language.

### 5.5.4 Subsystem Data Structures

The Signal Handler receives its input from the Event Detector as a digital signal using a general-purpose input pin. An interrupt will be in charge of monitoring and waiting for input signals.

### 5.5.5 Subsystem Data Processing

There are no specific algorithms used in this subsystem.

# 6 User Software Layer Subsystems

The User Software Layer contains all the subsystems to operate the Breathe Easy Sim software. The software is modeled after the software used on the AVEA Ventilator in order for students to easily transfer their user experience. The Graphical USer Interface (GUI) is modeled and programmed using the Java Swing libraries and is designed to be controlled and operated using a touchscreen, exterior buttons, and a rotary encoder.

## 6.1 Layer Hardware

All software in this layer is run on the Raspberry Pi 4.

## 6.2 Layer Operating System

The RaspberryPi OS will be used as the primary OS.

## 6.3 Layer Software Dependencies

No additional software dependencies are used besides Java.

## 6.4 Settings Screen Subsystem

The Settings Screen subsystem is in charge of showing changing ventilator behavior that are non integral to the patient. This subsystem is responsible for reflecting the changes in the ventilator behavior based on inputs from the user and the patient.

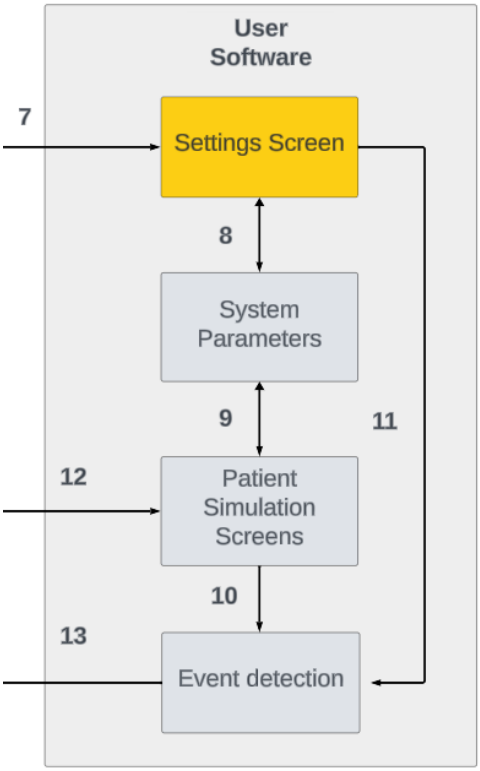


Figure 10: Subsystem description diagram of the Settings Screen subsystem

### 6.4.1 Subsystem Hardware

There will be no hardware components involved with this subsystem.

### 6.4.2 Subsystem Software Dependencies

This subsystem will use the Java Swing library for the GUI components.

### 6.4.3 Subsystem Programming Languages

The Settings Screen subsystem will be programmed using the Java programming language.

### 6.4.4 Subsystem Data Structures

As this subsystem is comprised of the GUI components, we will use Java Swing to create and manage the GUI components.

### 6.4.5 Subsystem Data Processing

This subsystem processes changes to the screen based off button presses. The information from the button presses comes from the Data Processing sublayer, which transforms the button presses into information for the GUI.

## 6.5 System Parameters Subsystem

The System Parameters subsystem is a vital component within the Simulated Ventilator (Sim Vent) architecture, residing in the Processing Layer. This subsystem is responsible for managing and maintaining critical system settings and parameters that dictate the behavior of the simulation. It serves as the central hub for configuring ventilation settings and patient simulation scenarios. It will be a class with a hash map.

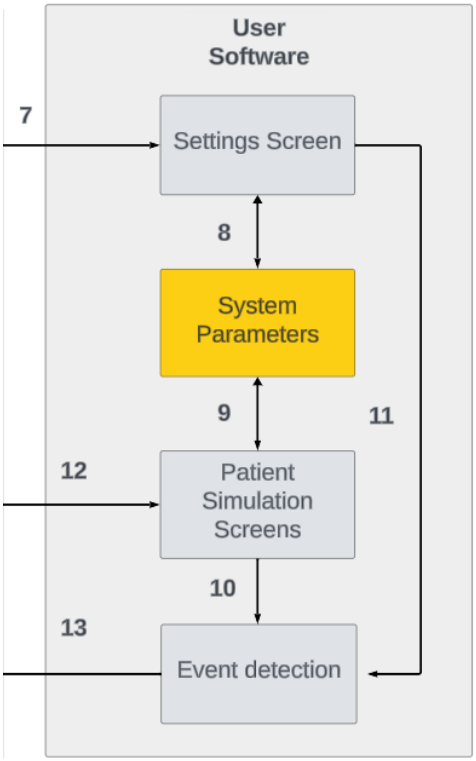


Figure 11: Subsystem description diagram of the System Parameters subsystem

### 6.5.1 Subsystem Hardware

There will be no hardware components involved with this subsystem.

### 6.5.2 Subsystem Software Dependencies

The Simulation Ventilator will be using the Properties library to read in a configuration file and placing these in a hashed map.

### 6.5.3 Subsystem Programming Languages

This subsystem will be implemented using Java.

### 6.5.4 Subsystem Data Structures

We will be using a hashed map to connect the information from the configuration file to the system.

### 6.5.5 Subsystem Data Processing

The System Parameters subsystem allows users to configure ventilation settings, including parameters such as tidal volume, respiratory rate, and inspiratory/expiratory ratios.

## 6.6 Patient Simulation Screens Subsystem

The Patient Simulation Screens subsystem is in charge of showing and changing ventilator behavior that are integral to the patient. The responsibility of the Patient Simulation Screens is to continually show simulated real-time data. It will also show changes made from the physical inputs of the touch screen and knobs through receiving input from the Biometrics Simulation subsystem.

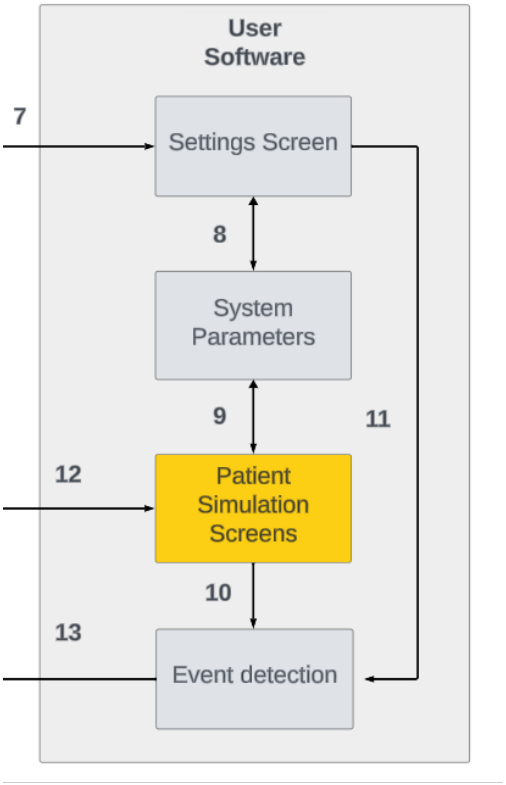


Figure 12: Subsystem description diagram of the Patient Simulation Screens subsystem

### 6.6.1 Subsystem Hardware

There will be no hardware components involved with this subsystem.

### 6.6.2 Subsystem Software Dependencies

This subsystem will use the Java Swing library for the GUI components.

### 6.6.3 Subsystem Programming Languages

The Patient Simulation Screens subsystem will be programmed using the Java programming language.

### 6.6.4 Subsystem Data Structures

As this subsystem is comprised of the GUI components,we will use Java Swing to create and manage the GUI components.

### 6.6.5 Subsystem Data Processing

This subsystem processes changes to the screen based off button presses. The information from the button presses comes from the Data Processing sublayer, which transforms the button presses into information for the GUI.

## 6.7 Event Detection Subsystem

The Event Detection Subsystem is a pivotal component within the Simulated Ventilator (Sim Vent) architecture, housed in the User Software Layer. This subsystem is responsible for monitoring the simulation in real-time, detecting critical events and alarm conditions, and initiating appropriate responses. It ensures the safety and educational effectiveness of the Sim Vent system.

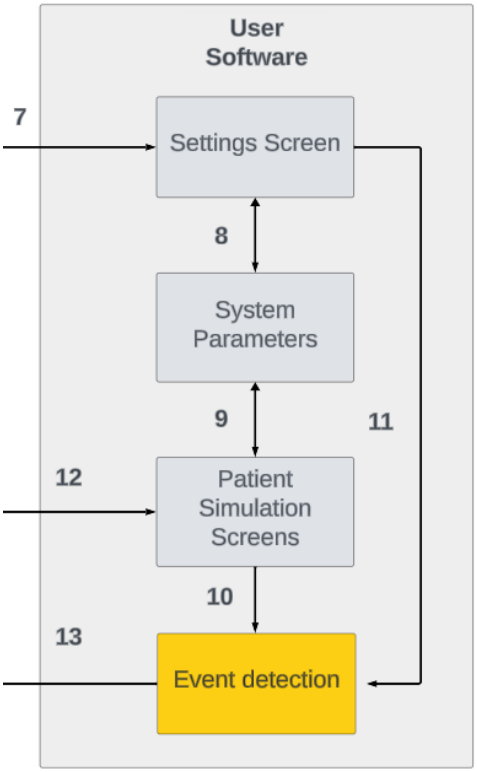


Figure 13: Subsystem description diagram of the Event Detection subsystem

### 6.7.1 Subsystem Hardware

There are no hardware components of this subsystem.

### 6.7.2 Subsystem Software Dependencies

There aren't any software dependencies for this subsystem.

### 6.7.3 Subsystem Programming Languages

We will be using Java for this subsystem.

### 6.7.4 Subsystem Data Structures

There will be an event detection class that takes input from the settings screen and the patient simulation screens.

### 6.7.5 Subsystem Data Processing

This is just a class that relays input from the settings screen and the patient simulation screens and sends changes to the signal handler to be processed.

# 7 APPENDIX A

This section intentionally left blank.

## REFERENCES