*Architecture Document for*

Smart and Connected Health Alert System (SCHAS)

Version: 1.5

Last Updated: 01/10/2014 Table of Contents

1 Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Definitions, Acronyms and Abbreviations 1

1.4 References 1

2 Architectural Goals and Constraints 1

2.1 Goals 1

2.2 Constraints 2

3 System Context 2

4 Business Process View 3

5 Use Case View 3

5.1 Main Actors 5

6 Logical Architecture View 5

6.1 Mobile device interfaces 6

6.2 Environmental Data Collection 6

6.2.1 Initial Prototype Development Board 7

6.2.2 GPS 7

6.2.3 Temperature Sensor 7

6.2.4 Humidity Sensor 8

6.2.5 Ozone Sensor 8

6.2.6 Nitrogen Dioxide Sensor 9

6.2.7 Case 10

6.2.8 Battery 10

6.2.9 Second generation Air Quality monitor 10

6.3 Inhaler cap and peak flow meter design 10

6.3.1 Circuit Design 11

6.3.2 RN-42 Bluetooth Module 11

6.3.3 Android Program 11

6.3.4 ATtiny85 11

6.3.5 Battery 11

6.3.6 Case Design 11

7 Deployment Architecture View 11

8 Data Architecture View 12

9 System Process View 12

10 Implementation Architecture View 13

10.1.1 Android Implementation 13

10.2 Database Design 13

10.3 Web Services 14

10.4 Architecture Road Map 15

11 Architecturally Significant Risks 16

11.1 Users movement information 16

11.2 View GIS maps that overlays user’s information 16

11.3 Predict and send alerts to users/doctors 17

12 Architectural Open Issues 17

# Introduction

## Purpose

The purpose of this System Architecture Document (SAD) is to provide an overview of the ICT project, brief overview of the architectural components and implementation details. This document should be considered as a document that captures decisions made for the system.

## Scope

This document can be used by:

* System implementers: as implementation guideline.
* Project Managers: To create roadmap of work and to ensure implementation progress.
* System owners: To understand the system boundaries.
* System Architects: To evolve the system as it is built and to integrate to other external systems.

## Definitions, Acronyms and Abbreviations

|  |  |
| --- | --- |
| Triggers | Symptoms that cause an attack |
| PFM | Peak Flow Meter |
| SCH | Smart and Connected Health |
| Inhaler | A device used to increase the airflow |
| SCHAS | Smart and connected health alert system. |

## References

|  |  |  |
| --- | --- | --- |
| 1 | ICT Fund Document | See ICT Fund document |
| 2 | Architectural qualities list | <http://en.wikipedia.org/wiki/Non-functional_requirement> |
| 3 | World Health Organization. (2010). Medical devices. Retrieved 12/15/2013 | <http://www.who.int/mediacentre/factsheets/fs346/en/index.html> |
| 4 | Solid Works | <http://www.oocities.org/wpsmoke/solidworkswhy.html> |
| 5 | Hiemenz, Joe.  (2012). Excerpt 2: 3D printing with FDM Technology: How it Works. Retrieved 12/03/2013 | <http://blog.stratasys.com/2012/09/08/excerpt-2-3d-printing-with-fdm-technology-how-it-works/> |

# Architectural Goals and Constraints

## Goals

This section provides goals in terms of architectural qualities that must be exhibited by the SCHAS system. Among all the architectural qualities (see [2]), the following three qualities are chosen primary goals of the system. The system must be simple because it must evolve easily – complex systems are difficult to evolve to accommodate for uncertain changes to system. The system must allow exploratory activity rather than fixed set of features. The system will be used to conduct experiments and must allow various views of data and make it easy to load and transform data; the system must be maintainable by being able to isolate errors quickly – the system will be developed by many researchers and hence is more prone to become difficult to correct errors. The focus of the system is implement sophisticated prediction algorithms – therefore isolating errors and detecting and fixing is critical to speed up the time for development.

The top three qualities of the system must be:

* Simplicity: the system must be Simple
  + Simplicity includes the system will choose to implement components simplicity and readability over the efficiency.
  + It must be easily understood and provide ability to add new components easily
  + System uses simple frameworks including any third party tools
  + Data is managed in a way that allows it to be used for many purposes
  + The system uses open source frameworks such as Linux O.S. and simple file systems
* Exploratory: The system allows for exploratory activities such as visualization of location, movements, overlay of environmental factors etc.
  + This includes ability view data in multiple ways (tabular, graphical, spatial etc.)
  + This includes customizing the views (changing layers in GIS)
* The system is *Maintainable* – by that it means, it allows quick detection of any errors via tools and techniques to trouble shoot the system or subsystems.
  + This includes, ability to move data/software from one system to another system
  + The system includes just enough documentation to deploy and manage the working of it.

## Constraints

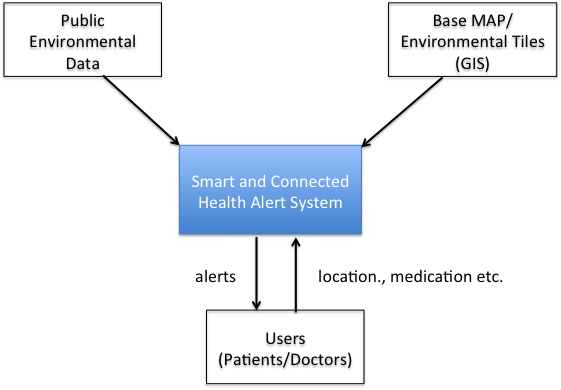
This section describes the constraints that application design such as: design and implementation strategy, development tools, technology, team structure, schedule, legacy code etc.

As such SCHAS will not have any legal constraints other than ensuring that the collected data does not have any personal identification. SCHAS also does not foresee any compliance to any standards.

# System Context

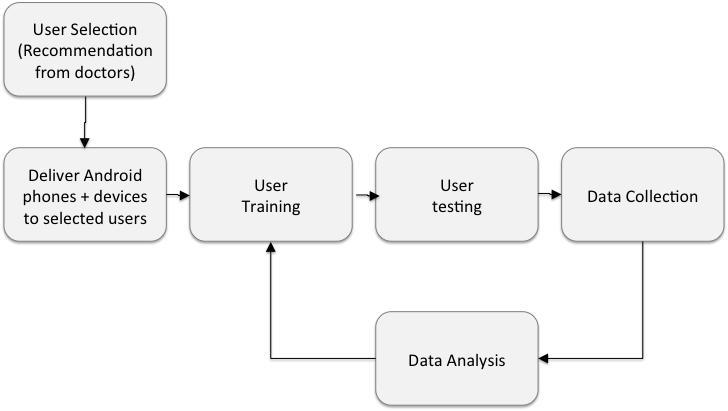
The following diagram shows SCHAS in the context diagram. The system will have minimum external interfaces and dependencies. The system will use base maps from freely available resources such as openstreetmap or base map from open layers. It will use freely available Web-Mapping-Services (WMS) to get environmental tiles to display on the map. Environment data for relevant locations will be downloaded from openweathermap (openweathermap.com).

The system will capture users location information and report to data collection service. Some users will be provide with environmental sensors – in this case, the data about the environment and location will be sent to data collection service. The system will allow users to input medication information and whenever they have a Asthma attack; The system will send alerts to users mobile devices.



# Business Process View

The main business scenario is to capture the data from the selected volunteer users. The business process view is as shown that shows the user selection and training to collect data for the system.



# Use Case View

The following diagram shows the architecturally significant use cases.

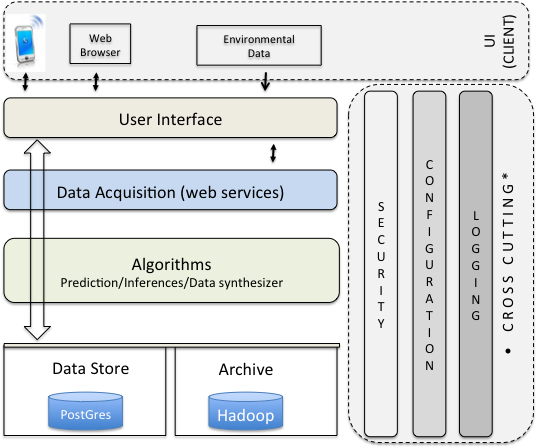
Three main functional use cases are identified.

| **Significant Use Cases** | **Description** | **Primary Actor(s)** | **Use Case Goals** |
| --- | --- | --- | --- |
| Store Patients movements and volunteer inputs | The goal of this use case is to capture the location information and other related information and send it to server successfully. Upon receipt Server will store the information in the database.  The patients mobile devices can capture the data from  (1) Upon usage of peak flow meter  (2) Upon usage of inhaler  (3) Patients input when attack occurred | * Patients sends data * System will receive and store it in the data store | * Send location data to server * Receive data and store it in server * Ensure no data loss |
| Architecturally Significant Requirements:   * Data received is stored securely and with minimum loss * Data received will clearly identify the sender and ensure that it is valid * Performance - Allow simultaneous sends and no data corruption * Make data available for analysis and online viewing | | |
| Ingest Environmental Data | Environmental Data (this include discrete and weather map tiles) will be stored for relevant time period. | * System Admin | * Environmental Data captured * Data is stored in the local storage * Data is reviewed and cleansed * Data is available for use |
| Architecturally Significant Requirements:   * Identify relevant time period and download environmental data * Ensure data is clean | | |
| Allow selection of interesting data to user | Allow users to review the data and select interesting data | * Researcher | * Selecting exploring interesting data |
| Architecturally Significant Requirements:   * Determine the data access efficiently * Usability of the system | | |
| Display patients stored data on the web browser | Process for defining projects, collection of expenditures and posting to the general ledger | * Researcher | * View on web * A way to validate the environmental data * A way to validate the customer inputs |
| Architecturally Significant Requirements:   * A reliable, secure interface is provided for interactive view and time reporting systems to process time transactions. * A reliable, way to explore the data visually | | |

## Main Actors

* System Administrator  
  Manages the system, adds users and loads data from external systems
* Researcher  
  Conducts exploratory analysis using the data collected from the patients and using the meta data such as environmental data. This role includes doctors and nurses as well.
* Patients  
  those who will volunteer to send the data required for the analysis.

# Logical Architecture View



The diagram above shows the logical overview of the system. It mainly consists of user interface layer to interface with mobile devices and web browsers.

User interface (UI) layer will be responsible for showing maps, of locations, overlaying environmental tiles (if available), allows users to load and view data for patients with interesting data.

UI layer will also implement minimum interfaces to manage users including patients. UI will be implemented using MVC pattern in most cases. Some exploratory analysis will be developed using

Data Acquisition system will be responsible for collecting the data or providing data to UI. This will be mostly implemented using web-services.

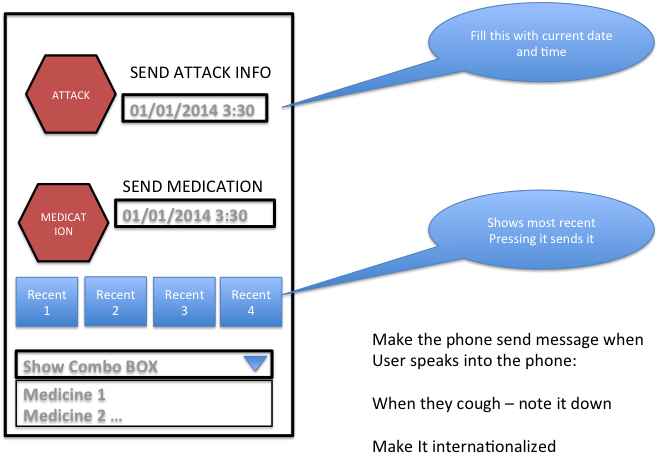
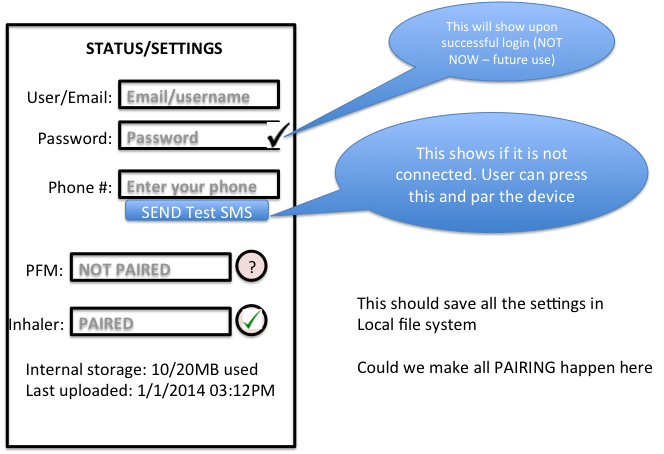
Algorithms will implement various machine learning algorithms, prediction services.

Data storage will be responsible for storing the data. The system will use RDBMS using Postgres and PostGIS database to store and manage data. RDBMS will have data to allow efficient querying; the system will use Hadoop system to achieve data.

The system will implement minimal security to ensure the data and views are accessible to valid users easily. However the data itself is available through secured password and all user information will be anonymized.

## Mobile device interfaces

The system will implement following set of interfaces on the mobile device. The goal of this subsystem is to make it simple to use for users.

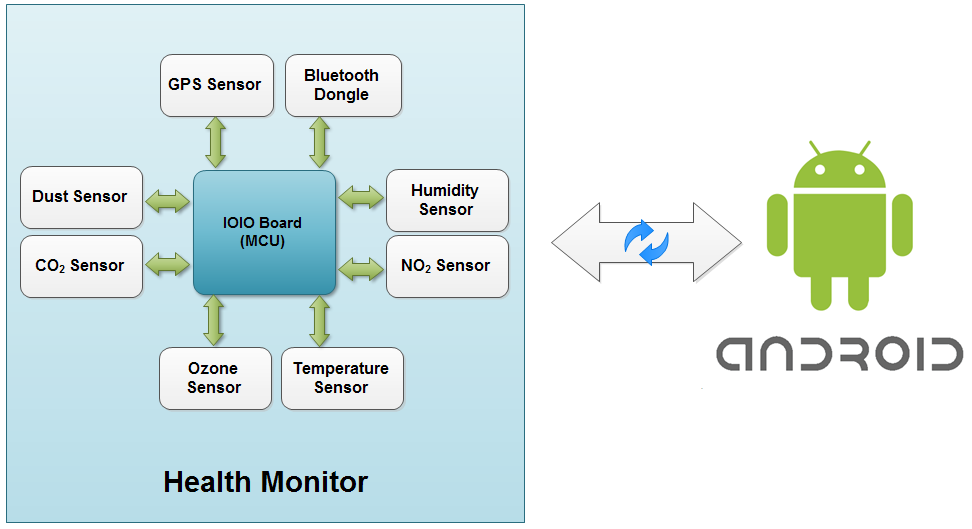
The first screen allows users to send medication and attach information. The second screen will allow users to setup the system for use.

The additional mobile modules that collect data from peak flow meter and inhalers are not shown here. These modules are shown as part of deployment view in the next section.

## Environmental Data Collection

The specifications for this module are as follows:

1. Communicate to an android device via Bluetooth
2. Get atmospheric condition updates using sensors such as GPS, temperature, humidity, NO2 and ozone.
3. Design an ergonomic enclosure
4. Supply power using a rechargeable battery
5. Interface a storage unit for sensor data
6. Eventually able to design on a single PCB



### Initial Prototype Development Board

The initial prototype will use IOIO board that is a development board that provides hardware I/O to Android applications. The board features PIC microcontroller and peripherals such as GPIO, PWM, ADC, I2C, SPI, and UART. The environmental data acquisition device will use ADC and UART to interface with the sensors. The PIC microcontroller gathers data from the sensors over the peripherals and sends it to the android device over an attached Bluetooth dongle or USB cable. The IOIO board does nothing unless it is receiving commands from the Android device.

To send commands to the board, the IOIOLibAndroid and IOIOLibBT libraries need to be added to the android project. The libraries provide simple commands for accessing the board’s peripherals. In the android application a class needs to be created that extends BaseIOIOLooper. BaseIOIOLooper has a setup method used for instantiating peripherals and a loop method used for reading the values of the peripherals. The setup method is run when the board first connects to the android device while the application is running. After the setup method finishes, the loop method is repeated for the remainder of the application run time.

### GPS

#### PMB-648

The GPS used in both devices is a PMB-648 module. This module tracks up to 20 satellites and updates at least once per second. It has a built in antenna that allows for some indoor use depending on the building structure. The module requires 3.3V – 5V DC, and consumes 65mA @ 5V DC. Data is provided in the NMEA0183 v2.2 format which contains the RMC sentence used in this application. The data strings are transferred using the UART protocol.

#### Wiring

The yellow wire is the TTL TX pin that the data transfers on this are connected to one of the UART pins on the IOIO board. The black wire is ground, and the red wire is connected to 5V.

#### Code

In the setup method the UART connection is established using the openUart method. In the loop method there is a check to make sure data it is sending data. If the module is sending data it is read from the InputStream and line-by-line using the ‘$’ symbol as an indicator for a new line. Once a new line is received there is a check to make sure that it is a RMC sentence. If it isn’t it repeats the above process. If it is a RMC sentence, the loop is exited.

### Temperature Sensor

#### TMP36

The temperature sensor used is the TMP36 device. The TMP36 is a low voltage sensor that requires 2.7V – 5.5V DC, and consumes less than 50uA of current. It scales linearly with temperature with ±2 degrees Celsius accuracy. The output pin provides an analog voltage value that can then be converted to temperature.

#### Wiring

The outer pins are connected to 3.3V and ground. The center pin is the output that is connected to one of the ADC pins on the device.

#### Code

There is no setup code for the Temperature sensor because there would be too many analog inputs left open if they were all opened in the setup code. This previously caused an error where the Bluetooth communication couldn’t keep up with the open connections. In the loop method the analog input needs to be opened, read, and then closed. This process is followed by a slight delay to prevent overloading the Bluetooth communication. The voltage value read is then converted to a temperature by using the following equation.

### Humidity Sensor

#### HIH-4030

The humidity sensor used is the HIH-4030 on a breakout board. The HIH-4030 measures relative humidity therefore an external temperature reading is required to get the actual humidity. The board requires 4V – 5.8V DC while consuming 200uA. NOTE: The datasheet claims the board needs at least 4V but other sources have tested and it works well at 3.3V. The sensor can report the relative humidity to ±3%. The output pin provides an analog voltage value that can then be converted to relative humidity.

#### Wiring

The outer pins are connected to 3.3V and ground. The center pin is the output that is connected to one of the ADC pins on the device.

#### Code

There is no setup code for the Humidity sensor because there would be too many analog inputs left open if they were all opened in the setup code. This previously caused an error where the Bluetooth communication couldn’t keep up with the open connections. In the loop method the analog input needs to be opened, read, and then closed. This process is followed by a slight delay to prevent overloading the Bluetooth communication. The voltage value read is then converted to a relative humidity value by using the following equations.

– 25.8

### Ozone Sensor

#### MiCS-2610

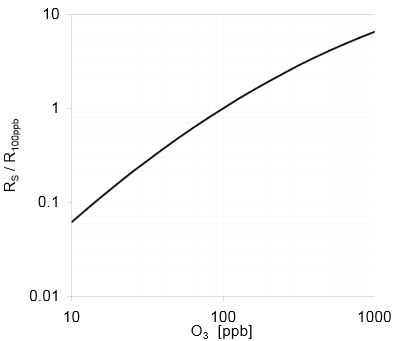
The ozone sensor used is the MiCS-2610. The MiCS-2610 has one circuit for heating the sensor up, and one circuit that measures the O3 in the air. For best operation the heater circuit is powered by 80mW with an internal resistance of 58Ω -78Ω. The sensor circuit is powered by 5V DC and needs to be placed in series with another resistor to measure the resistance of the internal sensor circuit. From additional reading the external resistor value should be close to the internal resistance of the sensor at low O3 concentrations (3kΩ -60kΩ). The internal resistance changes with the O3 concentration in the air within the range of 10-1000 ppb. This sensor is technically not accurate until calibration is completed. This was never completed properly so the readings should only be used to detect changes in the concentration.

#### Wiring

The heater circuit consists of pins 1 and 3. Pin 1 is connected to ground. Pin 3 is connected to a 75Ω resistor that is connected to the 5V supply. Pins 2 and 4 make up the sensor circuit. Pin 2 is connected to 5V supply. Pin 4 is connected to a 3.3kΩ resistor that is connected to ground. Between pin 4 and the resistor is the analog output value that needs to be connected to the ADC pin on the IOIO board.

#### Code

There is no setup code for the Ozone sensor because there would be too many analog inputs left open if they were all opened in the setup code. This previously caused an error where the Bluetooth communication couldn’t keep up with the open connections. In the loop method the analog input needs to be opened, read, and then closed. This process is followed by a slight delay to prevent overloading the Bluetooth communication. The voltage value read is then converted to a relative ozone concentration value by using a table, based on the chart below, which maps the sensor circuit resistance to the concentration in ppb.



### Nitrogen Dioxide Sensor

#### MiCS-2710

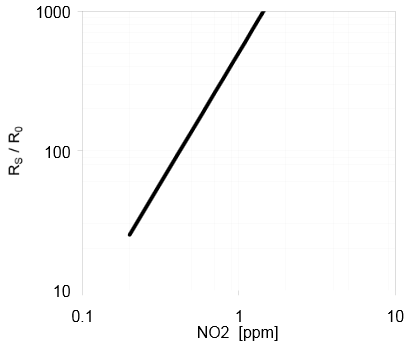
The nitrogen dioxide sensor used is the MiCS-2710. The MiCS-2710 has one circuit for heating sensor up, and one circuit that measures the NO2 in the air. For best operation the heater circuit is powered by 43mW with an internal resistance of 59Ω -73Ω. The sensor circuit is powered by a maximum of 2.5V DC and needs to be placed in series with another resistor to measure the resistance of the internal sensor circuit. From additional reading the external resistor value should be close to the internal resistance of the sensor at low NO2 concentrations (2.2kΩ -8kΩ). The internal resistance changes with the NO2 concentration in the air within the range of 0.05-5 ppm. This sensor is technically not accurate until calibration is completed. This was never completed properly so the readings should only be used to detect changes in the concentration.

#### Wiring

The heater circuit consists of pins 1 and 3. Pin 1 is connected to ground. Pin 3 is connected to a 220Ω resistor that is connected to the 5V supply. Pins 2 and 4 make up the sensor circuit. Pin 2 is connected to connect to a voltage divider between two 12k resistors that cut the voltage to the maximum of 2.5V required to power the sensor. Pin 4 is connected to a 2.2kΩ resistor that is connected to ground. Between pin 4 and the resistor is one of the analog output values that needs to be connected to the ADC pin on the IOIO board. Since a voltage divider doesn’t provide a steady voltage another wire connects pin 2 to the ADC. This way an accurate voltage drop over the sensor can be read.

#### Code

There is no setup code for the Ozone sensor because there would be too many analog inputs left open if they were all opened in the setup code. This previously caused an error where the Bluetooth communication couldn’t keep up with the open connections. In the loop method the analog input needs to be opened, read, and then closed. This process is followed by a slight delay to prevent overloading the Bluetooth communication. The two voltage values are then converted to a relative nitrogen dioxide concentration value by using a table, based on the chart below, which maps the sensor circuit resistance to the concentration in ppm.



### Case

The case for the device is 3d printed from solid works files. The dimensions are roughly 120mm x 83mm x 41mm for the newer device and 108mm x 83mm x 41mm for the older device. The reason for this size distance is that the IOIO board updated and replaced the USB port with a micro USB port. The change requires an adapter cable to plug in the Bluetooth dongle which takes up additional space.

### Battery

The device is powered using 4 AA NiMh rechargeable batteries in series with one another. They each provide roughly 2500mAh of charge and 1.2V. In series they add up to 4.8V, which is enough to satisfy the IOIO board’s voltage requirement. The entire sensor unit uses roughly 300mA of current so the device can theoretically run for 8 hours on the 4 AA batteries.

### Second generation Air Quality monitor

The system will experiment with a different Bluetooth device that is smaller in size both for inhaler and for the environmental data collection hardware system.

## Inhaler cap and peak flow meter design

### Circuit Design

The circuit is assembled on a perforated printed circuit board. The circuit consists of a button, the ATtiny85 microcontroller, and the RN-42 Bluetooth module on the top, and the battery holder on the bottom. The button is connected to the microcontroller and triggers an interrupt when pressed. This wakes the microcontroller and sends a message to the RX pin on the Bluetooth module.

### RN-42 Bluetooth Module

The RN-42 is a class 2 Bluetooth module that contains the entire Bluetooth stack. The only pins used on the module for this application are the power, ground, and RX pins. When a character is sent to the RX pin using UART communication, it sends the data over Bluetooth to the connected Android device. The module also has a command mode where the configuration can be changed. For this application the baud rate is set to 2400 and some other various low power settings are configured.

### Android Program

The Android device is paired with the inhaler cap it is going to use. After the initial setup, no pairing should be necessary and the device will connect automatically when in range. The Android code listens for messages that the inhaler cap sends, and stores the timestamp. Currently the cap may send multiple messages for one inhaler press so the device cannot be used for distinguishing each use of the inhaler but it can provide the general time it was used.

### ATtiny85

The ATtiny85 is an Atmel 8-bit AVR RISC-based microcontroller. The microcontroller is programmed to sleep until an interrupt is triggered by the button press. It then sends a message to the RN-42 Bluetooth module using UART communication. The microcontroller doesn’t have a hardware UART module so it is simulated in software by using a standard UART library. Due to inaccuracies in the internal clock, the baud rate needs to be set to 2400.

### Battery

The battery used is a lithium ion coin cell battery, specifically a rechargeable CR2450. The battery rated for 3.6V at 110mAh. The circuit uses around 4mA of current so the device should last a full 24 hours before it needs to be recharged. Originally non rechargeable CR2450 were used to containing 600mAh but they quickly drop below 3V which is below the minimum spec for the RN-42 Bluetooth module.

### Case Design

The case will be designed in Solidworks and is made of 3d printed plastic. It consists of a top part, an insert, and a bottom part. The top part is a hollow cylinder that has an opening at the top and bottom. The top opening is slightly smaller than the cylinder and insert in order to prevent it from falling out. The bottom opening has screw threads which are used to attach to the bottom part. The insert is a thin circle that sits inside the top half above the electronic circuit. The purpose of the insert is to provide a larger area that the user can push instead of trying to press the small button on the circuit. The bottom part is designed to have a tight fit on the top of an inhaler. There are screw threads on the outside which allow it to be attached to the top half of the case.

Peak Flow meter design follows as that of Inhaler cap design. However the measurement of peak flow usage and its design could be complex therefore the system may purchase it from third party.

## User Interface

### User Interface

The system will have user interface to:

* Register or create, edit, disable, or view account for users (User accounts are never deleted unless required for legal reasons)
* View existing users and query for users with interesting data

### Openlayer Maps

The system will use Openlayer Map java script library to show interactive maps. The data for the map will be procured from various weather map sites and through systems owns data collection methods.

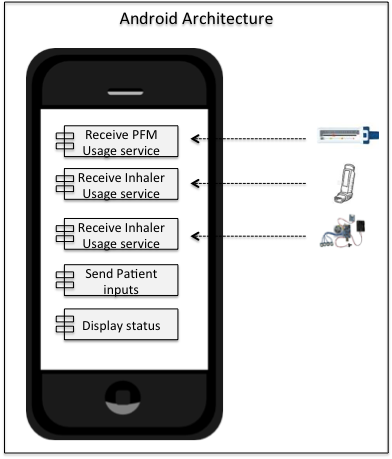
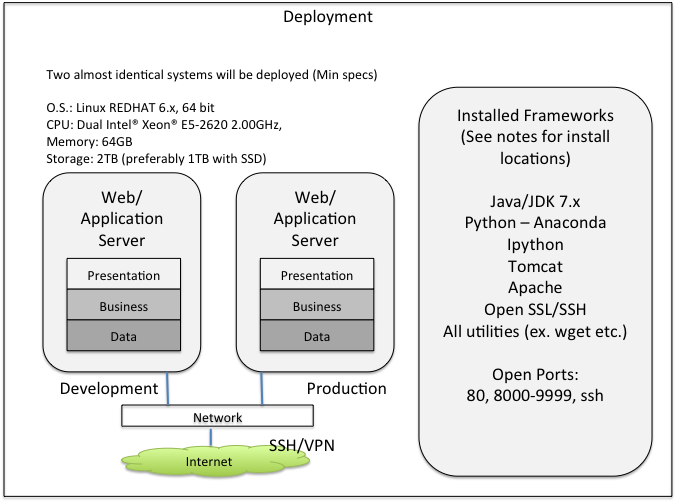
The map will have a base map from any of “open layers”, “open street map” etc.

The map will have additional layers that can be turned on or off. The layers are:

* Show patients trajectories
* Show Weather map data – vector layer
* Show Weather mapped tiles (a specific WMS service is unclear and must be determined during the project)

# Deployment Architecture View

The system deployment is as shown below.



# Data Architecture View

The significant data elements of the system are as follows:   
All these data are time stamped in GMT

* Patients movement data
* Patients medication and inhaler data
* Patients data that indicate when an attack occurred
* Environmental data

# System Process View

Most system processes will be implemented as state-less. There will not be a need for long standing transactions. However there are components that have interdependency on other components and thus must be available to ensure that overall working of the system. Such dependencies are shown below.

Web services are the central part of the overall system. Web services depend on the tomcat application server.

|  |
| --- |
|  |

# Implementation Architecture View

The implementation uses the following technologies.

* Linux Redhat 6.x (or Ubuntu) version of O.S
* Tomcat as application server front ended by Apache server
* Hadoop file system will used in the second phase of the project to store large amounts of data. In the initial phase all files will be stored locally
* Postgres with PostGIS extensions will be used as database.
* The project will use Java, Python as main programming languages.
* JUNIT for writing test scripts
* Openlayers will be used for displaying the GIS maps
* Javascript will be used for most of the client facing systems
* Jlog for logging

### Android Implementation

Activities: Screen that user can interact with

Services: Components that run in background

Content Providers: Data sharing mechanisms

Broadcast Receivers: Send and receive notifications.

## Database Design

The following simple schema will be implemented on the Postgres +PostGIS DB.



This is the simple database schema that will be created to store the data for the initial prototype.

User table will be used to stored users who are registered with their id. Each user will be assigned a mobile ID that they would use to report the readings.

“ENV” table will store the environmental data. The columns “values” in this table will be used to hold any key value pairs stored as JSON formatted strings.

Health table will store the users input collected from using PFM, inhalers and voluntary inputs of attacks or any medication intakes.

“ENV” and “Health” tables will have latitude and longitude information that will be used as POINT data for PostGIS to create a spatial index.

The data will be loaded dynamically as and when needed from the flat files. The scripts to load the data to the PostGIS table will be developed with parameters for loading data for time and set of users.

## Web Services

The system will develop following web services to support data collection. Initially, the data will be uploaded with no security protocols. Second phase will implement https protocols with official certificates.

These web services must be high reliable and available.

The devices that will upload data must ensure that the correct successful return codes are received so as not to risk the loss of data.

|  |  |  |
| --- | --- | --- |
| **Web Service** | **End point** | **Description** |
| Collect Environmental data | Env.jsp  Example:  http://www.geospaces.org/aura/webroot/env.jsp?api\_key=1&ver=1.0&id=23&lat=34&lon=12.3&alt=1000ft&temperature=34.0 | The service will be used to collect environmental data  The data will be stored in a file names ENV.txt which will be used to upload to postgres when needed |
| |  |  |  | | --- | --- | --- | | **Parameter** | **values** | **Description** | | code |  | This is to implement security in future. For now, if it is not sent the data will not be stored | | ver | 1.0 (optional) | Indicates which version of protocol being used | | lat | latitude | Location Data | | lon | longitude | Location Data | | time | unix time in GMT | Unix time is | | api\_key | Not used – | will be used to implement security in second phase | | |
| Collect Health data | Health.jsp  Example:  http://www.geospaces.org/aura/webroot/health.jsp?ver=1.0&id=23&lat=34&lon=12.3&alt=1000ft&time=5767667&htype=MED&<values> | The service will be used to collect health data  The data will be stored in a file names HEALTH.txt which will be used to upload to postgres when needed |
| htype= is one of [MED, ATT, PF, IH]    MED = Medicine in take  ATT = Attack  PF = Peak Flow usage  IH = Inhaler use | |
| Access to Data | A simple interface will show all the data files in the initial phase:  http://geospaces.org/SCHAS/data/ |  |
| Bulk Environment File Upload | ENV\_File  This end point will allow users to upload a environmental file as an upload. It will be used by mobile devices that will collect data locally and bulk upload the data | The file will confirm to the ENV.txt format. |
| Bulk Health data upload | HEALTH\_File  Similar to bulk environmental upload except this will support uploading bulk health data | The file will confirm to HEALTH.txt format |
| Test | Test | Ensures all the services are running |
| Version | Version | Displays the current version number of release of the software |

## Architecture Road Map

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Comments** |
| System setup and Environment setup | Set up servers and create development environment |  |
| Develop initial web services and establish connectivity of mobile devices with data collection service | Initial prototype to collect health and environmental data |  |
|  |  |  |
| **Android** |  |  |
| Application on Android devices | Application interface installed on Android devices to collect and send data to server | Android application download and installed from Google Play store |
| PMF sensor design and integration with Android phone | Develop PMF sensor and test the connectivity with the Android device |  |
| Inhaler sensor design and integration with Android device | Same as PMF device integration |  |
| Component to send location information from Android device | Receive location information from the devices along with user input data |  |
|  |  |  |
| **Data** |  |  |
| Environmental Data ingest | Investigate environmental data from openweathermap.  Ingest samples of vector data and raster data. | Create services/algorithms to get the weather for a given location and time. |
|  |  |  |
| **User Interface** |  |  |
| Openlayers to show map | Display base map and allow showing of trajectories of the patient (s) for a time period.  Show the indications of the attack or medication intakes. This will allow users to choose the patients whose data will be displayed. |  |
| CRUDX Users | Manage users for the system (create, delete etc.) |  |
| Users exploration | Allow exploring users and their corresponding available data in order to find interesting users |  |
| Graphs | Show charts and graphs of users movement and corresponding environmental data |  |
| Display Environmental data | Show various environmental data using charts and maps. Display weather data on the map using weather tiles |  |
|  |  |  |
| **Algorithms** |  |  |
| Interpolate environment | Determine the weather by interpolating the data between location and time |  |
| Predict | Predict attacks based on the users data |  |

# Architecturally Significant Risks

Risk: Manage the data, data archiving, and security of data

Mitigation: Consistently archive the data and move it into secured place

Risk: Availability of the system and system administration issues:

Mitigation: Handled manually with hard reboots; eventually move to Amazon cloud services

## Users movement information

Collect users location information along with data about environment or any voluntary inputs and send to data collection service. It is possible that users may not use the mobile devices to upload asthma attack or medication information. Architecturally, it makes it challenging for prediction algorithms to adjust for missing data.

Mitigation: Consult with doctors to promote motivation for patients to upload data consistently.

## View GIS maps that overlays user’s information

Select interesting users for whom the data has been collected and display them by overlaying it on a GIS map that shows a map along with weather conditions at the corresponding time.

The weather module is complex with multiple interpolations (1) location (2) time.

Mitigation: Start on the early design and consult with meteorology experts. Another option is to purchase data and service from paid services such as open weather map.

## Predict and send alerts to users/doctors

Create prediction service and send alerts to users when the system predicts an attack could happen. The alerts may be incorrect in the early phases or during the development.

Mitigation: Train the users to ignore the alerts. System will not send alerts directly to the users; instead, it will send a message to the system about the possibility of the attack. This will be correlated with the user input.

# Architectural Open Issues

|  |  |  |
| --- | --- | --- |
| **Issue** | **Description** | **Resolution** |
| User connectivity to data collection servers | Currently the architecture provides one server to collect the data from users. If the data collection service is down, then the data collected will be lost | 1. Develop a monitoring service to monitor the data collection service.  2. Send data that is clearly identified as mocked data  3. Send alerts to system admins if data collection service is down  (Need to communicate this to board) |
|  |  |  |