## C:\Users\Kurteisliga\Pictures\Capture4.PNGLogical Architecture

The logical architecture implements a three-tier system. This architecture includes a user interface layer, business object layer (which includes business object managers) and a persistence layer.

**User Interface Layer**

The user interface is broken into two components: the mobile client interface, and the web application.

**Mobile Client User Interface**

The mobile client user interface interacts with two logical packages, the mobile client business object managers, and the business objects. Within the mobile client user interface is a MobileClientUI class that acts as the main frame of the program. This includes all buttons, text areas, and dialog boxes. It acts as an interface manager for the remaining user interface classes. The MobileClientUI interacts directly with the MobileClient business object manager.

The mobile client user interface contains the following classes:

**MobileClientUI**

MobileClientUI acts as the main windows of the mobile client, as well as the manager of the mobile client user interface classes.

Figure 1: User interface layer

**iVOCButton**

iVOCButton is an interface for buttons with the functionality of pressing the button, an action handler, a success dialog box, and a failure dialog box. VOCConnect, VOCDisconnect, VOCInitialize, VOCDownload, and VOCMonitorShutDown all inherit for iVOCButton.

**VOCConnect**

VOCConnect represents the button to connect to the Arduino.

**VOCDisconnect**

VOCDisconnect represents the button to disconnect from the Arduino.

**VOCDownload**

VOCDownload represents the button to download the data from the Arduino.

**VOCInitialize**

VOCInitialize represents the button and window surrounding the functionality of initializing the Arduino. This includes longitude, latitude, zip code, minutes/hours/day/month/year, and the type of sensor it is reading from.

**VOCMonitorShutDown**

VOCMonitorShutdown represents the button surrounding shutting down the Arduino. This includes purging the data from the SD card and turning off all sensor reading capability.

**Web Application User Interface**

The web application interacts with the web business object manager package directly. It contains the methods behind the monitor map, downloading of raw data, viewing a graph, and uploading data to the database.

The web application user interface contains the following classes:

**MonitorMap**

MonitorMap is the Google Maps VOC Monitor location map which populates from the VOC Monitor location map manager.

**RawData**

RawData handles all the capabilities for the Download Raw Data web application.

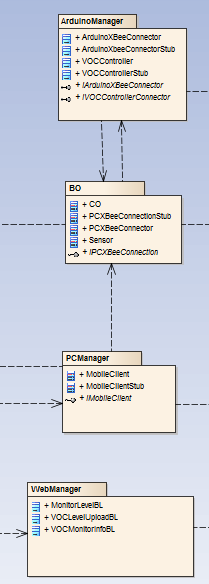
**VOCLevelGraph**

VOCLevelGraph handles all the functions needed for the View Graph web application.

**VOCLevelUpload**

VOCLevelUpload encompasses the UI functionality to choose a file to upload to the database.

**Business Object Layer**

The business object layer is broken into four components: mobile client business object manager, business objects, Arduino business object manager, and the web application business object manager.

**Web Application Business Object Manager**

The web application business object manager interacts with the web persistence.

The web application business object manager contains the following classes:

**MonitorLevelBL**

MonitorLevelBL contains all of the functions necessary in communicating between the user interface layer and persistence layer for displaying on the web application.

**VOCLevelUploadBL**

VOCLevelUploadBL is used in order to implement a three-tier architecture between the user interface and persistence layers.

**VOCMonitorInfoBL**

VOCMonitorInfoBL handles taking the VOC Monitor locations from the database, storing them into an array and populating the Google Maps map, from which the web user can navigate to a VOC monitor marker by entering a zip, or locate a monitor manually.

Figure 2: Business object layer

**Mobile Client Business Object Manager**

The mobile client business object manager interacts with both the business objects and the persistence layer. It contains the classes built for directing the flow of the mobile client, such as when to access the Arduino and store data. When calls are made to access the Arduino, the business object layer will then return values depending on how the interaction went (whether it was successful or not) and the business object manager will then direct the output to the user.

Because of the ability to direct the output to the user, there is a bi-directional relationship between the mobile client user interface package and the mobile client manager package.

The mobile client business object manager contains the following classes:

**MobileClient**

MobileClient is a manager of the mobile client business objects. This includes directing when to write/read from a file and connect to the Arduino. It inherits from iMobileClient.

**MobileClientStub**

MobileClientStub is used in unit testing in order to give the tests a known value. It inherits from iMobileClient and contains PCXBeeConnectionStub.

**iMobileClient**

iMobileClient is an interface used for streamlining the functionality of MobileClient and MobileClientStub.

**Business Objects**

The business objects do the heavy lifting for both the Arduino managers and the mobile client managers. This includes writing to and reading from files, the actual connection between the Arduino and mobile client, and reading from the sensor. Due to the ability of the business objects to interact with the Arduino and the Arduino to interact directly with it, there is a bi-directional relationship between the two packages.

The business objects contains the following classes:

**CO**

CO handles the reading from a carbon monoxide sensor. It inherits from Sensor.

**Sensor**

Sensor is an abstract class for inheriting other sensors from (such as carbon monoxide) so as to give them streamlined functionality.

**PCXBeeConnectionStub**

PCXBeeConnectionStub is used in unit testing in order to give the tests a known value. It inherits from iPCXBeeConnection.

**PCXBeeConnection**

PCXBeeConnection handles all interaction with the Arduino. It is directed by the MobileClient object manager.

**iPCXBeeConnection**

iPCXBeeConnection is an interface for streamlining PCXBeeConnection and PCXBeeConnectionStub functionality.

**Arduino Business Object Manager**

The Arduino business object manager package interacts with both the persistence layer and the business objects. This gives it the ability to communicate with the mobile client, store sensor data, and store initialization data.

The Arduino business object manager contains the following classes:

**ArduinoXBeeConnection**

ArduinoXBeeConnection contains the methods to establish a connection between the mobile client and the Arduino over XBee connection. It inherits from iArduinoXBeeConnection.

**ArduinoXBeeConnectionStub**

ArduinoXBeeConnectionStub is used in unit testing in order to give the tests a known value. It inherits from iArduinoXBeeConnection.

**VOCController**

VOCController represents the Arduino and the functionality it contains. It inherits from iVOCControllerConnector.

**VOCControllerStub**

VOCCOntrollerStub is a fake representation of VOCController with known, unchangeable values. It inherits from iVOCControllerConnector.

**iArduinoXBeeConnection**

iArduinoXBeeConnection is an interface for streamlining ArduinoXBeeConnection and ArduinoXBeeConnectionStub functionality.

**iVOCControllerConnector**

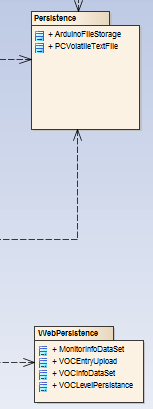
iVocControllerConnector is an interface for streamlining VOCController and VOCCOntrollerStub functionality.

**Persistence Layer**

The persistence layer is broken into two packages: mobile client/Arduino persistence, and the web persistence.

**Arduino/Mobile Client Persistence**

This package contains the functionality behind writing to and from the Arduino files, as well as the mobile client files.

****The Arduino/mobile client persistence contains the following classes:

**ArduinoFileStorage**

ArduinoFileStorage contains the methods that enable file storage on the VOC monitor.

**PCVolatileTextFile**

PCVolatileTextFile contains the methods for downloading the text file to the users’ PC.

**Web Persistence**

The package contains the functionality behind reading and writing to the database sensor data, monitor IDs, monitor location, date of sensor reading, and what type of sensor was used for the reading.

The web persistence contains the following classes:

**MonitorInfoDataSet**

MonitorInfoDataSet creates strongly typed objects dealing with information about monitors in the database

Figure 3: Persistence layer

**VOCEntryUpload**

VOCEntryUpload contains the methods around uploading monitor information to the database.

**VOCInfoDataSet**

VOCEntryUpload enables strongly typed objects pertaining to information about the Volatile Organic Component being sensed.

**VOCLevelPersistence**

VOCLevelPersistence provide the persistence layer logic to pull the data from the VOCMS DataSet objects and put them into primitive data types for the business object layer.

## Component Architecture

The component architecture is broken into three primary systems, just as the logical architecture has been: the monitoring system, mobile client, and the web.

**Monitoring System**

The monitoring system has four sub-components: ArduinoBO, ArduinoManager, ArduinoTestApplication, and VOCResident.

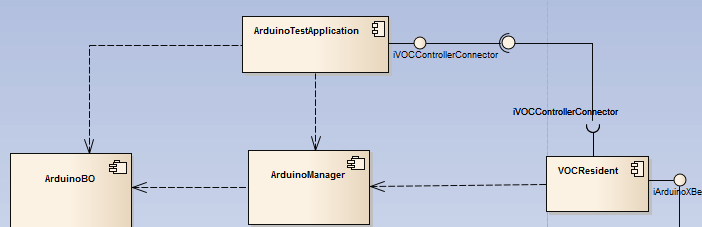
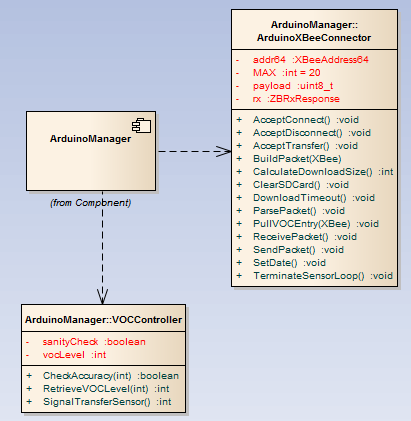


Figure 4: Arduino Components

**VOCResident**

The VOCResident is the executed program. This is to be uploaded prior to the monitor being used. It then accesses the library ArduinoManager. It interfaces to ArduinoTestApplication through a serial connection, with the XBee module turned on.

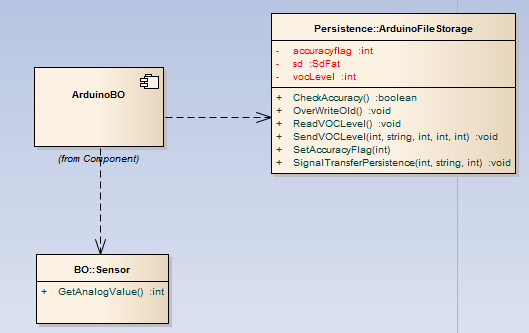
 **ArduinoManager**

The ArduinoManager component contains the classes ArduinoXBeeConnector and VOCController. With these classes, it then accesses ArduinoBO. These classes handle the connection between the mobile client, as well as sanity checking for the analog value read in from the sensor.

**ArduinoBO**

The ArduinoBO component contains the classes ArduinoFileStorage and Sensor. The purpose of this is for the sensor to read in a value, and then pass the value to the file storage in order to save it for uploading to the mobile client later.

Figure 5: Arduino business object manager library



**ArduinoTestApplication**

The ArduinoTestApplication contains tests for the Arduino. The XBee must be connected in order to use this application.

**Mobile Client**

Figure 6: Arduino business object library

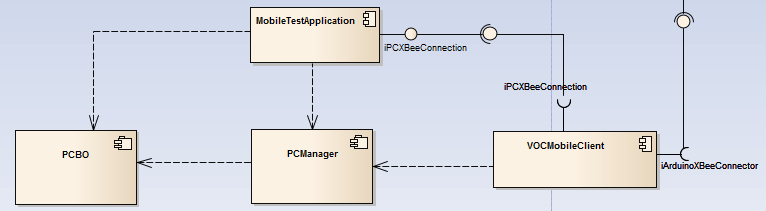
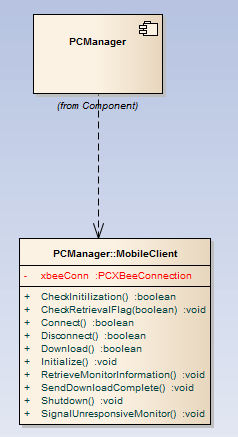
****The mobile client has four sub-components: PCBO, PCManager, VOCMobileClient, and MobileTestApplication.

Figure 7: Mobile client components

**VOCMobileClient**

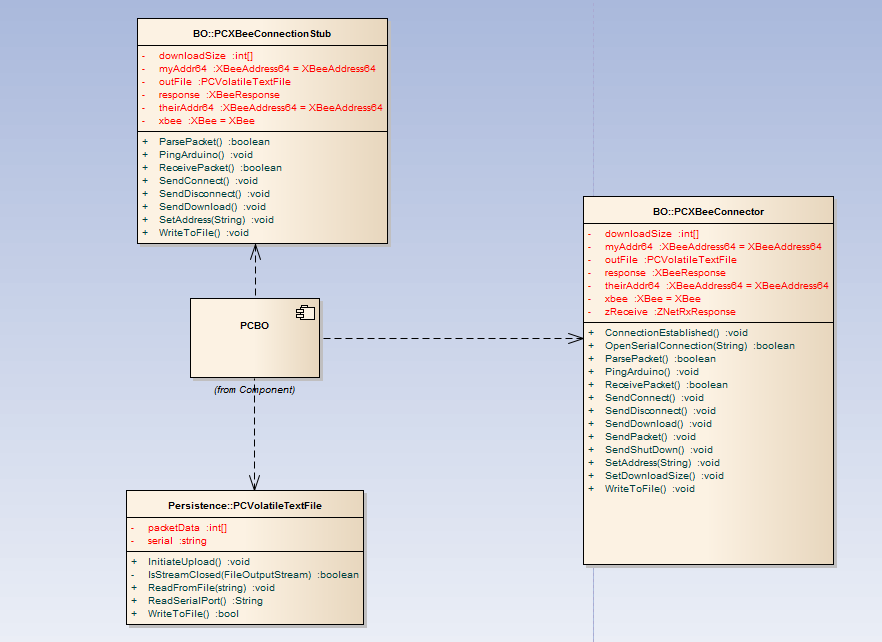
****The VOCMobileClient is the executed program. This can be ran at any time, though it will only be successful if the XBee Dongle is connected (as explained later). It interfaces with the PCManager library. It interacts with the VOCResident component through the iArduinoXBeeConnector interface.

**PCManager**

The PCManager library contains the class MobileClient. MobileClient acts as an interface between the GUI, business objects, and persistence layer.

Figure 8: Mobile client business object manager library

**PCBO**

****The PCBO library contains the classes PCXBeeConnector and PCVolatileTextFile. It also contains a stub for the MobileTestApplication to run off in order to simulate certain behaviors. PCXBeeConnector handles the interaction between the Arduino and the mobile client. It contains the methods required to connect to the Arduino, disconnect, and download data from the monitor. After the data has been downloaded, PCVolatileTextFile will then store the data captured on the mobile client for uploading to the database later.

**MobileTestApplication**

Figure 9: Mobile client business object library

The MobileTestApplication contains unit tests based around both the XBee connection and file storage.

**Web**

The mobile client has four sub-components: MonitorLevel\_BL, WebManager, DataBaseMobileTestApplication, and DataBaseMobileClient.

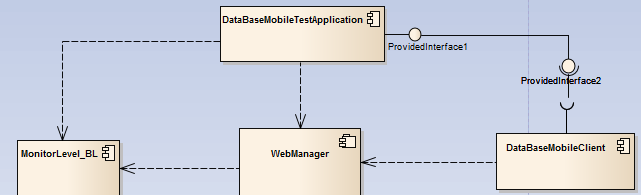
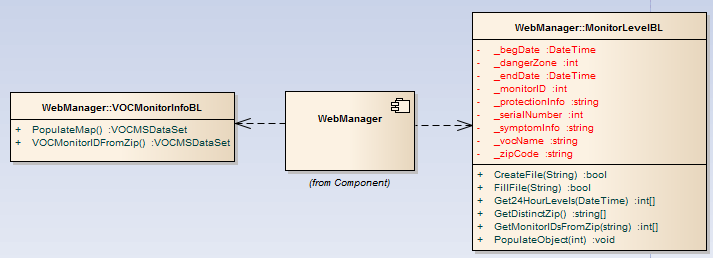


Figure 10: Web Application components

**MonitorLevel\_BL**

The MonitorLevel\_BL contains the business objects contained within the web portion.

**WebManager**

**** The WebManager contains the business object managers for the web portion.

**DataBaseMobileTestApplication**

Figure 10: Web Application business object manager library

The DataBaseMobileTestApplication contains the unit tests for the DataBaseMobileClient, as well as a stub for the database.

**DataBaseMobileClient**

The DataBaseMobileClient is the web application. It interacts with both the WebManager and DataBaseMobileTestApplication.

## Physical Architecture

This section covers the physical components of the system as well as how they map within the architecture. This includes the Arduino, SD card “shield”, SD card, XBee shield, XBee, sensor, the breadboard for which the sensor is attached to, and the dongle used to connect the mobile client to the Arduino. A “shield” is a module for extending the Arduino, giving it greater functionality.

**Arduino**

****The hardware used is an Arduino R3. The purpose of this choice was for its ability to be modified and connect to sensors in a simple manner. The hardware was between a Raspberry Pi or the Arduino. While the Raspberry Pi can do what the Arduino can and more, it has more overhead in that it requires a full operating system such as Linux. This would put a greater drain on the battery, which we could not sacrifice. The Arduino has the ability to store data long term using the SD card, connect wirelessly with separate devices using the XBee, and read in sensor data using the breadboard. These were key aspects that were required in the system. The Arduino contains the ArduinoTestApplication, VOCResident, ArduinoManager, and ArduinoBO components.

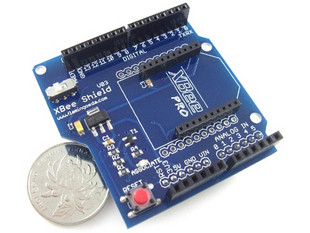
**SD Card Shield**

The SD shield is a simple interface for adapting the SD card to the Arduino. It plugged in directly to the top of the Arduino in such a way that it more modules could be attached on top of it. It also had the ability to adjust the voltage, and had a maximum SD card storage capacity of 8 gigabytes.

**SD Card**

The purpose of the SD is simply to store data. What was needed from this is being able to store analog values for months at a time. With that knowledge, a 4 gigabyte card was chosen, as running the monitoring system for a year, at 1 reading per hour, would allow over 8 years of straight running. Our second reasoning for choosing a 4 gigabyte SD card was the price; it was cheaper than comparable cards of equal, smaller, and greater storage capacity. The SD card is inserted directly into the SD card shield

**XBee Shield**

****The XBee shield, much like the SD card shield, is used as an interface for the component that it is attached to. In this case, the XBee is attached to the top of the XBee shield, and the XBee shield is attached to the top of the SD card shield.

**XBee**

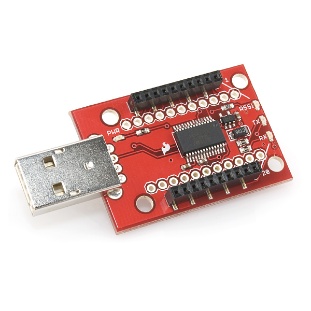
The XBee implements the IEEE Zigbee networking functionality. The purpose of this is to be able to communicate directly with the mobile client. The XBee Pro was chosen due to its ease of use compared to similar XBee modules (16bit addressing versus 64bit addressing for other XBees). It also has low power consumption. The downside to this module is the range: 300 yards of direct sight. While this is low, we considered it to be an acceptable fault, given the means for which data is collected.

**Sensor**

The purpose of [the sensor] is to read in analog values. The current sensor used is for carbon monoxide, due to the ease in encountering carbon monoxide and thus easier to calibrate.

**Breadboard**

The breadboard is the interface between the sensor and the Arduino. Its purpose is to give the sensor a stable location to be inserted, as well as the ease of wiring the data input and voltages for the sensor.

**XBee Dongle**

The XBee dongle serves the same purpose as the XBee shield, except it is the interface between the mobile client and XBee. It has the same model of XBee attached to it. It does not contain any program.