Smart Mirror

Justin Gentry, Michael Trivelli, and Hector Zacarias

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

Abstract — A staple furniture piece found in every bedroom and bathroom, the mirror has provided a means for effective personal grooming for thousands of years. Our team has brought it into the 21st century with Smart Mirror. While preparing for the day, users will be able to glance at their mirror and instantly retrieve important bits of information such as the current time, date, weather, news, and more. Our Smart Mirror is powered by the affordable yet powerful Raspberry Pi single-board computer. Our suite of software modules combined with a touchless UI ensures that you remain at the forefront of the emerging smart-home revolution.

Index Terms — Application software, Internet of Things, Microcontrollers, Microcomputers, Smart Homes, Home Automation, Smart Mirrors.

I. INTRODUCTION

Technological integration into homes, so called "Smart Home Technology" is becoming increasingly popular in the consumer electronics industry. The primary benefit of smart home technology is to simplify our day to day lives in any way possible. Some benefits include saving time, relieving stress, or even saving money. These benefits can be accomplished in a number of ways including automation of tasks, improved access to media and information, as well as varying degrees of personal comfort.

Smart Mirror will be a smart home implementation in the bathroom, a room currently lacking technological innovation. Smart Home technology has been integrated to a number of rooms and interfaces throughout the home however the bathroom has been left mostly untouched. Almost every person spends some portion of their time daily in the bathroom. As a result, there is ample opportunity to present users with information that could improve their daily lives.

The mirror will be able to present personalized information to users every morning as they prepare for their day. Ideally it will save users time by displaying information they would likely check in the morning. This includes information such as weather, daily schedule, news, and the time so the user can keep on schedule. These simple bits of information are commonly sought after in the morning while preparing for the day or at night before going to sleep. The information will be provided on the mirror in an unobtrusive manner, leaving the majority of the prime mirror real estate unaltered. This will allow the user to easily absorb the displayed information while going about their normal routine. User interaction will be achieved primarily through voice commands with some minimal additional interaction provided by side-mounted motion sensors.

II. SYSTEM COMPONENTS

Smart Mirror is built on a platform consisting of an assortment of components both purchased and designed to work together. The assembly of these components results in a cohesive product that is reflected in the user experience.

A. Primary Computer System

Smart Mirror utilizes a Raspberry Pi 2 Model B as the primary computer. Weighing in at just 0.1 lb. and sporting a Broadcom Quad-Core ARM7 900MHz processor, the credit card-sized Raspberry Pi was the optimal choice when selecting a powerhouse for the mirror. While similar projects opted for full-fledged compact PC builds, the Raspberry Pi provides all of the required processing power and interfaces for under forty dollars. All of the sensor information is transferred to the Pi via the 40 available GPIO pins. The software for the mirror will be built on the Universal Windows Platform allowing for execution on any Windows 10 or Windows 10 IoT machine. Our Raspberry Pi runs Windows 10 IoT as its operating system and will boot directly into the Smart Mirror interface upon powering up.

B. Microcontroller

The microcontroller is responsible for managing the information received from the mirror's various sensors. Our custom designed MCU features the ATmega328p [1] chip off of an Arduino Uno. We decided to feature the ATmega328p for economic reasons, since we already had an Arduino Uno board available for prototyping. Our design included all digital and analog pins.

TABLE I
ATMEGA328P SPECIFICATIONS

Clock Speed	16 MHz
Operating Voltage	1.8-5.5 V
EEPROM	1 KB
SRAM	2 KB

C. Temperature & Humidity Sensor

The DHT11 [2] sensor is able to measure the relative humidity (RH) and temperature. This digital sensor is commonly used for inspection of equipment and humidity regulation. This particular sensor was low-cost and had a precise calibration.

TABLE II
DHT11 SPECIFICATIONS

Response time (1/e63%)	6 to10 Sec
Operating Voltage	3.5-5.5 V
RH Accuracy	+/- 5%
Temperature Accuracy	+/- 2 ○C

D. Light Sensor

Photo resistors will be used for the light sensor. The photo resistors of interest are the GM55 series. The dark resistance for this series of sensors is relatively high compared to other sensors, typically between one and ten mega ohms. The circuit for the light sensor will be a simple voltage divider using said photo resistor. As the brightness of the room increases, the resistivity of the photo resistor will drop.

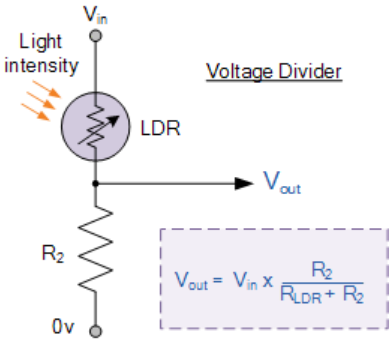

Fig. 1. Light Sensor Voltage Divider Diagram

E. Motion Sensors

The HC-SR501 is based on infrared technology, mainly used for automatically sensing various electrical equipment. This sensor will be used on the side of the mirror to detect any of the users swiping gestures from top to bottom. This close range sensor will be programmed to perform certain actions within the user interface when activated. For instance, swiping from top to bottom over the sensors while music is playing will pause the music to allow for voice commands. They can also be programmed as UI element selection and navigation tools.

The second motion sensor is the GP2Y0A02YK0F. It is a distance sensor that is composed of a combination of a position sensitive detector, infrared emitting diode, and a signal processing circuit. This sensor will be used to determine the distance between the user and the mirror.

TABLE III
HC-SR501 SPECIFICATIONS

Sensing Range	7 m
Operating Voltage	5-20 V
Output Voltage	3.3 V
Temperature	- 15 to 70 ○C

TABLE IV
GP2Y0A02YK0F SPECIFICATIONS

Sensing Range	20 to 150 cm
Operating Voltage	4.5-5.5 V
Output Voltage	0.25 to 0.55
Temperature	- 15 to 60 ○C

III. SYSTEM OVERVIEW

In order to obtain a better understanding of Smart Mirror, refer to the block diagrams for an effective overview of how we merge software and hardware design.

A. Hardware Design

This project has two major components that will be working together to accomplish the tasks required. The Raspberry Pi 2 will control all software components and handle all aspects of displaying information to the user. The MCU will handle all sensory inputs excluding audio to interact with the environment in the most helpful and convenient way possible. Shown in Figure 2 is a block diagram of the overall system.

The MCU will interface with a number of sensors to monitor the environment around the mirror including temperature, light, and motion. Each sensor will be mounted onto the mirror appropriately to transmit information to the MCU before being forwarded to the Raspberry Pi for analysis. The Raspberry Pi will output video to the display and receive voice commands from a microphone. Both the Pi and the MCU are 5V-powered.

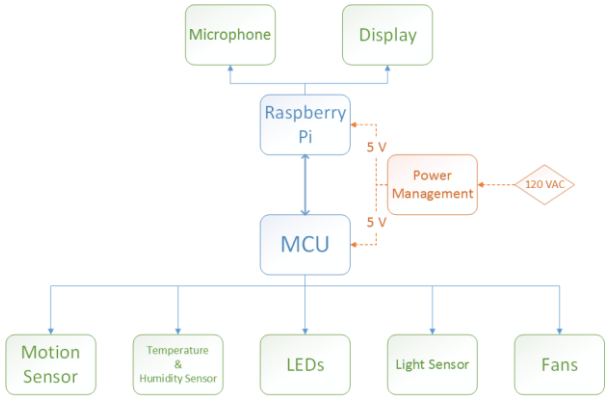

Fig. 2. Smart Mirror Hardware Block Diagram

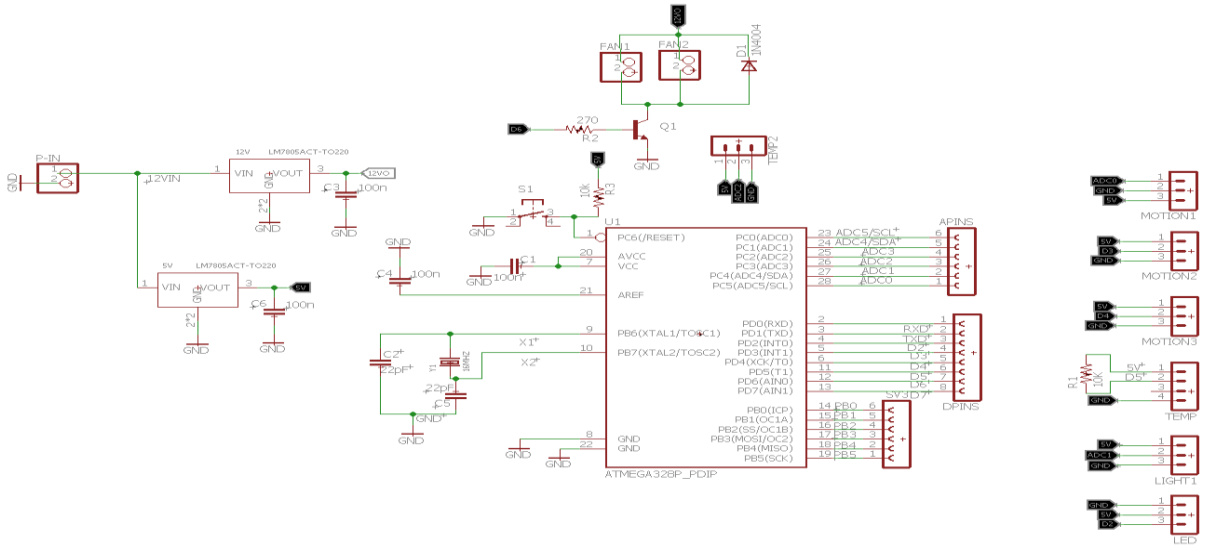

Fig. 3. Custom Microcontroller Schematic

B. Software Design

The software block diagram provides an overview of all the various software interactivity and how they manage data input from the MCU. As seen in Figure 4, the MCU will receive motion data from the distance sensors and then forward the information to the gesture processor in

Order to perform the necessary action. Audio data from the microphone is sent to the voice processor which is linked to all of the software modules to provide persistent voice recognition. Utilizing an MVVM software architecture, the software modules and main module are set up as Views. The Main View outputs the user interface to the display.

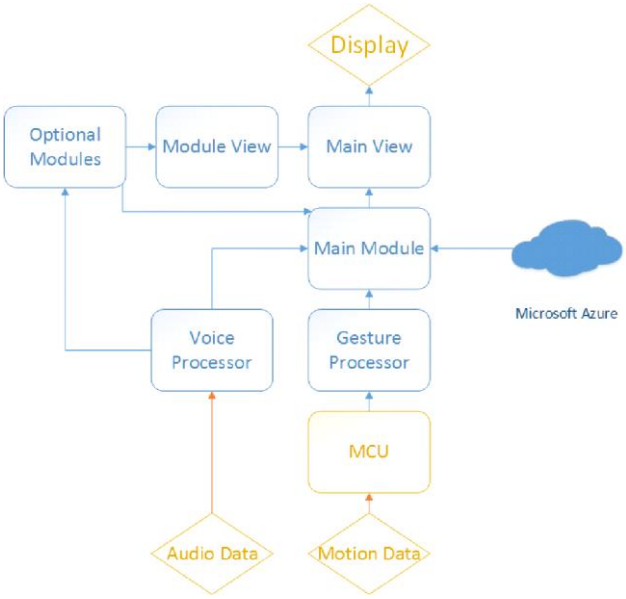

Fig. 4. Smart Mirror Software Block Diagram

IV. HARDWARE DETAIL

This section provides a more detailed look into the hardware aspect of the smart mirror which is comprised of the primary computer system, the microcontroller, and its various sensors.

A. Primary Computer System

As mentioned earlier, our primary computer system is the Raspberry Pi 2 Model B. This microcomputer was selected because of its small form-factor, adequate performance, and low-cost. Moreover, it is capable of running Windows 10 IoT Core which is necessary for our Smart Mirror's UWP platform. The Minnowboard Max was also considered; however, the Minnowboard was priced higher with negligible returns in performance. With the Pi's Quad-core ARM7 processor and 1GB of SDRAM, it is more than capable of being the powerhouse of Smart Mirror and interfacing with the MCU.

B. Microcontroller

As previously mentioned, the ATmega328p will handle all the input from the various sensors placed around the housing of the mirror. It will be powered from a 5 V voltage regulator.

C. Sensors

The mirror will have 3 motion sensors around its housing. One will work as a proximity sensor on the façade of the mirror and will be used to turn the display on and off automatically based on whether there is motion detected in the room. While the other two will be placed on the side of the mirror to receive gestures as a simple form of gesture control as mentioned in Section V, subsection C. Besides the motion sensors, the mirror will use a temperature and humidity sensor to ensure the electronics inside the mirror will be at operational temperatures.

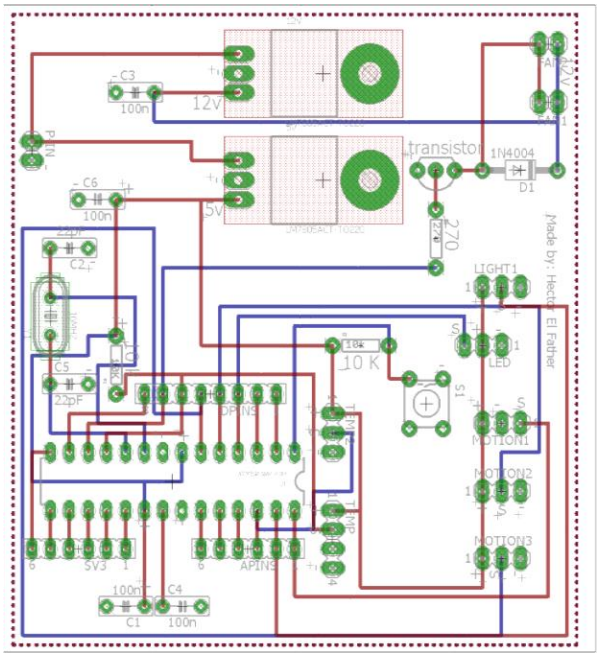

Fig. 5. Custom Microcontroller Eagle Layout

The mirror will also utilize a light sensor to determine whether the user has turned the light in the room on. This will be used in conjunction with the front facing motion sensor to turn on an LED for the user for low light situations. If the mirror senses movement but does not sense the light turning on, it will turn on an LED on the mirror to provide the user with a bit of light.

V. SOFTWARE DETAIL

This section provides a more detailed look into the software aspect of the smart mirror which is comprised of the different software modules.

A. Software Architecture

Due to implementation of the software on Windows 10 IoT which requires the new Windows software platform; the Universal Windows Platform (UWP). The project will be constructed utilizing the standard UWP design architecture; MVVM. The MVVM, Model-View-ViewModel architecture is derived from the basic Model-View-Controller (MVC) pattern. One of the core ideas of the MVVM model is separation of development of the graphical user interface (GUI) and the back end logic and data implementation. The 'View' component refers to the presentation layer and specifies the user interface. The view layer is generally specified by XAML and elements are bound to objects in the view model layer. The 'View Model" component is a thin layer used to convert information to a format presentable in the GUI which is then bound to the view. The 'Model' component of MVVM is a large layer that encompasses all business logic and data manipulation code required by the system.

This implementation allows designers to simply understand XAML, or to work with the Visual Studio Designer to develop the display without the need to burden software developers with user interface coding. This also allows software developers to make changes to the code behind the XAML without effecting the display. As long as the new code extends the same interface as the existing ViewModel, the biding between the View and ViewModel will still work.

The majority of the program will be written utilizing the .NET framework and C# for the Model layer of MVVM. UWP programs generally utilize XAML to describe the View with the thin ViewModel layer behind it being written in the same language as the model, in this case C#. The majority of the data that will be displayed on the mirror is coming from third party sources and it will thus be required to implement HTTP GET requests to a number of third party APIs. We will implement a RESTful interface for each of these third party systems.

The software will utilize a modular system to implement all the different pieces of data required. Each module will be self-contained and should be able to be enabled or disabled with no effect on other modules. The only required module will be the main module which will handle all general display status and layout of the applications on the mirror display. This main module, will maintain the state of the mirror at all times and all display elements will be controlled by this module. It will also contain references to each submodule in order to keep track of the status of each application such was whether or not it is currently focused or running some specific functionality for the user. This class will be the heart of the application and will essentially manage everything going on. All sensor information, except audio, will be fed into this class as well to be sorted out so action may be taken.

B. Voice Controller

The voice controller is the primary component used to interact with the smart mirror. Due to this, its implementation details are vital to the function of the mirror overall. The requirements of the mirror's voice commands need to be thoroughly considered and tested to ensure that they work as desired in an efficient, user friendly manner.

As people tend to phrase similar requests in different ways, our mirror needs to understand what a user is attempting no matter how they say it. To give a set list of commands required for each application will simply have users needing to continuously refer to the list of commands

which will simply bog the user down in minutia making the mirror ineffective in its use of saving time and providing easy access to necessary information. Thus, each command should be implemented in a number of ways to account for different types of speech and phrases used. This is to ensure that each user finds interaction with the mirror intuitive and unencumbered. To implement this in the best way possible would be to utilize a neural network that learns phrases different users use. Unfortunately, for this to work correctly, a large user base, or large data set, is required. Since we do not have access to either, for this project we have decided to go with a different method. The windows speech platform allows you to specify grammar files with keywords to listen for. We will utilize this to set up a list of keywords to be expected and will allow us to cover a large number of different phrases for the same command.

The implementation of the voice recognition software will utilize the Microsoft Speech Recognition API for all voice command processing. There will be a specific module that listens for voice command, the Voice Recognition Module. This module will be run asynchronously and listen

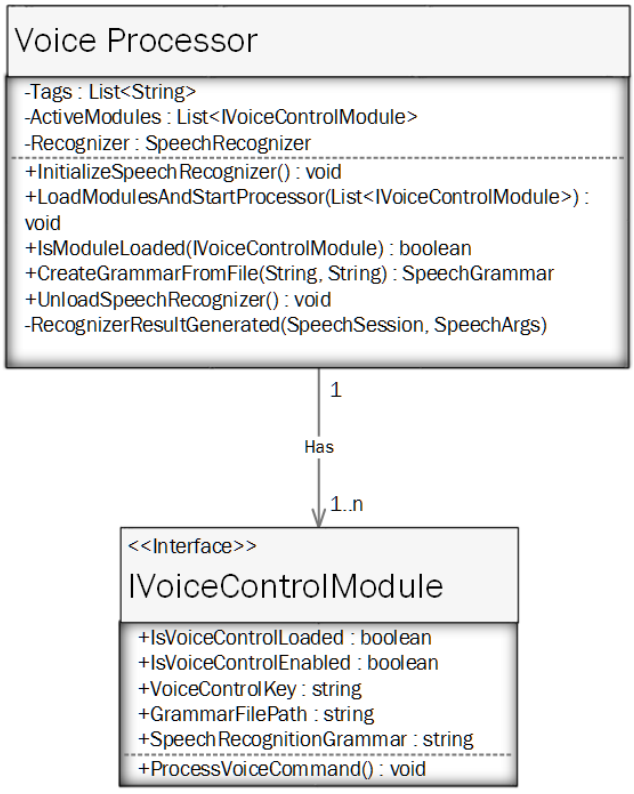

Fig. 5. Voice Process Class Diagram

for voice commands then handle them appropriately. For this to work seamlessly with any module we are implementing an interface that each module is required to extend if it is to use voice commands. This interface, IVoiceControlModule, will specify all required methods required by the Voice Recognition Module. This will also allow the Voice Control Module to hold a list of the voice controlled modules. This is important because when the voice control module receives a command, it will check which grammar file the command came from.

This grammar file will be able to be linked back to a specific module and the Voice Recognition Module will then pass the data to the correct module. This will allow the voice recognition module to continue listening for more user input while the submodule handles its command. A UML class diagram demonstrating this point is shown in Figure 5.

C. Gesture Controls

For the Smart Mirror, gesture controls were a heavily discussed and debated feature. In the initial design it was intended to have full gesture control support utilizing a Microsoft Kinect or a Leap Motion. Due to the decision to utilize a Raspberry Pi 2 as the main computer for the project, both of these option were eliminated. This was determined to be acceptable as when a user is utilizing a mirror their hands tend to be busy anyways thus the loss of gesture control is not an issue. However, it was decided that some very simple gesture controls would be implemented due to the music playback capability of the Smart Mirror.

Since Kinect and Leap Motion were not possible due to the hardware limitation of the Raspberry Pi, it was decided to implement a system of gesture control with three possible commands by utilizing two motion sensors. Using two motion sensors we will be able to perform three options based upon the timing at which the sensors detect motion. If a user puts their hand straight in front of both sensors, they will sense motion simultaneously and that will be one command. The other two commands will be implemented as a waving motion up or down, which will allow the microcontroller to see if one sensor was triggered before the other and interpret which type of swipe was performed, up or down.

D. User Interface

The goal with the user interface is to provide as much information as possible with minimal user interaction. It is important to maintain the right balance of information and unobstructed mirror space, specifically the upper and center areas; it is in these areas where your eyes naturally fall. In order to support all of the various software modules while maintaining an appropriate amount of mirror space. Due to this, we have put careful consideration into how the elements are arranged on the mirror. All the main display data will be limited to the corners of the mirror while the center areas on the top and the bottom will have optional

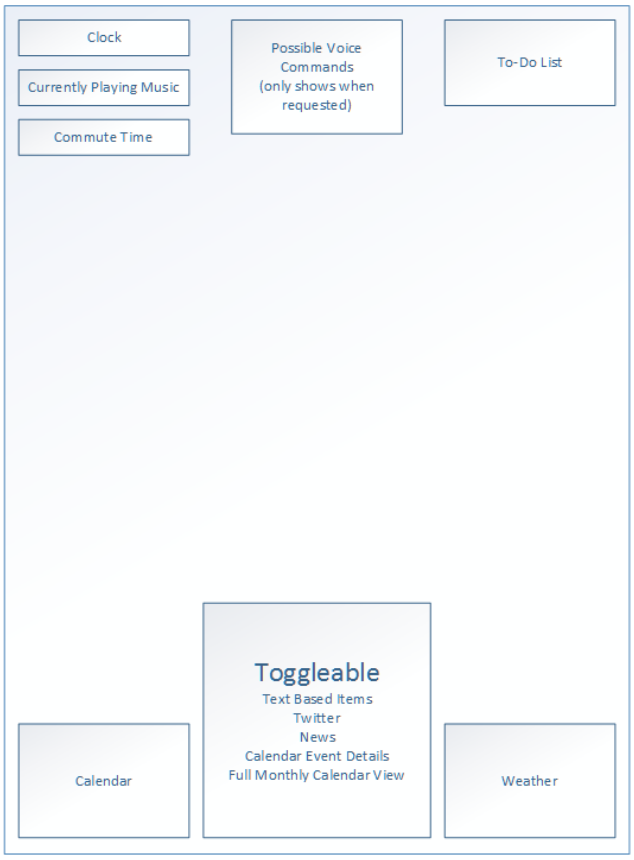

Fig. 6. User Interface Layout

display based upon user interaction. It was determined that if the user is asking the mirror for additional information, it is acceptable to utilize a bit more of the valuable mirror space. The decided upon layout for the mirror is shown in Figure 6.

E. Software Module Design

As mentioned earlier, Windows 10 IoT Core allows you to specify a program to boot upon startup. When you power up Smart Mirror, the main module is loaded and the user interface appears consisting of the date, time, and weather. Each software module will be self-contained and have no dependency on other modules besides the main module and, if it uses voice controls, the voice recognition module.

While each module has a different functionality, they all have a similar list of required classes. Every module will have a Model which will be the class that will handle all business logic required and any manipulation of data. Almost every module utilizes data from a third party source. This data will be fetched using HTTP GET requests to retrieve the information. All code related to accessing third party data will be in its own class. This class will implement only a single public method per type of data the module requires. For instance, the weather module service class will have methods to retrieve current weather, today's weather, tomorrow's weather, and the week's weather. These method will be required to parse the received data and return the data in the form of a specified POCO (Plain Old C# Object). This implementation will allow us to change the third party API of any module with little to no effect on the functionality of the module itself. Each module will also have a Voice Processor class, this class will implement the IVoiceControlModule interface mentioned in the voice control section. The voice processor class of each module is the class that will receive the commands from the Voice Recognition module. Once the voice processor has the command, it will determine what type of voice command was received, and pass this information to the main model class to handle the logic required to execute the voice command. Each module will also contain it's own individual view and viewmodel. The module will specify the layout of its own data within its view and this will be referenced by the main view and placed into the correct location on the mirror display. Finally, each module will contain its own helper POCO's for manipulation and handling of data.

Although these modules could likely be implemented in a simpler manner, this extra bit of work allows for easy changes in the future. Modularity in code is extremely important as it makes developers lives easier in a number of ways. Firstly, if the pieces of code as individual as possible, changes can be made easily with less potential effect on the rest of the software. Also, with independent, well segmented code like this, once the developer is familiar with the architecture, it is much easier for them to follow what is going on and to know exactly what to expect from each class.

Shown in Figure 7 is a simplified UML representation of a module and its interaction with the rest of the system. The module must implement the IVoiceControlModule, which requires a grammar xml, if it wishes to use voice commands. It is also required that every module must be referenced by the main module which will handle all management of the display itself.

F. Software Modules

The clock module is persistent and will always be visible in the upper left corner of the display. To ensure everlasting accuracy, the time module reflects the system time which is synced with an NTP server managed by the OS.

The weather module will be located in the bottom right corner of the display and will provide the user with the current temperature at all times. The weather is denoted

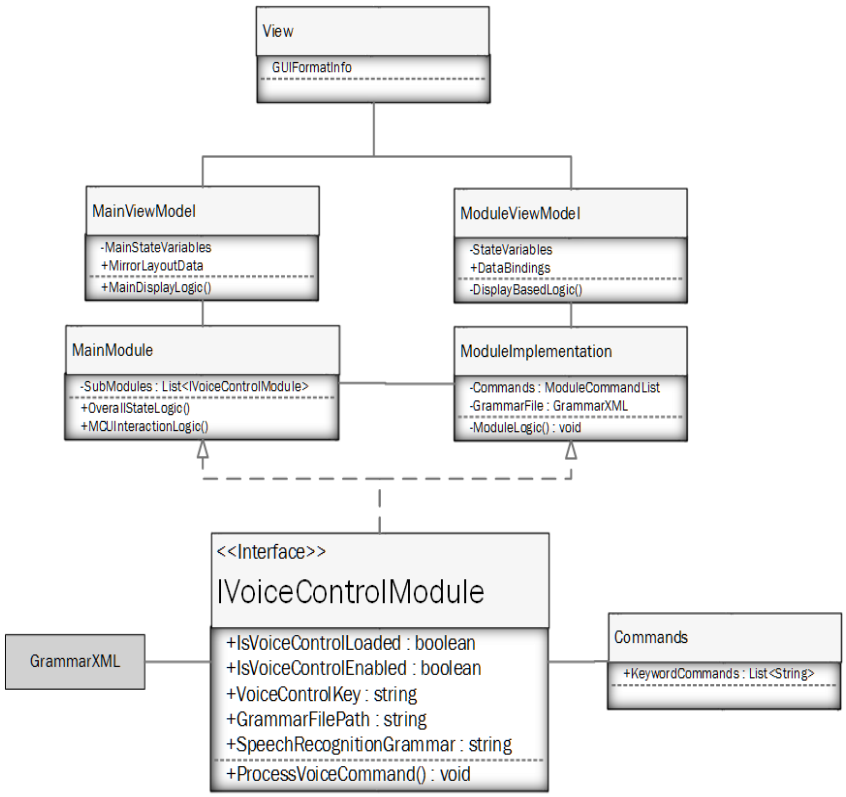

Fig. 7. Module Class Diagram

Fig. 8. Mirror Housing

with the temperature along with a corresponding icon. Weather information will be provided by OpenWeatherMaps which offers a number of APIs with different weather information. All weather information will be displayed in the same location on the mirror but only one type of information at a time. Voice commands will be created to allow users to allow the user to request that the mirror show four different types of weather reports. The first will be the current weather, the second will be a 3 hour forecast for the current day, the third will be a 3 hour forecast for tomorrow, and finally the last will be a 5 days forecast showing some basic information about each day. In the morning, the current day's weather will be shown while in the evening, the next day's weather will be shown. This will allow the user to glean the most useful information at a glance while checking others with a simple command.

The calendar module will provide users with a look at any events they planned for that day. A month view can be requested with a voice command and days that contain events will be highlighted.

The news module will be provided three news headlines. The source of the news is the CNN Top News RSS feed. Users will be able to request different preset news categories such as World News, U.S., Politics, Technology, among others.

The to-do-list module will utilize the Todoist API to provide the user with task managing and reminder functionality. Todoist also supports task list syncing to provide a seamless multiplatform experience.

The Smart Mirror will offer a commute time module. This module will require that the user enter information at mirror initialization, the first time it is booted up. The user will be asked to enter their home address and work address as well as the time they arrive at work each day. The module will utilize Google Maps Distance Matrix API and determine what time the user needs to leave in order to make it to work on time. This will allow the user to recognize and account for any delays in their normal traffic route before even leaving the house.

VI. Mirror Housing

The way the mirror works is by utilizing a one-way mirror which is placed directly in front, flush with the screen of a television. This will allow light to shine through the mirror wherever the television is lit but it will simply reflect like a mirror wherever there are black pixels on the display. To accomplish this, the one-way mirror will be set

into the face panel slightly and the television will be mounted directly behind it.

The housing of the Smart Mirror will be designed in a way that is cost effective but also displays the mirror and software in the clearest way possible. The housing will be built from wood with a face panel made of four miter cut boards. The wood will be done with a burned finish then stained for a nice presentation that people might like to have in their homes. It is important that this is presented as a quality display piece otherwise most people would have little interest in having it in their home. All sensors will be as hidden as possible to present the image of a regular mirror.

VII. Conclusion

The Smart Mirror project was born out of the desire to introduce smart home technology to a piece of furniture that had not yet been thoroughly explored, the mirror. Smart Mirror is our attempt at transforming mundane tasks, such as checking the weather or your calendar, into a high-tech and engaging experience. Moreover, Smart Mirror is capable of improving your overall time management by consolidating all of the information you need in the morning or night into a single user interface. Built on the Universal Windows Platform, Smart Mirror provides a unique and intuitive user interface that is achieved by harnessing the power and flexibility of the Raspberry Pi. By combining a microcontroller with various sensors and clever programming, Smart Mirror is able to intelligently wake itself up or provide a guiding light in a dark room. The overall cohesion between the hardware and software adds up to a solid prototype that is indicative of where smart home technology is headed in the years to come.

Acknowledgement

Justin would like to acknowledge Steven Barkdull, without his mentorship, my code would be much less organized, clear, and concise. Without his teaching, I would understand little of how to efficiently architect software for reusability and modularity. I would also like to thank him for teaching him how to write good if statements.

Justin would also like to acknowledge John Vaccariello who forced me to learn that detailed planning of a software project is more efficient, and produces better software, than jumping straight into coding.

References

[1] Atmel Corporation (2016). Retrieved November 2016, World Wide Web: http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_datasheet.pdf

[2] Adafruit (2016). Retrieved November 2016, World Wide Web: http://www.micropik.com/PDF/dht11.pdf

The Engineers

Michael Trivelli is a senior at the University of Central Florida and will be receiving his Bachelors of Science in Computer Engineering in the Fall of 2016. After graduating, Michael plans to continue working at the Orlando Utilities Commission in the NERC Standards Compliance department where he is programming a Raspberry Pi to monitor the electric grid networks.

Justin Gentry is a senior at the University of Central Florida and will be receiving his Bachelors of Science in Computer Engineering in the Fall of 2016. After graduating, he aspires to work for a large software corporation such as Microsoft or Google.

Hector Zacarias is a senior at the University of Central Florida and will be receiving his Bachelors of Science in Electrical Engineering in the Fall of 2016. After graduating, he plans to continue working at the Orlando Utilities Commission where he plans transmission systems using Siemens energy software.