BRADLEY
University

Internet of Things Information Display

Project report

Benjamin Daszkiewicz
Jacob Nading

Advised by Aleksander Malinowski

Abstract

The Internet of Things (IoT) is a rapidly expanding field of embedded computing technology. Integration of this technology can be used to increase quality of life in a multitude of ways. This project applies an IoT approach to communication between a professor their students in the form of an interactive calendar, weather, and announcement display. This document provides a detailed report of design, work to date, future considerations, and other information relevant to the project.

In its second year of development, the project was taken on by Benjamin Daszkiewicz and Jacob Nading. The first year of development was done by Jason Morris and Cole Lindeman in Fall of 2016. Dr. Aleksander Malinowski of Bradley University's Electrical Engineering Department has advised the project since its inception.

Contents

Introduction

Scope

The scope of this project and report is an Internet of Things (IoT) Information Display which interactively displays a collection of information which a professor wishes to share with their students. The report will lay out a series of functionality requirements and describe efforts to date made to satisfy these requirements.

Description

The project was begun in Fall of 2016 by Mr. Jason Morris and Mr. Cole Lindeman. Their "IoT Smart Calendar" brought ease of access to students looking for Dr. Malinowski's availability and office hours. In its second year of development, the IoT Information Display does more than build off of last year's model. It takes the basic idea and recreates it using a more concise set of technologies to meet functionality requirements.

Functionality revolves around a Google Calendar managed by Dr. Malinowski's Google account. It also uses Twitter to allow easy delivery of announcements to the display, and therefore, students. Weather is pulled from a free weather service, openweathermap.org, and finally, advertisements of the professor's future classes are displayed in a "screensaver" mode.

The Display is a 13" touch screen monitor which hangs outside the professor's office door and allows students to see the professor's Google Calendar for the day and near future, the current and forecast weather conditions, recent announcements from a dedicated Twitter feed, as well as advertisements for upcoming courses taught by the professor.

Other functionality was included in the proposal, but was ultimately abandoned in this project iteration due to time constraints. The primary such feature was a GPS geofencing feature that would alert Display viewers when Dr. Malinowski was available, but not in his office. These will be discussed further in Changes from Fall 2017 Proposal Design.

Functionality Requirements

The IoT information display includes the following:

- Daily calendar data
    - Displays professor's schedule, synchronized automatically from their Google account
- Short memos or announcements
    - Professor is able to easily publish announcements to the display remotely from their personal device via Twitter
- Advertisements for the courses taught by the calendar owner
    - Display contains automatic timeout feature that displays relevant course advertisements when its other functions are not in use
    - Device wakes from advertisement timeout when engaged by a user
    - Advertisements are also available on demand from the home screen

- Current and forecast weather data
    - Display users can access current and future local weather data

Modes of Operation

The varying modes of operation for the display and the transitions between them are discussed here. The Information Display does not currently have a low power mode, but does have a

Power Off

All power is shut down to the monitor and the Raspberry Pi. The Raspberry Pi can be shut down via software or by disconnecting power from the Pi. Two of the GPIO pins can also read a switch across them to trigger Power Down and Power Up modes. The monitor has a switch on it that is accessible via a hole in the mounting frame. A stylus inserted into the hole will touch the monitors capacitive power switch.

Power Up (transition)

Power is resupplied to the Raspberry Pi using GPIO switch or by re-inserting the 5V micro USB plug into the Pi. The OS will boot and will either load the operating system or will boot the Display application, depending on whether the application is stored in the /home/pi/.bashrc directory or elsewhere.

Operating System

If the application is not stored in the “run on boot” directory, the Raspberry Pi will boot the operating system and the application will need to be run manually from the GUI or from the terminal. SSH access to the Pi will allow the application to be started from a third-party computer.

Home Page

If the Display application is located in the run-on-boot directory when the Power Up transition completes, it will load the GUI application and display the home page including the Calendar for that day, the current weather conditions, recent announcements, and a button for advertisements. Touching one of these features on the home page will load an overlay that shows more information. These could be considered sub-states of the Home Page state of operation.

Advertisement Loop

After an inactivity timeout of ten minutes on the Home Page, the application will show full-screen images pulled from a directory on the Pi. The advertisement loop can also be activated from the Home Page state by touching the “Advertisements” button. This state loops until the program detects that a user has touched the screen. It then transitions back to the Home Page

Power Down (transition)

Power down transitions from the Operating System mode of operation to Power Off. The application requires no special method to close. Once the device is back to Operating System mode, the Pi can be powered down via GUI or terminal. The monitor must either be powered down using the capacitive switch accessible using a stylus, or by removing power from the display.

Additional Functionality

While it is outside of the scope of the current project, it is worth mentioning that other ideas for the IoT Display have been discussed and requested by Dr. Malinowski and Dr. Imtiaz. Future direction for the project is discussed in detail.

Review of Literature and Prior Work

From the prior project development team, the main focus of the IoT Information Management Display is mostly the same as what it is now, but build on a more stable set of technologies. Based on the previous iteration of the development of the project, the focus entailed a wall-mounted smart calendar, interface with sensors, and communication with the internet. The current iteration includes displaying advertisements and calendar information to passing by people, and access to the professor's announcements. The focus on sensor interface was left out of the current iteration of the project. The advancement of this goal previously didn't get as developed and stable as originally intended. Therefore, a few of these features are disabled to allow the better functionality of the device. Communication with the Internet is essential to how it works and is a currently well-developed process in the iteration today.

The past development team also did a bit of research on two devices that were similar to the aspects they were looking for in designing the project. They felt like an even greater idea for the more updated and advanced project ideas and thoughts that are ongoing today. One device, the DAKboard, is very similar to the eye-appealing aspect that the current project wanted to incorporate. DAKboard consists of a customizable wall display that can also show pictures, calendar events, and weather. Some of these concepts are very similar to what went into the IoT Display design; a customizable interface where a user can decide what they want to be synchronized and displayed in an easy-to-read format. Additionally, in the DAKboard design, everything is done through a web interface, which was the direct design of the IoT Information Management Display in its first iteration [1]. There is also another previously researched display, the Raspberry Pi Framed Informational Display, that specifically uses a Raspberry Pi for the whole interface [2]. It displays a Google calendar along with local weather data, which is incorporated in the current iteration of the project. This display, although having a similar process design to the previously completed IoT Calendar, has a bigger emphasis on displaying everything in a better layout as well as making it look more appealing and organic. There have been multiple similar devices designed with very similar ideas in mind. Another example of a design that with an interesting concept was an informational display within a mirror [3]. Because of the excessive, consistent use of a mirror, information can constantly be reminded to a user, making remembering certain tasks exponentially easier.

Applicable Standards and Patents

Standards to Consider

Any standards to consider have been met because the project uses entirely COTS (commercial off-the-shelf) parts. Standards that would otherwise have to be considered are internet connection protocols for connection of the IoT and wireless transmission protocols that have already been considered by the manufacturers of the pats used.

Patents for Similar Ideas

There are a few patents for similar ideas to the IoT Information Display. Some of these are closely related to the IoT Display, as the design concept is pretty general. Having a device display information is too broad an idea to have any specific patent on it. There are a few more in-depth ideas that go into much more detail about their device and its usage.

- A display device and content display system
  - https://patents.google.com/patent/WO2016061626A1/en?q=smart&q=touchscreen&q=information&q=display&q=raspberry&q=pi
- Raspberry Pi based smart device control apparatus and control method
  - https://patents.google.com/patent/CN106789459A/en?q=smart&q=touchscreen&q=information&q=display&q=raspberry&q=pi
- Smart interactive billboard device
  - https://patents.google.com/patent/US20050021393A1/en?q=smart&q=device&q=display

Subsystem Level Requirements/ Changes from Prior Models

Hardware

Figure 1 shows a connection diagram of the very simple hardware connections necessary for the project. The Raspberry Pi and display passes data back and forth via HDMI (to display) and USB for touch data (to Raspberry Pi).

Software

In brief, the technologies, languages, and modules used are:

- Python 3.6 scripting language and suppost modules
  - appJar graphics module, built on Tk and Tkinter
  - datetime module
  - urllib2 HTTP handling module
  - PIL (Python image library)
  - Os module
- Google Calendar API module
- Twitter API module module
- OpenWeatherMap.org weather API
- Sunrise-Sunset.org API

The GUI is the central component of the project. One of the main focuses of the project was to convert the original Display software, which was in HTML and PHP scripts that run in an instance of Firefox, to a graphically rich application written in Python using the wxPython graphics module. However, the wxPython module proved to have too steep a learning curve for the time allowed to complete the project. Therefore, we changed to appJar, a high level GUI module that allowed for much more rapid development at the cost of some graphics flexibility.

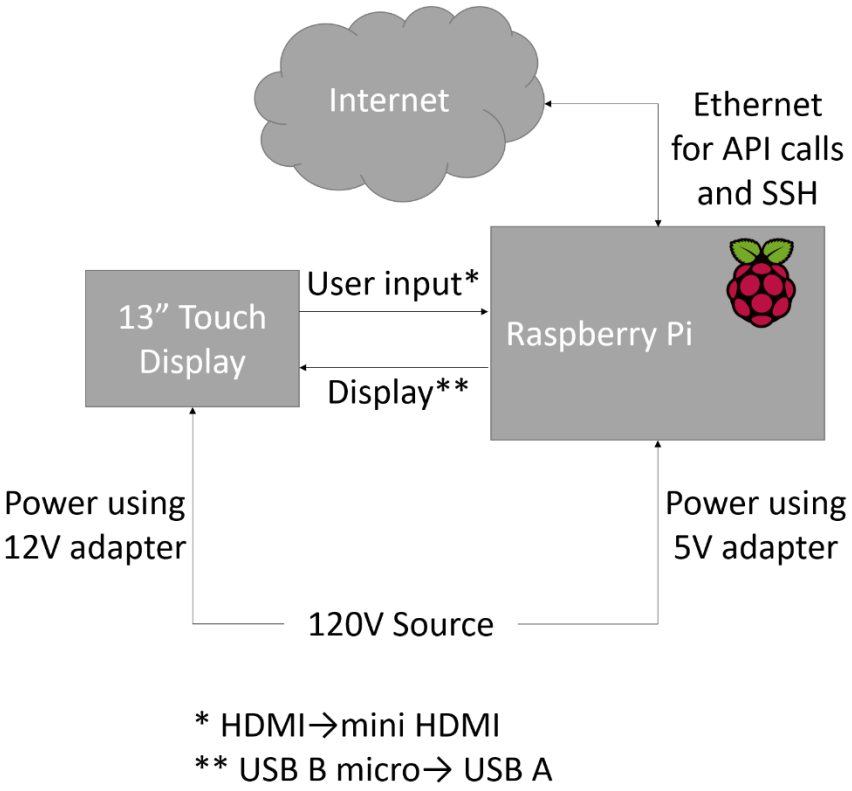

Figure 1: Hardware configuration diagram

As detailed in the Figure 2, the GUI process diagram, most features of the app will be visible and accessible from the home screen. This contrasts the present design, where users must scroll through pages or navigate a menu to see content. Not all calendar, announcement, or advertisements data will fit on the home screen, however. Abbreviations can all be shown on the homepage, and the full data is accessible with a touch.

Calendar

The calendar was the heart of the original design as indicated by the title of last year project. We looked to make some improvements from last year's calendar. Instead of directly displaying the calendar on a page in a web browser, as is the case presently, we made use of Google Calendar's rich Python API to pull the data itself from the calendar which is then embedded in the GUI in full color.

One particular feature that Dr. Malinowski was looking for in the calendar was the ability to use the color coding feature available on Google Calendar's native web platform. For instance, when viewing calendars there, events can be color-coded by each course a professor may teach. When importing the calendar into a custom browser layout as was done originally, the events display in one uniform color and do not keep their color coding. We were able to pull color ID data from the original event settings to implement a more colorful and useful calendar.

Announcements

In today's world, one of the easiest ways to update your status is through social media. Most people are already accustomed to using such technology and do so anyways. Harnessing the tools given to us by

Twitter, we created a system that allows the professor to post messages and updates quickly and easily that is not only be visible at the IoT Display but is also be accessible through students' personal devices.

Setting up a Twitter account for the professor to use for their classes keeps it separate from their personal account. Then, they can post messages that can be updated to the display periodically using Twitter's feed-pulling functions in the API. If applicable, students can also follow the professor's account on their own Twitter accounts to receive the messages to their phones.

Availability Status

In place of the original geofencing functionality of the project, there is a status availability light. This light is updated using the Twitter announcement functionality. The professor can Tweet predetermined phrases that allow him to change the availability status between "Available," "Busy," "Away," and hidden.

Advertisements

Special technology did not need to be applied to the advertisements. The advertisements remained as they are were in the original iteration of the project. They are presented as static images that can be uploaded to the Raspberry Pi file system and displayed from there by appJar using its built in timer interrupt functionality.

Weather Data

The weather data functionality of the calendar was improved upon from the previous project. Modern devices often show the weather at a glance from the home screen, and we have implement similar convenience, integrating the weather data in an interactive way.

Originally, data was pulled from the National Weather Service via a modifiable URL that returns different data depending on how it is modified. The information is accurate and reliable, and this method was kept for the forecast to meet the minimum requirements of the project. However, the homepage uses the OpenWeatherMap.org API and urllib2 in Python to create a graphically righ and dynamic at-a-glance weather display.

The center of the home screen is a large icon that represents current weather conditions at that time. The page background is also representative. As the day/night cycle changes and the weather changes, the icon also changes. For instance, once the sun is set, the "clear weather" icon changes from Sun to Moon and the background from a clear blue sky to a clear starry one. If it begins raining midday, a "partly cloudy" icon and puffy cloud background changes to rain clouds. Current temperature data is also displayed and clicking on the large weather icon is how more detailed weather information is accessed.

Software Development Life Cycle Diagrams

Part of the project's original design process included creating software development lifecycle diagrams. These diagrams acted as guidelines for design of the scripts throughout the project. Although the plans were modified as the project progressed, the diagrams have been included in the final report as a reference for how the project is different from the original design in its current state.

Data Flow

The Context Data Flow Diagram shows on a high level how data and what kind of data move between modules with the GUI home page main loop being centric to the model. Here, the functions that move data are named and tasks can be broken down by module. A Level 0 Data Flow Diagram would break down each module into submodules and display data stores along with the functions that pass data around the sub-modules.

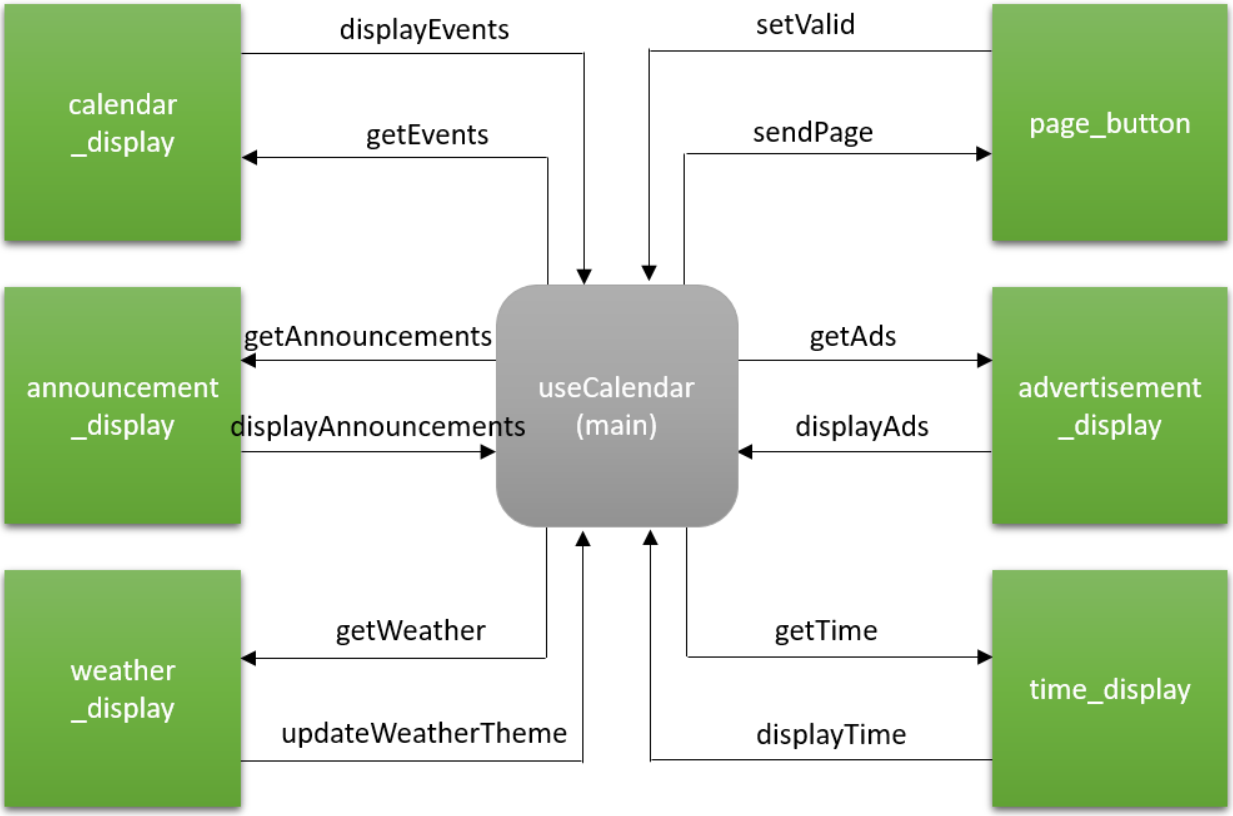

Figure 2: Context DFD for Software Modules

State Transition

Figure 3 shows a high-level state transition diagram (STD) of the software focusing on the minimum functionality requirements. It details how users will navigate from place to place in the software and gives an idea of what features will be accessible from certain areas of the software. The simple STD indicates successfully streamlined software that is user-friendly to navigate.

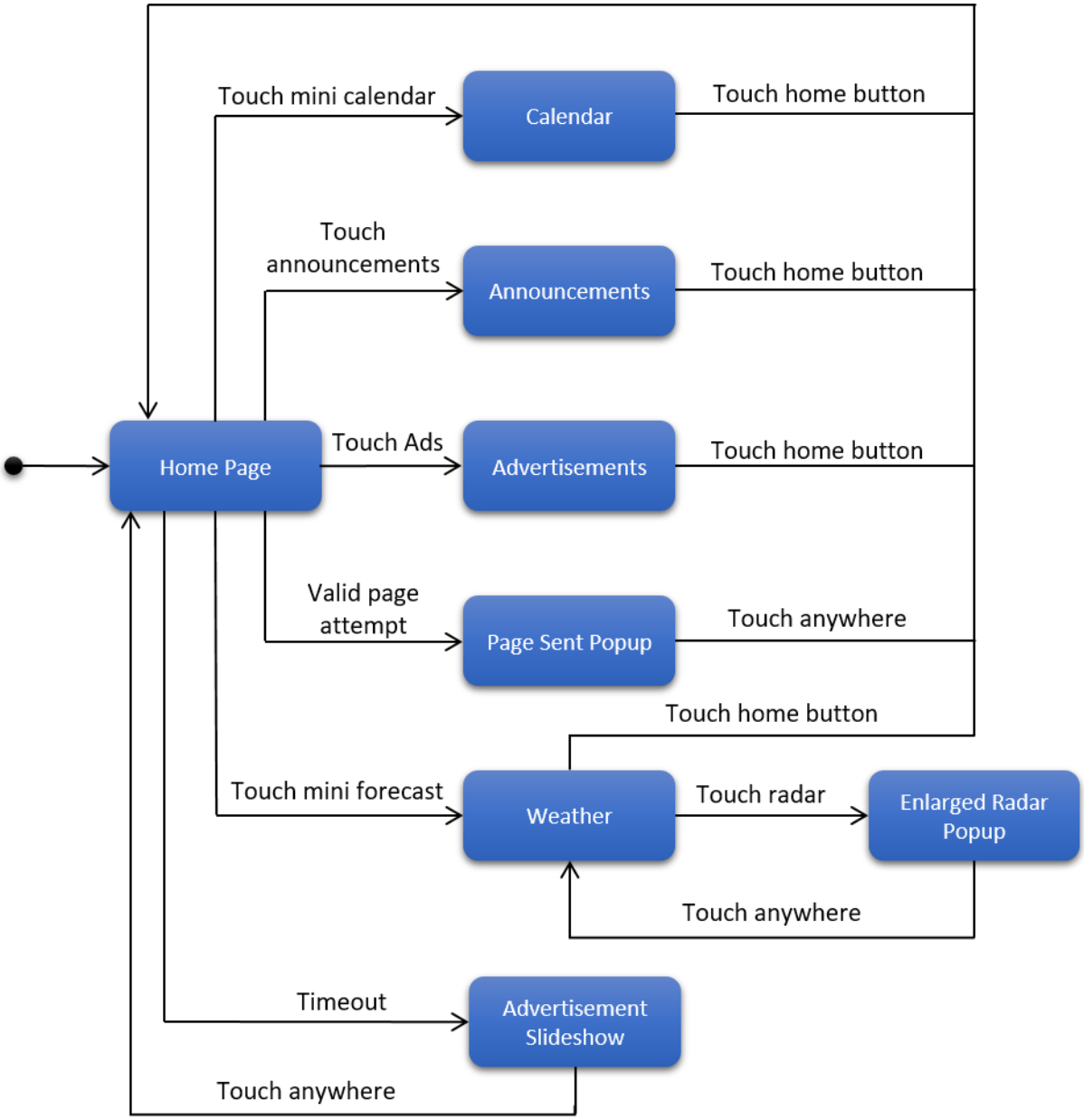

Figure 3: High-level state transition diagram for Display GUI

GUI Hierarchical Design Model

The GUI Hierarchical Design Model breaks down the visible objects on the main page of the Display. Each instance of each element is listed under the general heading. Underneath, attributes of each general element are listed. This diagram helps greatly in the creation of class objects for the GUI environment. Some of these elements exist natively in the appJar graphics library such as Image, Text, and Button. However, not all native objects are visually customizable. For instance, the Button object has limited customizability on its appearance. In that case, a new object will have to be created that allows its image to be modified. These classes can be inherited from the Image object, or can be drawn as Canvas elements.

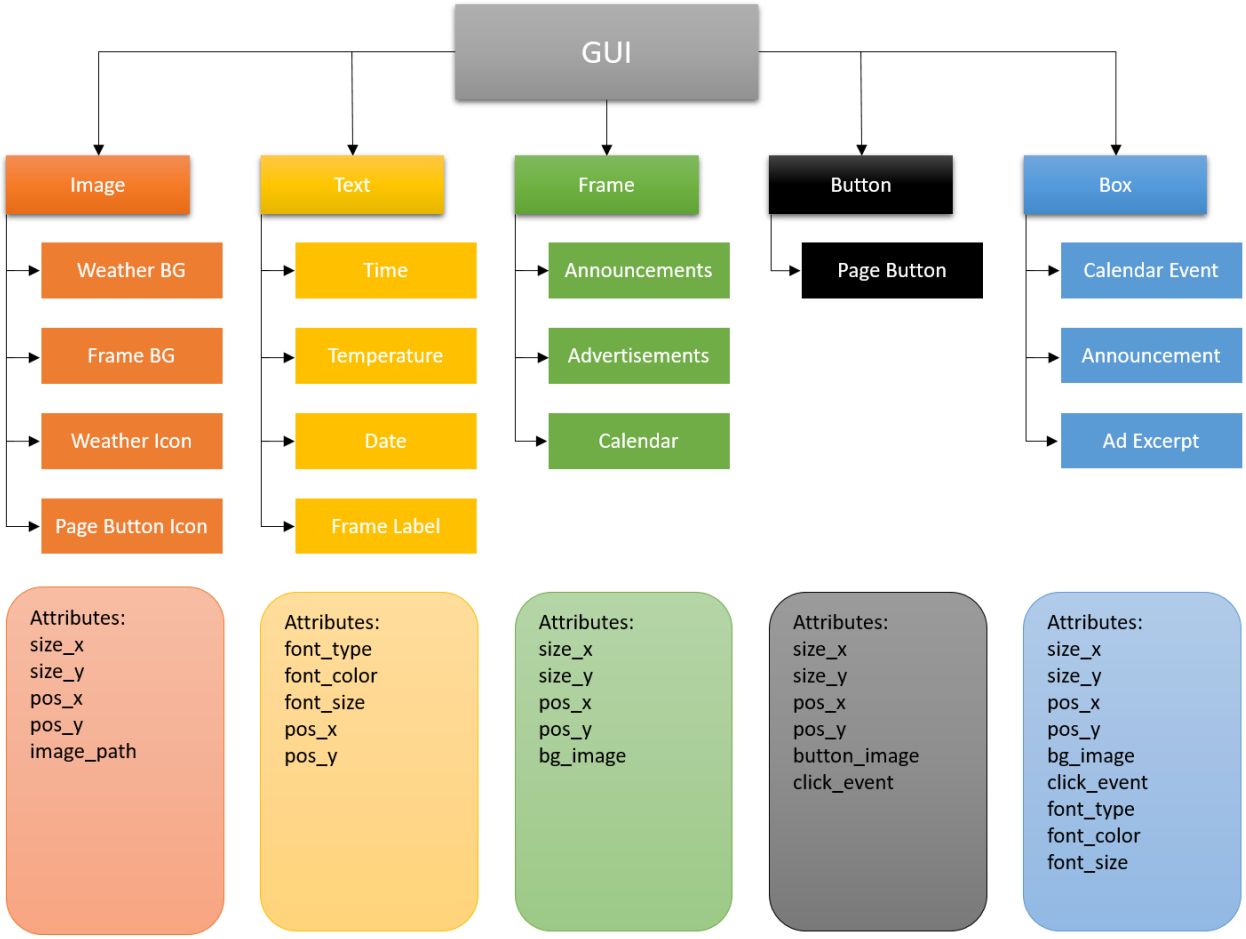

Figure 4: Hierarchical design model for GUI home page

Changes from Fall 2017 Proposal Design

The original scope of the project also included GPS geofencing technology to alert users when Dr. Malinowski was near his office. This functionality has been abandoned for a simpler solution involving Twitter. The professor can leave predetermined phrases in a Twitter message that the Twitter module will search for. Instead of displaying the message as usual, it will update the “Available” icon on the home screen. Previously, the “Available” button was to be automatically updated by cross checking the professor’s location via Geofencing from a phone app and office hours on his Google Calendar.

A lot of time was spent at the start of the Spring semester attempting to write even a simple GUI application using wxPython. Due to inexperience with Python, and wxPython's low-level approach to GUI programming, it seemed as though a new module would be needed for GUI programming. The project needed a library of objects with pre-existing member functions for event handling and binding, which is why appJar and its parent module, Tkinter, are used. appJar is a module made for teaching GUI programming in Python and is therefore very high level and useful for quickly defining the GUI process or creating a window. The sacrifice made with these high-level GUI modules is loss of flexibility and functionality. There are some things that can be done in wxPython that cannot in appJar and Tkinter that would have been useful for the project such as displaying a web page within the Python GUI and better handling of images with transparency.

System Level Implementation

Hardware

Mounting the Screen

Hardware is implemented by mounting the screen on the wall outside the professor's office using the mounting system in Figure 5. This mounting diagram is only a plan, as head lab technician Mr. Mattus has final say over the design of the mount. The mount stands the screen roughly one inch off the wall and has ventilation slits on the top and sides of this one-inch space to allow heat from the screen's power dissipation to be minimized inside the mount.

Placing the Raspberry Pi

The Raspberry Pi for Dr. Malinowski's Display is kept on a shelf at the top of the wall in his office. This puts the Pi just on the other side of the wall from the Display and allows easy connection. Wires can be run through the drop ceiling and down the wall using plastic external wall conduit that is already in place from the first iteration's mount.

Hardware Connections

Power cabling is already run to the shelf that the Pi will sit on from last year's project. The power cable for the screen (from its 12V adapter), HDMI cable, and USB cable for touch data will all have to be run through the plastic conduit already in place. The connections cannot exist in a wall-mounted box as they do currently as the connection ports on the Eleduino screen are too far apart to work with the current box. To compensate, dead space has been built into the mount plan for connectors.

Software

Figure 6 shows the GUI process diagram. The GUI process is run in the home.py script and calls all other scripts, organizing their data into the graphics displayed to the user.

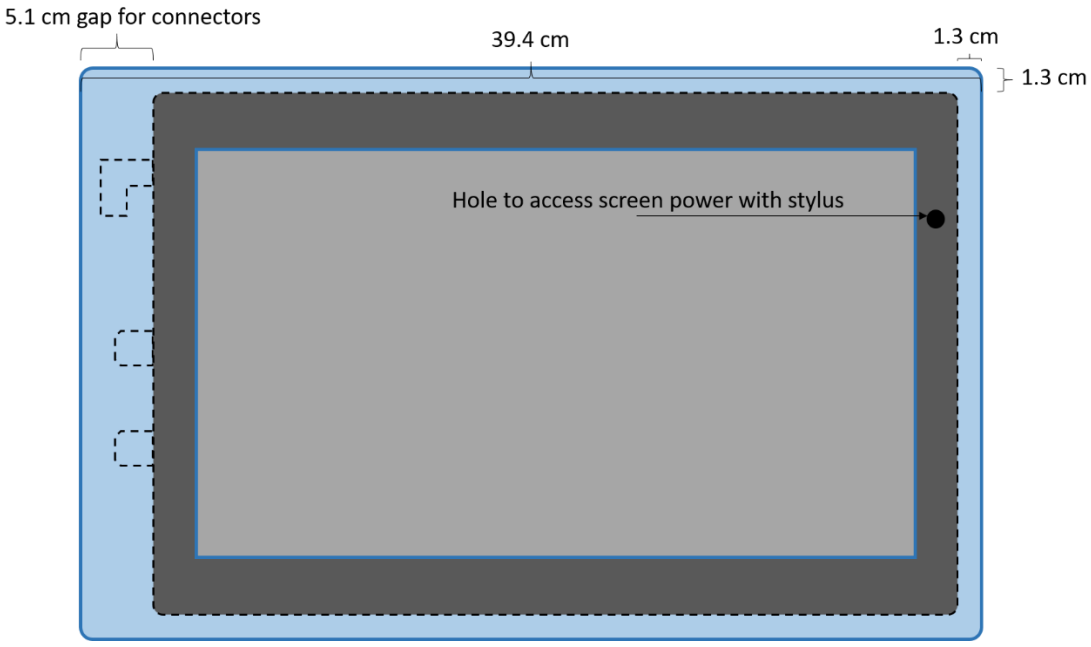

Figure 5: Screen mounting diagram (front)

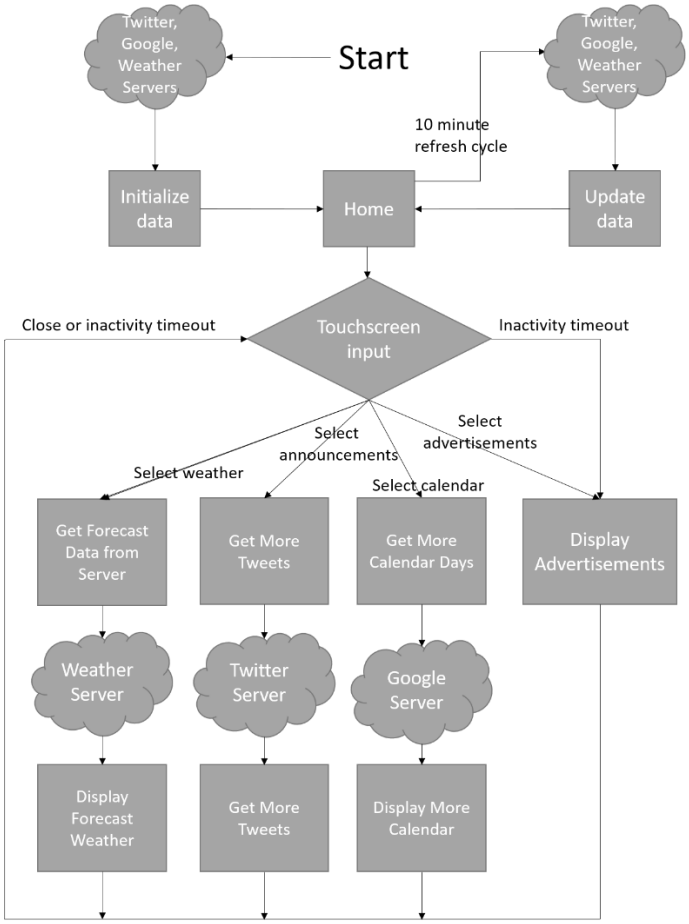

Figure 6: GUI Process Diagram

home.py

The home.py script begins by initializing data for the calendar, announcement, weather, and availability displays on the main home screen. It initializes the data by calling each features corresponding external Python script. It also checks the file directory for advertisements and gets a maximum number of advertisements so that it can set the cycle numbers.

After locally storing the initialization data in volatile memory, the script draws the home screen for the first time. After drawing the home page it sets the polling time for the timer interrupt to one second, defines the interrupt handling function and then starts the GUI application process loop.

The timer interrupt handler is run from the home.py script and counts up to 10 minutes for inactivity as well as for data refresh. If no user input is given in 10 minutes, the screen times out and calls the advertisements.py script to handle ads. Every 10 minutes it also refreshed the home page, or schedules a refresh for when the screen becomes active again. It will only schedule one refresh if the screen is displaying advertisements to minimize the number of API calls made.

googleEvents.py

This script is used to retrieve the calendar events for one full day from the user's Google Calendar. The main(datetime date) function takes a datetime object to determine what day's events to return. It then connects to the API using the API credentials provided in the code. Eventually, moving these credentials to an external file would be a more flexible and secure way of handling API keys. After connecting to the API its requests data for the day give and the decodes the JSON data that the API returns.

After decoding the JSON data it pulls out the elements it needs for each even and builds a Python list of events to return to the home.py file. This list contains condensed information on each event. It includes only start time, end time, description, and color for each Calendar event.

tweety.py

Similar to the googleEvents.py script, the getTweets(int numTweets) method of the tweety.py file connects to the API using a hard-coded set of credentials. It loops numTweets times through the tweets returned and pulls out only the text of the tweet and when each was posted. It takes the string value for when each was posted and converts it to a datetime object before returning.

Each iteration through the loop, the function also checks each Tweet's text for status indicators.

- *Availalble
- *Away
- *Busy
- *Hide

These status indicators are enumerated into integers and returned as status with a Python list of the last numTweets Tweets to the home script, which draws the Tweets.

weather.py

The weather.py file contains the getWeather() function and the isDaylight() function.

The getWeather() function connects to the OpenWeatherMap.org API using credentials stored as hard-coded variable values. These could eventually easily be retrieved from an external file. Once connected,

it gets current weather conditions for a latitude/longitude coordinate that is also stored in variables. The function returns a Python list containing temperature in Fahrenheit and Celsius, as well as an enumerated integer that corresponds to weather conditions.

The isDaylight() retrieves the sunrise time and sunset time from the Sunrise-Sunset.org API and returns wherther or not it is dark in a Boolean variable. The values for sunrise and sunset times are run through a series of logical comparison statements with the current time to determine whether it is after sundown at the current location. This location uses the same latitude and longitude as the getWeather() function.

Parts List

The parts for the IoT Information display are all consumer off-the-shelf (COTS). Hardware subsystems come preassembled and only require interconnecting cables as peripheral parts. The parts are presented in Table 1.

Table 1: Parts list

Qty	Item Description	Source	Price/Unit	Price
1	Eleduino 13.3” 1080P IPS Capacitive Touch Display (sky black)	Amazon.com	$189.00	$189.00
2	CanaKit Raspberry Pi 3 Kit	Amazon.com	$49.99	$99.98
2	SanDisk Ultra 8GB Class 10 UHS-I MicroSDHC	Amazon.com	$9.99	$19.98
1	KabelDirekt (15 feet) Mini HDMI to HDMI Cable	Amazon.com	$9.99	$9.99
1	Micro USB Cable, 3 Pack 10 ft Braided High Speed USB 2.0 A Male to Micro B	Amazon.com	$10.99	$10.99
			Subtotal:	$329.94

Firstly, multiple Raspberry Pis were going to be necessary for the software development process. As soon as a working prototype was established, we had planned that it be displayed using one Pi while the other Pi was reserved for continued code development. Updates were then to be pushed periodically to the Pi that is running the monitor.

Many of the COTS components of the project are sold as kits thus eliminating the need to buy power supply cables, cases, or adapters. The Elduino display comes with necessary power adapter and cable, although the cable may need to be extended pending where the display will ultimately be installed. The Pis will be housed in a case included in the CanaKit, which also contains the power supply for the device.

Two MicroSD cards are necessary, one for each Raspberry Pi. UHS-I cards are capable of 10Mbps data transfer speeds. While the Pi effectively tops out around 22Mbps for reads from the SD card slot, the display application does not require particularly fast data-fetching or writing , and therefore, the Class 1 SD card will be sufficient. While it is undetermined how large the final application file will be, the mounted Raspbian Stretch image

The screen and Pi must be linked together to display and response to touch. The display is connected with the HDMI to Mini HDMI cable, with the Mini HDMI needed on the monitor-end. Touch data is

returned from the display screen which is done via a micro USB to USB A cable, with the monitor needing the micro USB connector.

Schedule and Division of Labor

Original Schedule of Deadlines

Table 2 lays out the original plans for the Fall 2017 semester and the Spring 2018 semester with highlights on deliverable due dates, as well as team goals for meeting functionality requirements.

Table 2: Original Project schedule of deadlines

Date	Item Due/Requirement Met
Fall 2017	
11/16/17	Proposal presentation draft
11/30/17	Project proposal and presentation
12/7/17	Website with proposal presentation and report
	Non-functional, rough layout prototype for display written in Python
	Majority of graphical project aspects created
Spring 2018	
2/15/18	Midpoint progress report
2/16/18	Working calendar and announcements prototype (API work) with home screen and functional weather icon
2/23/18	Added advertisements and display mounted
3/9/18	Student Expo registration deadline
3/23/18	Weather/radar screen (final minimum requirements implemented)
3/29/18	Final report draft
3/30/18	Student Expo abstract
4/5/2018	Project poster
4/10/2018	Student Expo poster setup
4/12/2018	Expo poster judging
4/13/2018	Award ceremony
4/27/2018	IAB poster session

4/28/2018	Project conference
5/1/2018	All deliverables completed and uploaded to website

Table 3: Aactual work completed by date and by group member

Week	Ben's Work	Jacob's Work
2/5	weather.py module completed	Weather icons completed
2/12	Homepage GUI design with wxPython	Weather backgrounds completed
2/19	Homepage GUI design with wxPython	Google api research
2/26	Research/experiment with GUI modules	Basic Google api foundation along with working code and call functions
3/5	Homepage GUI design with appJar	Calling and analyzing more data sent from the Google api code
3/12	Spring break	Spring Break
3/19	Display weather on homepage	Start research on Twitter api; Google api 'endtime' variable functions correctly
3/26	Convert homepage background and graphics to canvas elements	Base Twitter api completed; Google api configured with newly configured 'colorId' variable
4/2	Display calendar on home page	Cleaning Google api, fixing 'morning' and 'night' times to help configure UTC time
4/9	Timer interrupts and interrupt handler	Cleaning of Twitter api code
4/16	Create popup content boxes and display Twitter on home page	Twitter 'status' variable created and amended to the api code
4/23	Created ads using timer interrupts Fill popup content boxes for calendar, weather, and announcements	Complete clean, configuring, setup/installation guide, and uploading to GitHub complete
4/30	TODO: Finalize website and deliverables	

Division of Labor

The labor was split primarily between the graphics programming and the other programming. Much of the time spent working on the project was in the GUI programming and instead of having both team members take on the learning curve of GUI programming at once, it was decided that one would program the GUI and the other would handle much of the API calling and graphic design work.

Nading

Jacob Nading wrote the tweety.py and googleEvents.py files as well as doing all of the graphic design for the project in Photoshop. He also did much of the graphics work on the project poster and created much of the presentation slideshow for both presentations during the course of the project. The original modeling for the GUI was also done in Photoshop by Nading.

In addition to writing the code for the Google and Twitter APIs, he wrote a guide for how owners of the Display get API keys to use for their individual accounts.

Daszkiewicz

Benjamin Daszkiewicz wrote the weather.py file while final decisions were being made on which graphics module to use. The home.py file was written by Daszkiewicz as well as the majority of the project final report and code for the project website.

While not a diverse part of the project, writing the home.py took the majority of the time, as there was a learning curve with GUI programming, and the home.py file also contained the timer interrupt and interrupt handling.

Discussion and Future Direction

There are a few key places another development team should take the IoT Display.

- External configuration file with graphic settings menu
- Message professor from Display
- Appointment scheduler
- Sensors to help confirm professor availability status

Configuration File

An external configuration file and settings menu would allow for a better user experience for the calendar's owner. Presently, API data and location data for weather are hard-coded into the project. In the future, moving this data to an encrypted file outside the project code would make for easier configuration as well as more security for the user's Google Calendar and Twitter accounts.

An in-app settings menu would put the finishing touch on easy use for the Display owner, allowing them to change the weather data location and Twitter/Calendar accounts with ease.

Send Messages from Display

A very useful feature from the student user would be to message the professor directly from the Display. Currently, the Display is a one-directional communication device. Establishing student to professor communication would only bridge the information gap further and could allow for students to request information from the professor without having to schedule an entire meeting or wait for office hours.

Appointment Scheduler

Further increasing the two-way functionality of the Display, an appointment scheduler would be ideal to combine with the calendar. This would fully complete the Display as a scheduling device for both professor and student, maximizing efficiency of office hours.

An API for appointment scheduling already exists with YouCanBookMe. This service is being used on campus in the Smith Career Center, and would be very applicable for future iterations of this project.

Sensors

The original iteration of this project had planned on using sensors to get a better feel for the professor's location and availability, as well as helping manage Display timeout to advertisement. These ideas were put aside for this iteration in the name of simplifying the project and making it easier to expand upon in the future. Now that that has been accomplished, adding interfacing sensors could be a valuable addition to the display.

Conclusion

The IoT Information Display solves the problem of schedulues becoming outdated before their events come to pass. Synchronizing these ever changing schedules with students can cause professors undue frustration. The Display lends a place where the professor's schedule will always be up to date and will synchronize in real time. If off campus, the Google Calendar will still be viewable by traditional means. It is the best of both worlds. Twitter announcement functionality will allow for real-time announcements to be displayed in such a way that they can also be viewed from a student's personal device while they may not be around the display.

Minimal hardware is required, in that only a Raspberry Pi is needed to run the touch display. Some minimal connection is needed between the two, but most of the project will be software based. The system will run as a Python application to make data handling as efficient as possible and avoid the need for hosting a web server or using Bash scripting to pass variable between various php and HTML destinations.

Minimum functionality requirements of a calendar, announcements, weather, and advertisements have been completed dispite setbacks with our original design and graphics moudle. In addition, a groundwrok has been laid for future groups to add more features and functionality. This was another of the main goals for this iteration of the project. Moving forward, the IoT Informaion Display should be an easilty modifiable and rich project for professor use, and student development.

References

[1] M. Archambault, "DAKboard is a Customizable Wall Display for Photos, Calendar Events, and Weather," PetaPixel, 19 August 2015. [Online]. Available: https://petapixel.com.

[2] kmccb, "Raspberry Pi Framed Informational Display - Google Calendar, Weather, and More.," 7 April 2016. [Online]. Available: https://imgur.com/gallery/z94Vr.

[3] B. Eagan, "Smart Mirror (with Optional Alexa)," hackster.io, 18 Aprit 2017. [Online]. Available: https://www.hackster.io.

Additional Readings and References

Google Calendar API

https://developers.google.com/google-apps/calendar/

Twitter API

https://developer.twitter.com/en/docs

https://pypi.org/project/twitter/

Weather request

https://www.weather.gov/

appJar

http://appjar.info/

Project Website

http://ee.bradley.edu/projects/proj2018/iot_display/