Smart Home System

Software Engineering Project Second Report

Group 12

https://github.com/RUSEGroup12

Akhilesh Bondlela, Brian Ellsworth, Bhargav Tarpara, Nicholas Grieco, Vinay Shah, Huy Phan

Contribution Breakdown

Everyone did equal amount of work

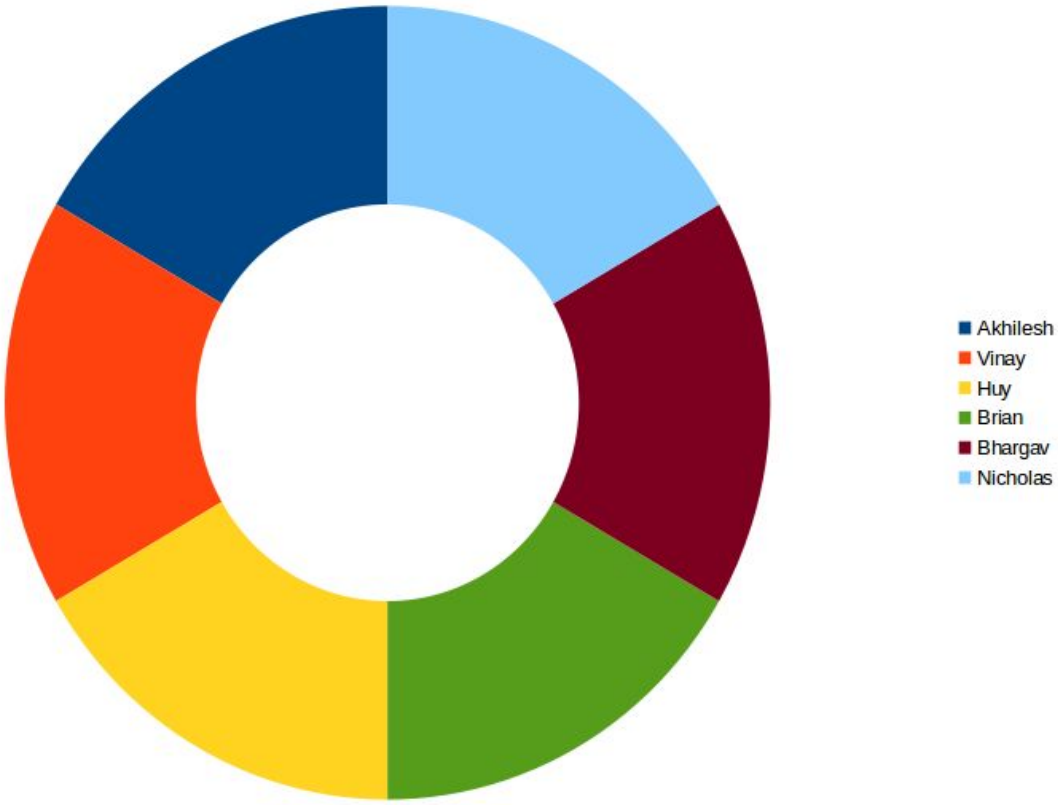


Table of Contents

1. Interaction Diagrams **4**

2. Class Diagram and Interface Specification **10**

A. Class Diagram 10

SES Class Diagram 10

SAS Class Diagram 11

SSS Class Diagram 12

B. Data Types and Operation Signatures 13

C. Traceability Matrices 17

SAS 17

SSS 19

SES 20

3. System Architecture and System Design **20**

A. Architecture Style 20

B. Identifying Subsystems 21

C. Mapping Subsystems to Hardware 21

D.Persistent Data Storage 22

E. Network Protocol 22

F. Global Control Flow 22

G. Hardware Requirements 23

Project Management for Part 2 23

4. Algorithms and Data Structure **24**

Algorithms 24

SAS 24

SSS 24

SES 24

Data Structures 25

SAS 25

SSS 25

SES 25

5. User Interface Design and Implementation **26**

6. Design of Tests **27**

Casual Test Cases 27

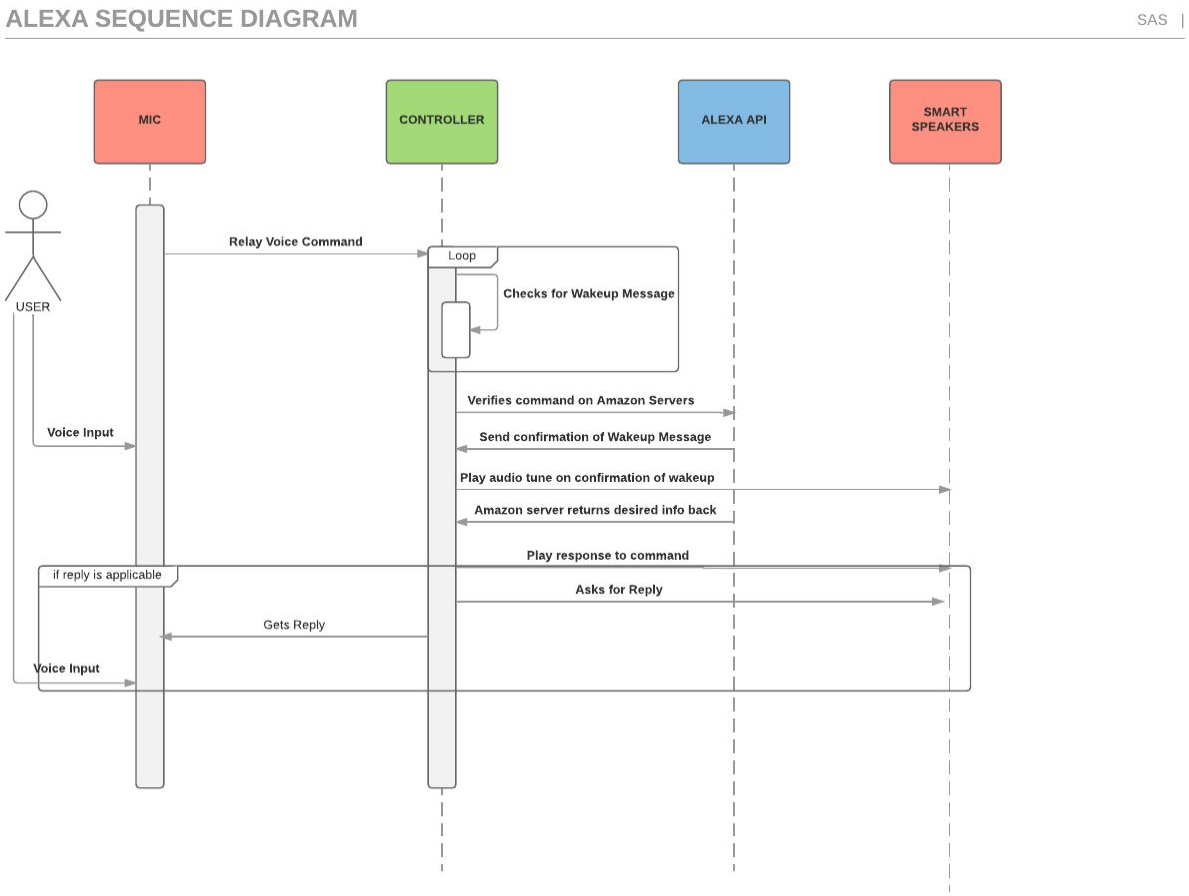
Descriptive Test Cases 29
Integration Testing 34

7. Project Management and Plan of Work 35
A. Merging the Contributions from Individual Team Members 35
B. Project Coordination and Progress Report 35
C. Plan of Work 35
D. Breakdown of Responsibilities 37

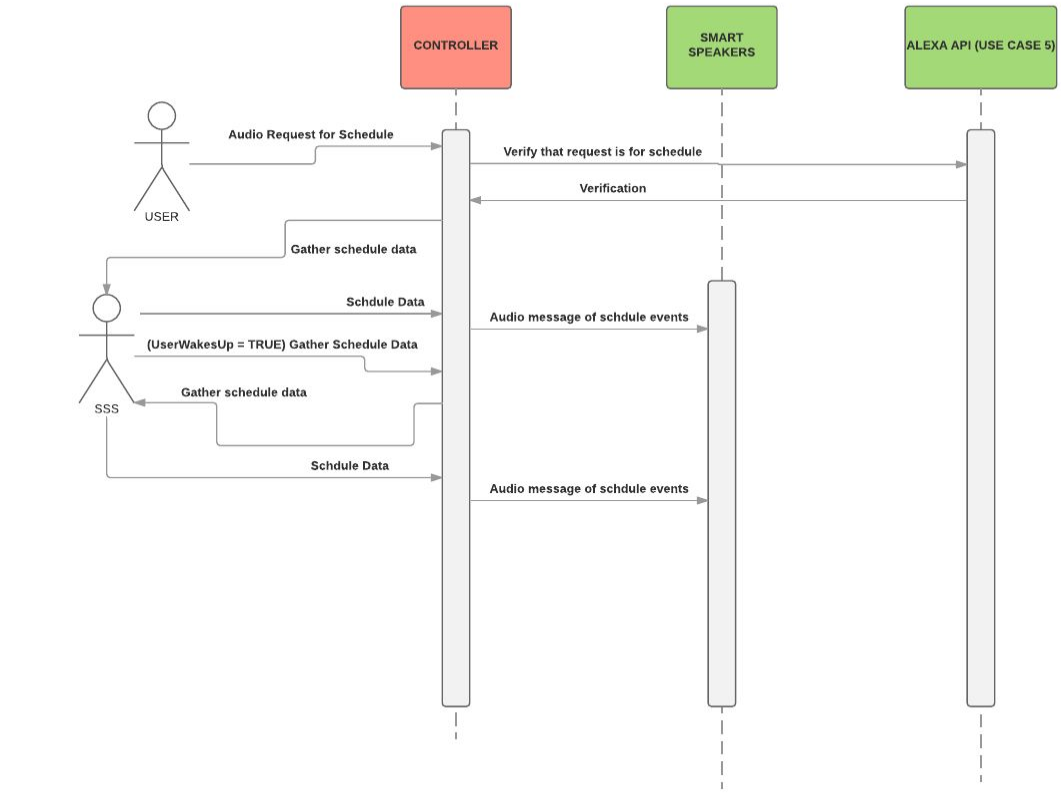
8. References 37

1. Interaction Diagrams

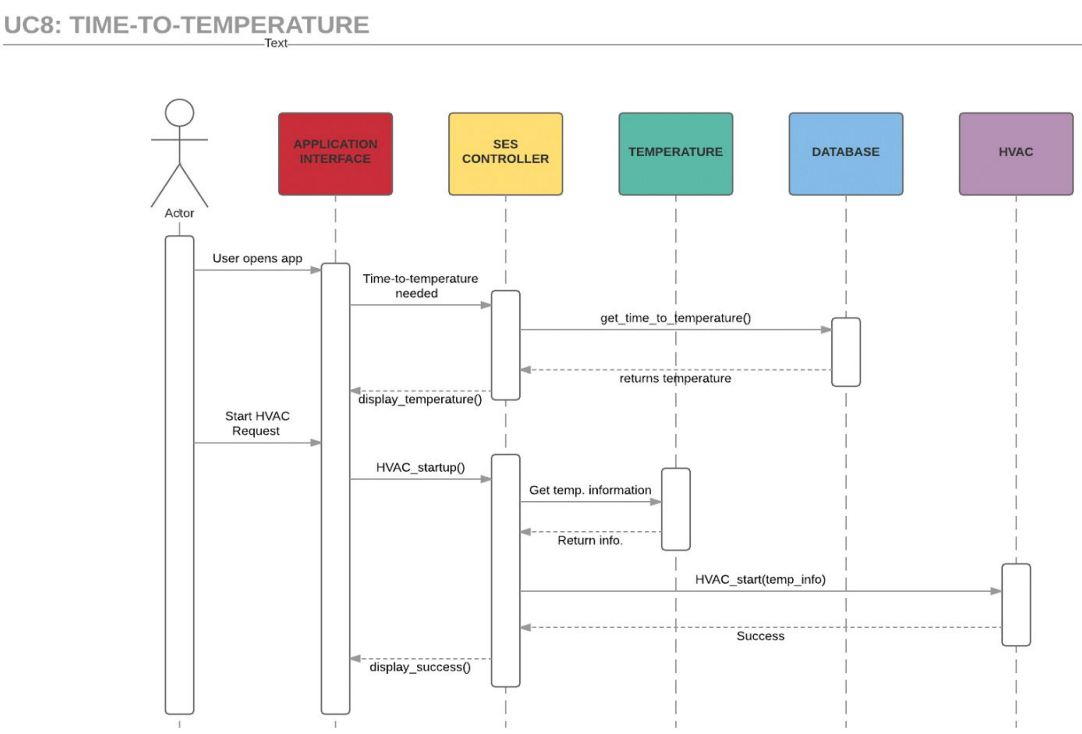
Use Case 5 - Amazon Alexa[6] *Interaction*



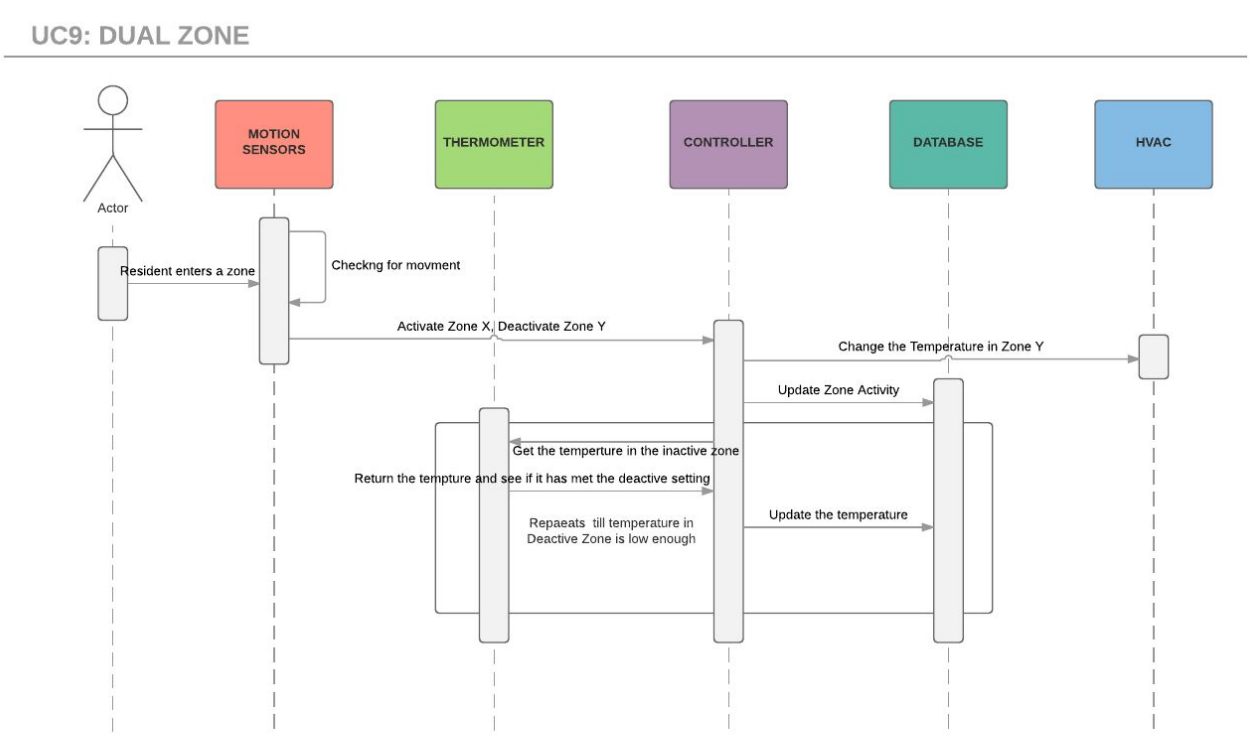
Use Case 6 - Speakers for SSS



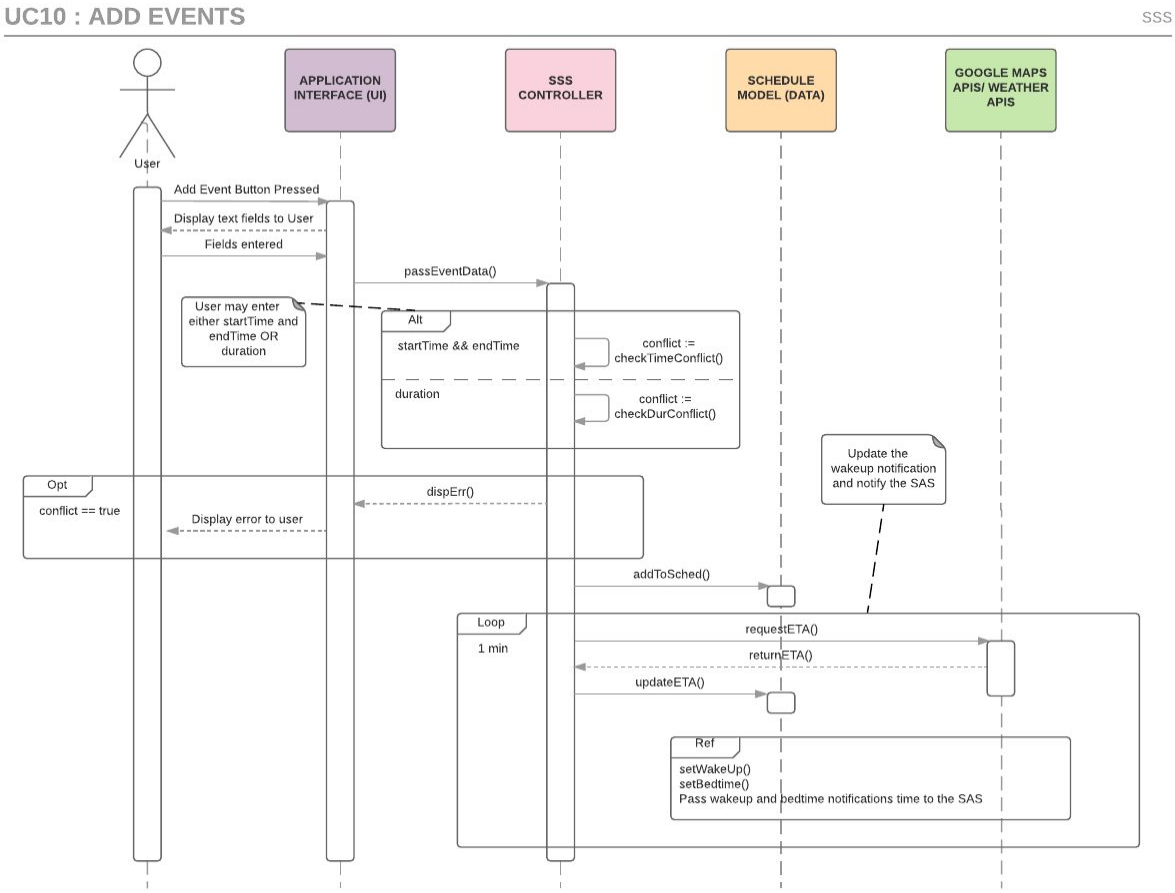
Use Case 8 - Temperature Ready



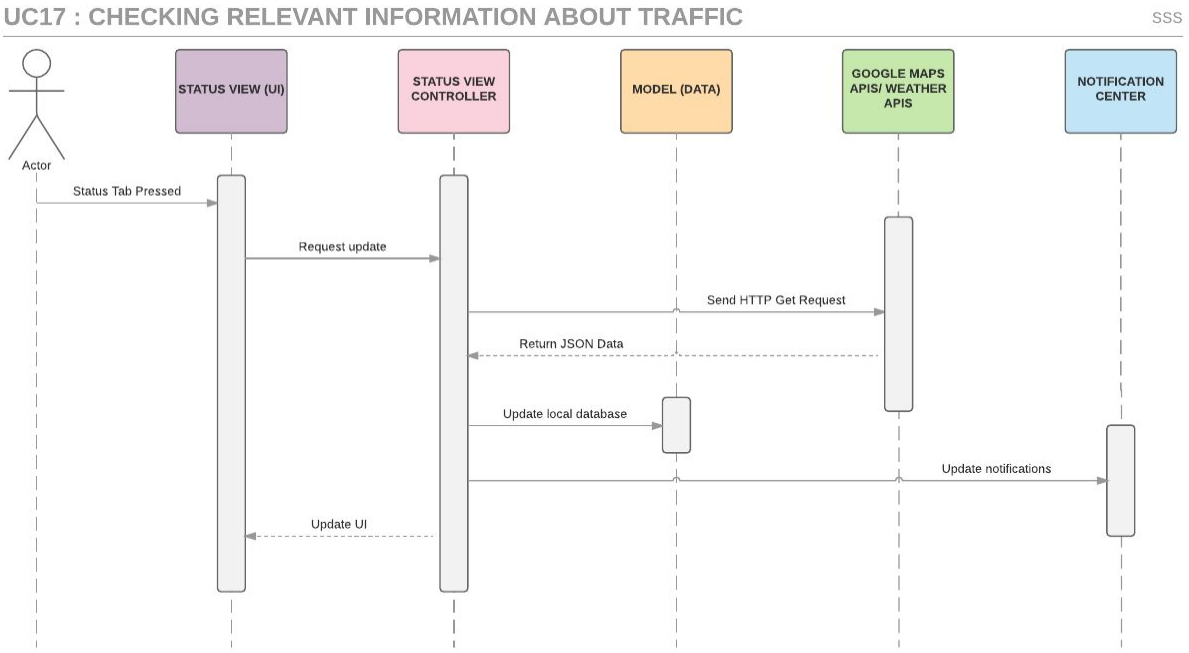
Use Case 9 — Dual Zone Activation/Deactivation



Use Case 10 — Schedule Event Addition



Use Case 17 - Checking relevant information about traffic



Our design principles are a hybrid of low coupling principles, expert doing, and high cohesion. Each object is an expert at a small task and communicates its information to other objects. For example, in use case 9 there are many actors, like the physical sensors, thermometer, and HVAC [1]. Each piece of hardware has his its own object that controls its physical responses. In order to preserve the cohesiveness of each object, the controller class was given a significant amount of responsibility.

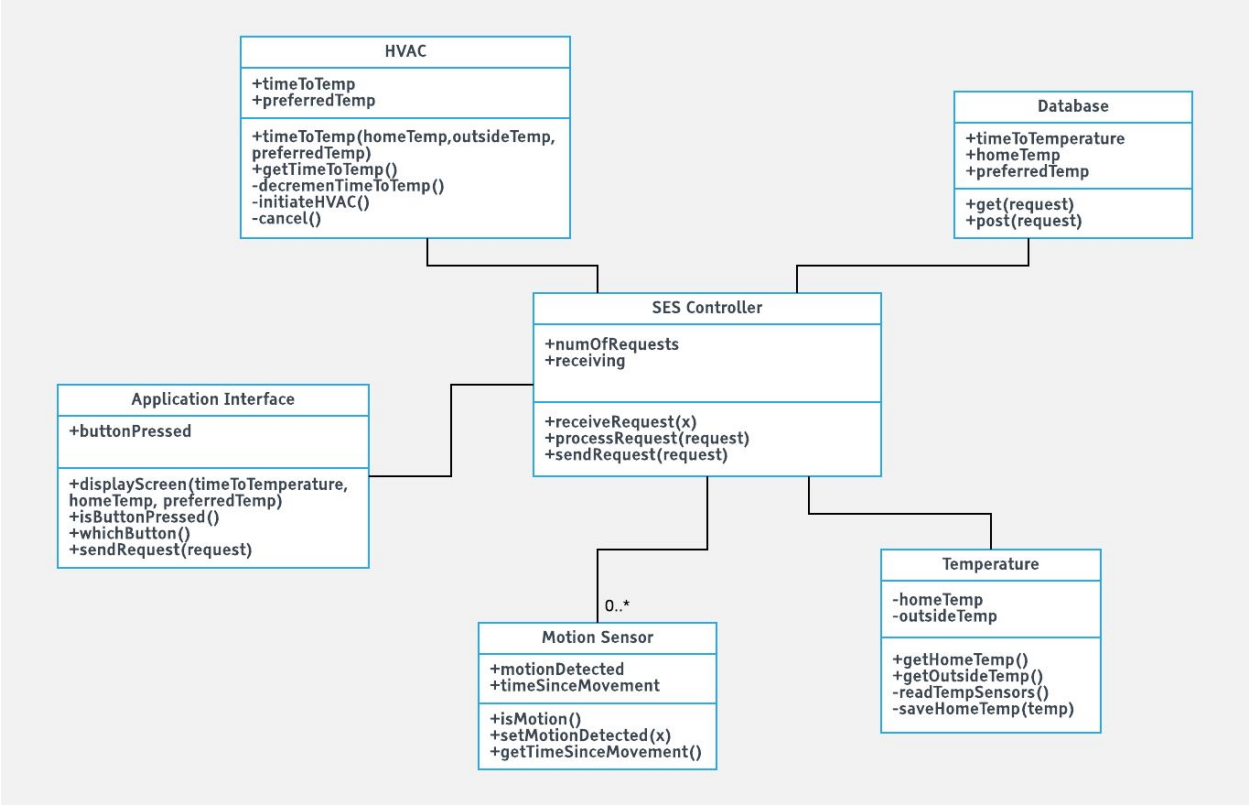
Each of our subsystem is its own stand alone system. Technically, each subsystem can function minimally by itself. Each subsystem sends data to other subsystems to complete certain tasks; the SSS would send audio data to the SAS to play the audio through the speakers The controllers of each our subsystem employ tight coupling however each of the objects that communicate with the controller employ loose coupling. The job that controllers play is generally to identify the type of information it receives from the actor and relay it to different objects. In theory controller is high cohesion but since it handles many different type of information it's more logical to label the controllers as low cohesion.

Overall we are type 1 and type 3: responsibility of knowing and responsibility of communicating respectively.

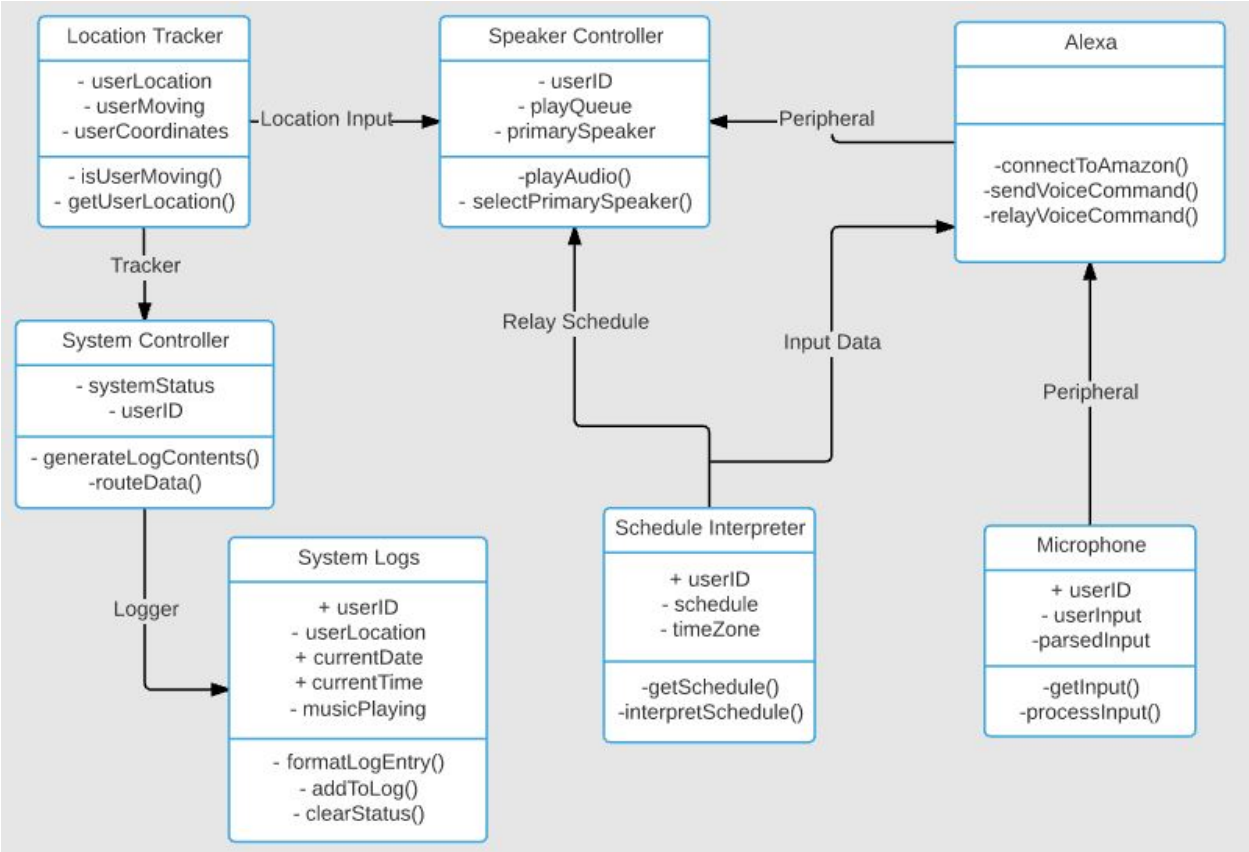
2. Class Diagram and Interface Specification

A. Class Diagram

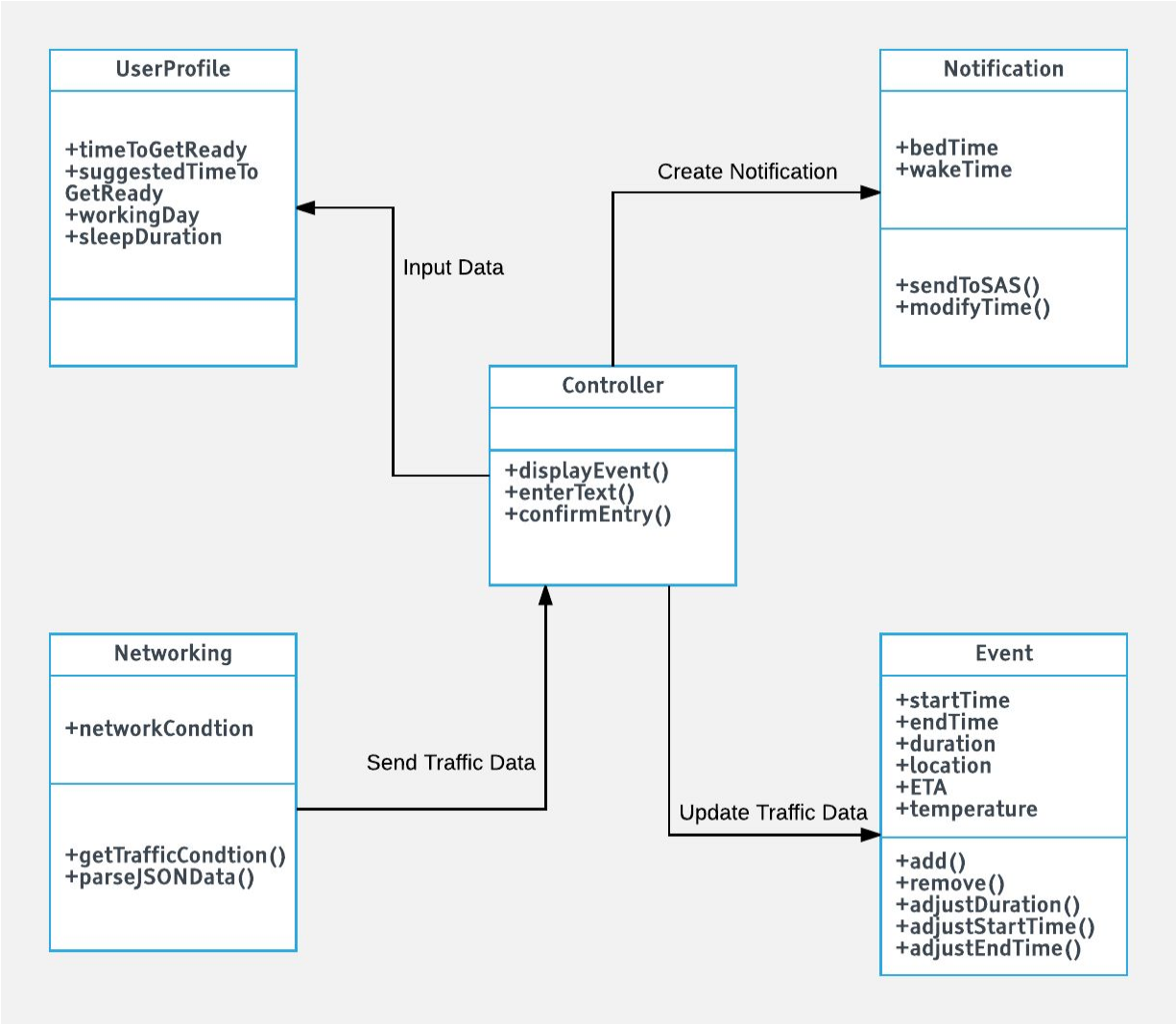
SES Class Diagram



SAS Class Diagram



SSS Class Diagram



B. Data Types and Operation Signatures

Class: SES Controller

The SES Controller controls all requests and controls the exchange of information between different classes.

Attributes:

+numOfRequests: int --- Current number of requests being received by the controller

Operations:

+processRequest(request): boolean --- Receives a request struct and extracts relevant information. Returns boolean 1 if processed correctly.

+receiveRequest(x: bool): boolean --- Sets the controller's 'receiving' data item to one or zero which controls flow of requests. Returns 1 for success.

+sendRequest(request: Request, targetClass): bool --- Reroutes request to target class for handling.

Class: HVAC

This class directly controls the heating/cooling system and calculates important data used for many use cases.

Attributes:

+timeToTemp: int --- The time needed to go from the current home temperature to the preferred temperature.

+preferredTemp: float --- User's preferred home temperature

Operations:

+timeToTemp(homeTemp: float, outsideTemp: float, preferredTemp: float): float --- Calculates the time needed to transition between current temperature and preferred temperature.

+getTimeToTemp(): float --- Returns timeToTemp value

-decrementTimeToTemp(): boolean --- Decrements timeToTemp as time progresses after the user initiates heating/cooling

-initiateHVAC(): boolean --- Initiates the physical heating/cooling system

-cancel(): boolean --- Stops heating/cooling process from starting

Class: Database Controller

The database stores all important data for later use

Attributes:

+timeToTemperature: float --- Described in the previous class and stored for data persistence

+homeTemp: float --- Temperature inside the user's home

+preferredTemp: float --- User's desired home temperature

Operations:

+get(request): boolean --- Used to request information from the database

+set(request): boolean --- Used to update or create new data in the database

Class: Application Interface

Displays and provides an interface with the application/website

Attributes:

+buttonPressed: boolean --- Boolean value that keeps track of when a button has been pushed

Operations:

+displayScreen(timeToTemperature: float, homeTemp: float, prefferedTemp: float): void --- Sends and displays important data to the GUI

+isButtonPressed(): boolean --- Returns 1 if user has pressed a button, else returns a 0

+whichButton(): string --- This function returns the string name of the button that was pressed

+sendRequest(request): boolean --- Sends a request struct to the controller for processing

Class: Temperature

Interfaces with hardware and API's to obtain relevant temperature information

Attributes:

-homeTemp: float --- Temperature in user's home

-outsideTemp: float --- Temperature outside user's home

Operations:

+getHomeTemp(): float --- Returns homeTemp

+getOutsideTemp(): float --- Returns outsideTemp

-readTempSensors(): float --- Interfaces with hardware to retrieve temp. data

-saveHomeTemp(temp: int): boolean --- Changes homeTemp variable to the argument passed in the function

Class: Motion Sensor

Interfaces with motion sensors and keeps track of time dependent data

Attributes:

+motionDetected: boolean --- Variable that is 1 when motion is detected and 0 when no motion detected

+timeSinceMovement: float --- Stores how much time has passed since the last movement was detected

Operations:

+isMotion(): boolean --- Interfaces with hardware and returns 1 when PIR sensor is set off

+setMotionDetected(x: boolean): boolean --- Sets motionDetected to 0/1 based on the function parameter

+getTimeSinceMovement(): float --- Returns timeSinceMovement

Class: Location Tracker

Interface with SAS controller and provide user location

Attributes:

-userMoving : 1 or 0, moving and stationary respectively

-userLocation : holds float or double(TBD) based based on user location

-userCoordinates : (int array), coordinate location of the user in the house

Operations:

-isUserMoving(): boolean - returns boolean value whether the user is stationary or in motion

-getUserLocation():int/float - return back the approximate location of the user in the home

Class: Speaker Controller

Attributes:

+userID : (int/long/hash key), ID of user who is currently using the system

-playQueue : (struct), list of the names of the next files to be played on the speaker system

-primarySpeaker : (string), closest speaker to the user

Operations:

-playAudio(): int - play audio that has been cached on the Speaker Controller. Similar to a print queue

-selectPrimarySpeaker(): finds the key of the speaker closest to the user

C. Traceability Matrices

SAS

Domain	Classes						
	Location Tracker	**Alexa**	**Microphone Controller**	**Schedule Interpreter**	**System Logs**	**System Controller**	**Speaker Controller**
User Function Input			The microphone is used to gather user input		The logs will maintain and/or display important user info		
SAS Controller	The controller will read the data from the location tracker	Alexa replies to voice command are relayed to controller			The logs will maintain and/or display important controller info	The SAS controller is the system controller	
SAS Operating Interface						The system controller provide information to the UI	

Smart Speakers					The logs will maintain and/or display important speaker info		The speaker controller controls the speakers
Alexa API		The Alexa API allows the system to integrate with Alexa					
Find API	The FIND API includes all the key function of the tracker.				The logs will maintain and/or display important user info		

SSS

Domain	Classes				
	User Profile	**Controller**	**Notifications**	**Event**	**Networking**
Application Interface		Controller get data and display UI			
Controller					Parse JSON Data return from networking class
Schedule				List of events	
Wake Up Notification				Send user notification from event lists	
Bedtime Notification				Send user notification from event lists	
Google Maps API					Networking class get data from GG maps and add them to events

SES

Domain	Clases					
	HVAC	**Application Interface**	**Motion Sensor**	**SES Controller**	**Temperature**	**Database**
Request		Get the views from the application controller				Post requests to the database
Application Interface				Is main iterator	Display data from temperature	Parse JSON data returning from db
Sensors		Displays where interaction is				Posts to the databases at time intervals
SES Controller	Interacts with HVAC to send boolean values	Forwards views	Forwards data to database connection		Gets data points from temperature objects	Is the primary communicator of the database controller.
Lighting		Displays where heat points are				

3. System Architecture and System Design

A. Architecture Style

We have a Control Center which all subsystems will communicate with. Each subsystem with upload its data, get relevant data of other subsystems through Control Center. Also we adhere to the MVC design pattern.

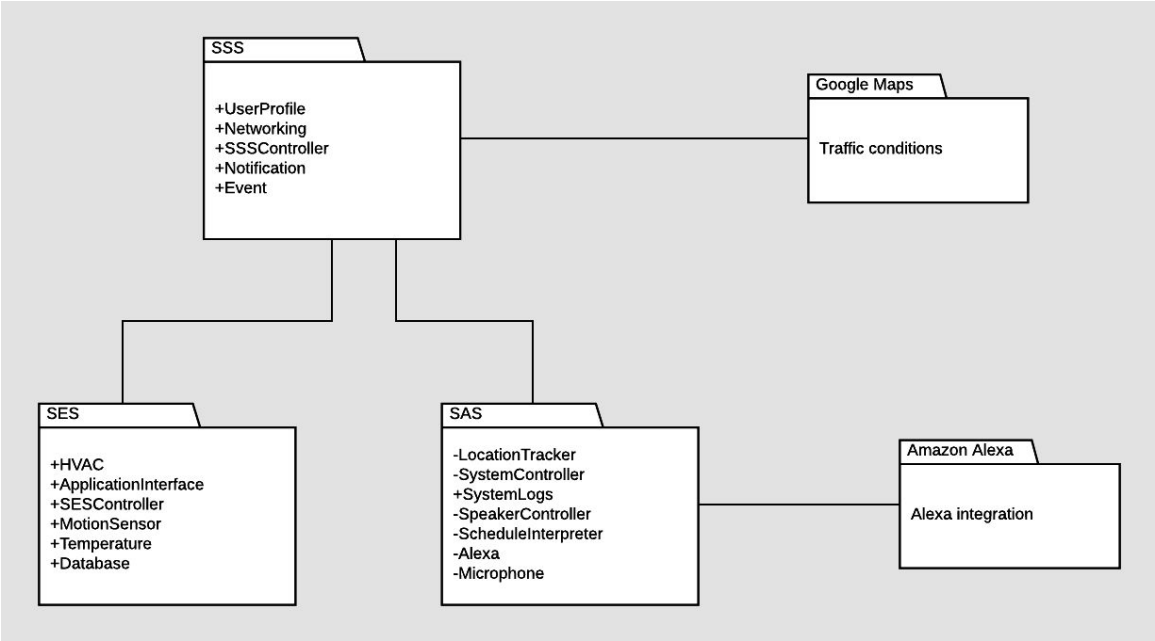
The model represents the data, and does nothing else. The model does NOT depend on the controller or the view.

The view displays the model data, and sends user actions (e.g. button clicks) to the controller. The view can be:

- independent of both the model and the controller
- the controller, and therefore depend on the model

The controller provides model data to the view, and interprets user actions such as button clicks. The controller depends on the view and the model. In some cases, the controller and the view are the same object.

B. Identifying Subsystems



C. Mapping Subsystems to Hardware

In general the project adheres to a client server model. The client could be any device: a laptop, mobile phone, really anything with a browser. The server program needs to be hosted either on one of our laptops or we can get hosting from AWS or Heroku. All three subsystems at least for the dashboard part of the application are mapped to this client server model.The SAS will require a speaker module. This system will also require a minimum of 1Mbps steady WAN throughput at all time to function properly with a FastEthernet(100Mbps) LAN connection with the other subsystems and

the database. For our purposes we used a raspberry pi to run our subsystem and connect to Amazon services for Alexa.

D.Persistent Data Storage

The database schema of choice is a Relational Database. This is because we will have an organized element look up table. Some data for the SES subsystem will be Current Temp: float, Preferred Temp: float, Time to Temp: float, Motion Detected: boolean, timeSinceMovement float, Outside Temp: float. The SAS user tracking software will have reference points stored in the database. These reference points are used to pinpoint user location. SSS will save a list of events.

E. Network Protocol

In general, there will be a central server with an api route where the data in the url will be parsed and stored or requested. Communication between the SAS and User Device for Audio Tracking purposes will occur using sockets in C. The reason we chose this was to more finely control the hardware and scalability for the future. The max number of user audio tracking devices will directly correlate to the machine that the SAS is running on. For example as the number of user increase dramatically the machine can(TBD) fork new processes to handle new users and track them. The number of forked processes will depend on the power of the host machine. This provides the user with an efficient way of scaling the the SAS system. For the SSS we will use HTTPs request to get data from Google Maps API . We will also use OpenWeatherMap's API to retrieve temperature information used in the SES system. In between our objects http requests will be used to transfer information and route requests.

F. Global Control Flow

SES:

A majority of our code is event driven. As seen in our use cases, most of our scenarios depend on the user interacting with the system or some sort of inactivity. For example, when no motion has been detected for a fixed amount of time in an area, the heating/cooling minimizes energy consumption. The system depends on time for certain functions like this that are used to judge when it's acceptable to alter the temperature in a home. One of our main features, time-to-temperature, is an estimation of time which allows the user to choose when to execute the event of starting the heating/cooling system.

SAS:

Our system is event driven and mostly triggered by the user and/or other subsystems. Whenever the user activates the SAS by the Audio Functionality or other subsystems want transmit audio SAS is called upon to complete the function. Our system is not directly reliant on time but the SSS is and when a time event occurs on the SSS, the SAS is called upon to do a certain function from ringing the alarm to playing the user's schedule. For the tracking between user device and speaker location we will be using multiple forked processes for concurrency.

SSS:

Our system is driven based on user inputs. The alarm will sound base on the weather condtions, traffic conditions, the time for user to get ready, the time user have to be at work. User input his schedule, then the system fetch all related information, and finally it sets the alarms.

G. Hardware Requirements

For the SES system to work we need several types of sensors. For some of use cases we need to detect whether or not there are people in the area and we will use PIR sensors. The main features of our system require sensors that measure inside home temperatures and humidity. To get the temperature of the house a thermometers data will be needed. As well two machines that have web browsers.For the SAS system, we will need speakers which will be connected to Alexa and the Speaker controller.

Project Management for Part 2

This week we had refocused ourselves for a more code first outlook as well as reminding ourselves of the goal. After reading an expert from the group management paper: "The State of Cooperative learning in Postsecondary Settings" by David W. Johnson [4], we did an activity at our round table meeting where we asked two questions openly to the group. This was a useful exercise in which we were able to get on the same page. We also went back to the drawing board in terms of which technologies to use and how to code up the project. We meet together on friday night to work these ideas out as well as work on the report.

7. P4. Algorithms and Data Structure

Algorithms

SAS

The SAS tracks the user's location in the home and automatically plays audio on speakers closest to him while also avoiding conflicts with other user's audio playing. To implement this feature the SAS will have to constantly gather user geo location(with respect to the mapped location). The user's phone or tablet will transmit this statistics to a computer(raspberry pi for our case). We do this by having listening sockets on the raspberry constantly listening to new user locations and each location will coordinate with a certain speaker. Our Location API will provide the user location and transmit that over the local area network to the raspberry pi. This will occur simultaneously.
The inefficiencies are apparent if the user wants to scale the SAS to include more users. To combat inefficiencies and allow for future expandability we plan to have sockets opened in concurrent forked processes for future expandability use only. However for a limited number of users, the user does not have to initiate a concurrent version of the SAS.

SSS

The SSS uses array index lookup in order to insert events into the schedule. If an event already exists at the specified array index, then the event creation fails. The SSS will fetch predicted traffic condition at a particular time from Google Maps API. The time of the wakeup notification will be calculate by adding the traffic travel duration calculated by Google Maps to the user's timeToGetReady attribute in the UserProfile object. In addition, the SSS uses a networking algorithm to get JSON and parse JSON data from Google.

SES

For this first demo we plan on using an algorithm based on Matlab's Thermal House Model [1]. Home temperature data will be read in from temperature sensors and outside temperature will be retrieved from a weather API. This data will be used to calculate the approximate time it will take to reach the users preferred temperature if the HVAC was started at any given moment. The flow of heat will be calculated based on these two

temperature and some general parameters similar to most HVAC systems. On top of that we will be incorporating Newton's Law of Cooling [2] which states that the heat transfer is proportional to the difference in temperatures.

Data Structures

In general, all systems will use a database. The question then becomes what datastcuters is the database using? In particular MySQL uses a combination of binary trees and hashtables. Because we are using a database the data structures are is abstracted away from us so we do not have to worry about updating.

SAS

The SAS will have to keep track of geo location points throughout the home. To store such data points we will use a hash table with chaining. So each geo location will getted mapped to a certain index in the hash table and each location in the table will correlate to a speakers in the vicinity of the geo location. Given more than one speakers in a single geolocation, speakers will be chained to the position of the table and audio will be played throughout all.

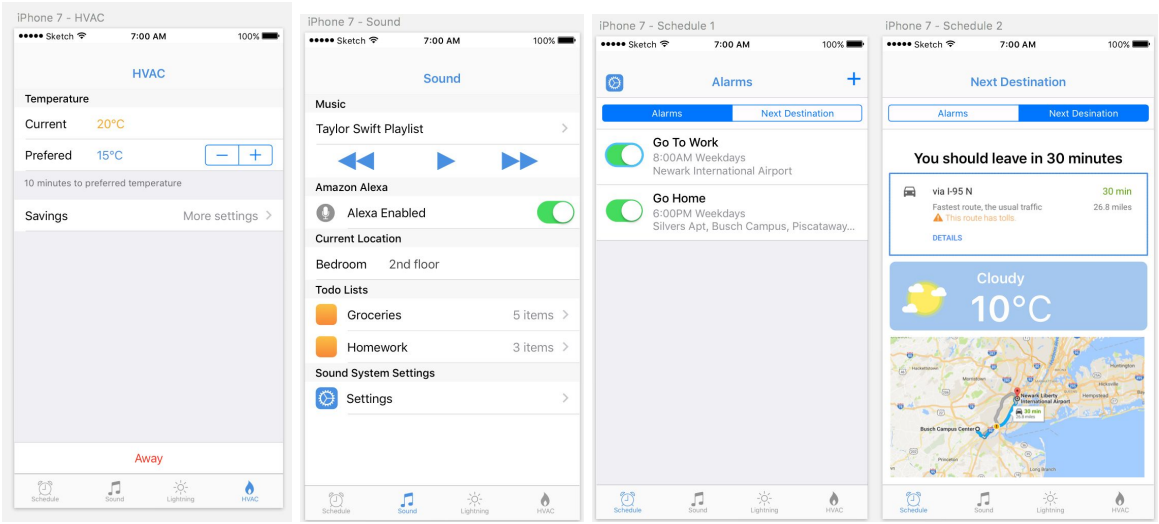
SSS

The SSS will store events as objects. These objects are saved in a public Schedule object, which contains an array of events.

SES

Temperatures will be stored in vectors. This is because after every cycle the most recent temperatures will be added to the ends of the list dynamically. If we want to access temperatures a few cycles earlier the time complexity is O(1) whereas stacks, queues and linked lists are O(n). We won't be performing many other operations on the data besides dynamically inserting, deleting and and accessing data. Therefore, vectors are the most efficient.

5. User Interface Design and Implementation



We have not made any significant changes to our graphical user interface. However, we are working to streamline our interface to make operations as easy and concise as possible. We are making progress in eliminating technical jargon in the user interface for ease of use for the user. We want the user to take advantage of all our features and not get lost in technical jargon. We are going to implement settings option for each of our subsystem that the user can easily navigate through and understand what each option does with little or no research. Android may look a little different.

6. Design of Tests

Casual Test Cases

Use Case 1: Todo List
Simulate a wake up alarm and check to see whether the SAS notices the wake up and plays the todo list

Use Case 2: Audio Room Selection
Attempt to change audio playback from one room to another manually while in the middle of a playback such as a song.

Use Case 3: Audio Selection Conflict Control
Attempt to play audio on a speaker where audio is already being played by another user.

Use Case 4: User Audio Tracking
Move location of the user's phone from the vicinity of one speaker to another and see if the audio playback transitions to new speakers while stops playing on the original

Use Case 5: Amazon Alexa Integration
Attempt to access Amazon Echo services with voice command and attempt to get a response through the speakers

Use Case 6: Speakers for SSS
Trigger other subsystems to attempt playback audio messages through speakers.

Use Case 7: Monitoring for SES
Check if log file is there.

Use Case 8: Temperature Ready
Check if time to temp is a positive number and that they match at some point.

Use Case 9: Dual Zone Inactivity
Run a test where sensor input is nothing in one zone and active in another.

Use Case 10: Activate Dual Zone

Check if the system responds to no activity

Use Case 11: Sensor Activation
Check if the system responds to activity

Use Case 12: Wake Up Adjustments
Check temperature at a given time point to see if it matches setting.

Use Case 13: Going to Bed Adjustments
Check temperature at a given time point to see if it matches setting.

Use Case 14: Empty Home
Check if button for empty home is a clickable object.

Use Case 15: Energy Saving
Check if graph is generated and function generating the graph is working.

Use Case 16: Schedule Initialization
When the user first uses the app, a schedule is imported from the user's local calendar application. The user should be able to add events initially to the schedule

Use Case 17: View Scheduling Factors
When the SSS provides a bedtime notification time, the user should be able to view the factors that led the SSS to make its decision. This includes viewing traffic data from Google Maps including anticipated travel duration.

Use Case 18: Schedule Modification
The user should be able to modify existing schedule events. This includes modifying parameters of each event: start time, end time, duration, location, estimated time of arrival, and temperature.

Use Case 19: Wake-up Notification
The user must be awoken at the time specified by the notification. The notification is set by assessing the Google Maps traffic data.

Use Case 20: Event Addition to Schedule
Events that the user enters into the application must be successfully saved into the schedule. Schedule conflicts must be handled and reported to the user.

Use Case 21: Bedtime Notification
The user must be provided with a notification at night so that he can go to bed at a time that allows him to wake up after a user-specified duration.

Descriptive Test Cases

Use Case 1: Todo List
An alarm wakeup is simulated with a filled todo list

Success
-SAS should be triggered by a wakeup
-SAS should playback todo list through the speakers

Failure
-SAS remains dormant during a wakeup event
-SAS is triggered by wakeup event but plays nothing from speakers

Use Case 2: Audio Room Selection
Audio playing on one speaker is changed to playback on another speaker

Success
-Playback stops on the first speaker and starts playing on the chosen feature

Failure
-Nothing happens
-Audio plays on both speakers
-Audio switches to wrong speakers

Use Case 3: Audio Selection Conflict Control
Simulate a conflict of audio playback

Success
-If conflict occurred due to movement playback on speakers is first come first served basis
-If user manually changes playback to speaker, user with highest priority is given playback access

Failure

-User priority is used to judge playback rights due to movement of user

-Manually changed playback conflict isn't resolved based on user priority

Use Case 4: User Audio Tracking

User walks from one speaker location to another during playback

Success

-Audio follows user as he/she walks from one speaker location to another while cutting playback

Failure

-Audio playback remains stationary as user moves.

-Delay of audio transition is arbitrarily long

Use Case 5: Amazon Alexa Integration

Activate Alexa

Success

-Alexa remains always listening and replies to voice commands

Failure

-Alexa does not capture most voice command

-Alexa transmits commands but doesn't reply back

Use Case 6: Speakers for SSS

Other subsystems to attempt playback audio messages through speakers

Success

-Other subsystems can triggers SAS and relay audio through speakers

Failure

-Other subsystems cannot communicate with SAS

-Other subsystems can associate to SAS but cannot play audio

Use Case 8: Temperature Ready

User presses button to initiate heating/cooling

Success

- HVAC system starts
- Time-to-temperature countdown is initiated

Failure

- HVAC fails to start
- Time-to-temperature stays static

Use Case - 9: Dual Zone Inactivity

No movement for a set period of time

Success

- HVAC decreases heating/cooling power
- Lights shut off

Failure

- HVAC stays at same power or is increased
- Lights stay on

Use Case - 10: Activate Dual Zone

Users notifies system to adjust lighting and temperature in a zone they are moving to preferred settings

Success

- Room requested will start time-to-temperature process
- Lights will turn on in respective area

Failure

- HVAC does not start
- Lights are not turned on

Use Case - 11: Sensor Activation

User’s enter a room and lights are turned on if it is dark (low luminosity)

Success

- Light turn on

Failure

- Lights don’t turn on when motion is detected
- Lights turn on when no motion is detected

Use Case - 12: Wake Up Adjustments
Time specified before user wakes up the SES initiates

Success
- Temperature is users preferred temperature when they wake up
- Lights gradually turn on when motion is detected

Failure
- Preferred temperature is not reached upon waking up
- Lights do not turn on

Use Case - 13: Going to Bed Adjustments
Temperature adjusts to energy efficient temperature around normal sleep hours. Lights turn off when user notifies the system through the application

Success
- Temperature is users preferred temperature when they go to sleep
- Lights turn off when the user indicates they are going to sleep and there is no/minimal movement

Failure
- Temperature is not what the user would prefer during sleep
- Lights do not turn off

Use Case - 14: Empty Home
User notifies the system they are leaving their home

Success
- HVAC decreases power to optimal temperature
- Lights shut off

Failure
- HVAC stays on or at inefficient levels after user left
- Lights stay on

Use Case - 15: Energy savings
User checks their energy savings for a given time duration

Success
- Energy savings are calculated

- Energy savings can be displayed on the app

Failure

- Energy savings not calculated properly
- Energy savings won't show up on the app

Use Case 20: Event Addition to Schedule

Success:

-The user presses the "OK" button and the new event object is saved in the schedule object.

-The user presses the "cancel" button and the new event is not saved.

-Event data is saved properly.

-New events with notification or temperature settings should send a signal to the SAS or SES, respectively.

Failure:

- The user presses the "OK" button, but the new event object is not saved in the schedule object.
- The user presses the "cancel" button, but the new event is saved into the schedule.
- Event data is not saved properly. It appears as null or as invalid data values.
- New events do not properly signal the SAS to play a wakeup notification or bedtime notification.
- New events do not properly signal the SES to change the temperature for an event with a temperature setting.

Use Case 19: Wakeup Notification

Success:

- The SSS is unable to appropriately set the wakeup time based on the user data and the data from Google Maps.
- The user's location is properly conveyed from an Event object to Google Maps.
- Google Maps returns an appropriate travel duration value.
- The SAS plays a notification sound at the time of the wakeup notification.

Failure:

- The SSS is able to appropriately set the wakeup time based on the user data and the data from Google Maps.

- The SAS does not play a notification sound at the time of the wakeup notification.

- The SSS is not able to communicate with Google Maps in order to receive a time estimate for the approximate travel duration.

-The SAS plays a notification sound at the wrong time.

Integration Testing

The test will simulate the event of the user waking up. The SSS will monitor the traffic and wake up the user at the optimal time. The SSS triggers wakeup events on other subsystems. The SES will then turn on the lights in the user's room and make the temperature optimal in the area where the user has breakfast. The SAS will play the alarm to wake the user and then read the user's todo list for the day.

Given Conditions

-User has a set breakfast location

-SHS knows of user's commute schedule

-User's todo list is filled with testable instructions

Success

-Wakeup was triggered on SSS with respect to traffic condition on commute

-Wakeup is triggered on SSS and SSS triggers wake up on SES and SAS

-SES adjusts climate appropriately in breakfast area shortly before impending event

-SAS rings wakeup alarm and plays todo list shortly after

Failure

-Nothing happens

-Wakeup event doesn't take in account of traffic condition

-Each subsystem doesn't perform their respective tasks

-Subsystem perform tasks at incorrect times and act disjointed

roject Management and Plan of Work

A. Merging the Contributions from Individual Team Members

Subgroup 1: Akhilesh Bondlela, Vinay Shah
Subgroup 2: Brian Ellsworth, Huy Phan
Subgroup 3: Bhargav Tarpara, Nicholas Grieco
Subgroup 1 is responsible for coding the Smart Audio System.
Subgroup 2 is responsible for coding the Smart Scheduling System.
Subgroup 3 is responsible for coding the Smart Energy System.

The application will be done in Java using Android Studio.
Each subgroup work on their system, we will merge everything into one Android Application, which acts as a control center.

B. Project Coordination and Progress Report

The use cases that will be implemented, at least for the first demo, will be the most critical to our functionality. Probably the most difficult endeavor we are on is figuring out how to implement the use case in actual code. Simple designs on the user's end has lead to a complex design on the backend. We are trying to tackle how to implement our use cases so that it can be implemented in the user's day to day operations and not create other complications of its own from it. Our user interface, as pasted above, has been successfully agreed upon. We believe we have developed an intuitive approach to use the SHS. Each subsystem has their own independent tab in the app sort of like a sub-app per subgroup.

C. Plan of Work

Task	2/26	3/5	3/12	3/19	3/26	4/2	4/9	4/16	4/23	4/30	5/7
Report 2 Part 1											
Report 2 Part 2											
Report 2 Part 3											
Report 3											

Demo 1											
Demo 2											
Archive Documentation											
Server Backend Endpoints											
Database Connection to server via API											
General App UI											
SES frontend											
SAS Frontend											
Integration of Alexa API											
Integration of in home positioning API											
Prototype of an individual speaker with Alexa functionality											

D. Breakdown of Responsibilities

Name	Modules/Class Objects
Akhilesh Bondlela	SAS System Controller, Alexa Integration, Speaker Integration, Microphone Controller
Bhargav Tarpara	SES Controller, Main Activity, and Zone functions
Brian Ellsworth	SSS App Testing, Integrate SSS App to the Main App
Huy Phan	UI Design, SSS App (Coding, All classes)
Nicholas Grieco	SES App testing, Time to temperature function
Vinay Shah	SAS System Controller, User Location Tracker, System Logs, Schedule Interpreter

8. References

[1] HVAC https://en.wikipedia.org/wiki/HVAC
[2] Matlab Sim
https://www.mathworks.com/help/simulink/examples/thermal-model-of-a-house.html
[2] Newton's law of cooling https://en.wikipedia.org/wiki/Newton's_law_of_cooling
[3] Andriod https://developer.android.com/studio/index.html
[4] The State of Cooperative Learning
http://biologytransformed.org/wp-content/uploads/2014/07/Johnson20071.pdf
[5] UI Design https://www.sketchapp.com
[6] Alexa https://developer.amazon.com/alexa
[7] Google Maps API https://developers.google.com/maps/
[8] Open Weather API https://openweathermap.org/api